Gewinnung, Verwaltung und Anwendung von Performance-Daten zur Unterstützung des autonomen Datenbank-Tuning

Dissertation

zur Erlangung des akademischen Grades
Doktor-Ingenieur (Dr.-Ing.)

vorgelegt dem Rat der Fakultät für Mathematik und Informatik der
Friedrich-Schiller-Universität Jena

von
Dipl.-Inf. David Wiese
geboren am 13. September 1981 in Gera, Thüringen

Gutachter:
1. Prof. Dr. Klaus Küspert, Friedrich-Schiller-Universität Jena
2. Prof. Dr.-Ing. habil. Thomas Ruf, Universität Erlangen-Nürnberg
3. Prof. Dr. Bernhard Convent, Fachhochschule Gelsenkirchen

Tag der Einreichung: 11.05.2011
Tag der öffentlichen Verteidigung: 11.07.2011

Meiner Familie

Danksagung

An dieser Stelle ist es Zeit, denjenigen Menschen zu danken, die mich auf dem langen Weg von der Idee bis zur Fertigstellung der Dissertationsschrift begleitet und unterstützt haben.

Ein besonderes Wort des Dankes möchte ich daher zunächst an meinen Doktorvater, Prof. Dr. Klaus Küspert, richten. Seine ausgesprochen angenehme, stets kritisch hinterfragende wissenschaftliche Betreuung, seine Unterstützung sowie unzählige wertvolle fachliche und außerfachliche Ratschläge sorgten für das Gelingen der Arbeit. Neben den bereichernden Fachdiskussionen hat mich vor allem seine Förderung meiner fachlichen, beruflichen und persönlichen Weiterentwicklung bereits vom Beginn der Studienarbeit bis über den Ausschied aus dem universitären Bund hinaus begeistert. Die 5 Jahre als wissenschaftlicher Mitarbeiter an seinem Lehrstuhl waren für mich eine glückliche Zeit, vor allem aufgrund der ausgesprochen angenehmen und kollegialen Atmosphäre.

Nicht weniger zu danken gilt es meinen beiden externen Gutachtern, Herrn Prof. Dr.-Ing. habil. Thomas Ruf sowie Herrn Prof. Dr. Bernhard Convent, für die bereitwillige Übernahme des Zweit- und Drittgutachtens sowie ihre wohlwollende Begleitung und zügige Durchsetzung meines Promotionsverfahrens.

Des Weiteren gilt mein großer Dank den am Kooperationsprojekt beteiligten Mitarbeitern des IBM-Labors in Böblingen. Insbesondere seien hier Michael Reichert und Rüdiger Stumm genannt. Vor allem durch ihre Bemühungen und die administrative sowie fachliche Unterstützung konnte das gemeinsame Forschungsprojekt zum Leben erweckt und zu einem erfolgreichen Abschluss gebracht werden.

Mein Dank gebührt auch den zahlreichen Studenten, die mit ihren Leistungen in Form von Programmiertätigkeiten vor Ort in Böblingen sowie Studien- und Diplomarbeiten wesentlich zum Gelingen der Arbeit beigetragen haben.

Ein großer Dank geht aber auch an meine Lehrstuhl-Kollegen, die während meiner Zeit an der Friedrich-Schiller-Universität zu Freunden geworden sind, mich stets aufbauten und für die erforderliche Abwechslung sorgten. Sie waren mir allzeit Ansprechpartner und haben mein Dissertationsprojekt durch ihre Ideen und fundiertes Fachwissen um viele Anregungen und ihre konstruktive Kritik bereichert.

Auch meiner Freundin danke ich für ihre tatkräftige Unterstützung, das entgegengebrachte Verständnis sowie für das hohe Maß an persönlichem Verzicht, besonders in den letzten Monaten. Nicht zuletzt danke ich meiner Familie, die in jeglicher Hinsicht die Grundsteine für meinen Weg gelegt hat und ohne die ein Studium der Informatik und eine Doktorarbeit niemals möglich geworden wären. Ihnen sei die vorliegende Arbeit von ganzem Herzen gewidmet.

Norderstedt, im Juni 2011 David Wiese

Kurzfassung

In den letzten Jahrzehnten ist die Komplexität und Heterogenität von Informationssystemen rapide gestiegen. Die Folge ist, dass viele moderne IT-Systeme aufgrund ihrer heterogenen Architektur- und Applikationsvielfalt sehr kostenintensiv in der Entwicklung, fehleranfällig in der Nutzung und schwierig durch Administratoren kontrollier- bzw. konfigurierbar sind.

Neben kontinuierlicher manueller Administration und Optimierung im laufenden Betrieb bildet der Einsatz intelligenter Tools zur Systemverwaltung eine entscheidende Voraussetzung für die Gewährleistung zuverlässiger, hochverfügbarer und performanter IT-Systeme.

Initiativen wie das Autonomic Computing helfen, der steigenden Komplexität Herr zu werden, indem sie den „Problemfaktor Mensch" entlasten und Technik nutzen, um Technik zu verwalten. Durch die Anpassung bzw. Erweiterung der System-Umgebung versuchen derartige Ansätze neben derzeitiger manueller, reaktiver Performance-Optimierung, eine automatisierte reaktive und proaktive Performance-Kontrolle zu gewährleisten.

Die vorliegende Arbeit stellt zu Beginn ausgewählte autonome Funktionalitäten, Komponenten und Tools von IBM DB2, Oracle und Microsoft SQL Server als bedeutende heutige Vertreter der relationalen Datenbankwelt gegenüber und untersucht in Folge ihre Tauglichkeit in Bezug auf die Automatisierung von Optimierungs- und Administrationsaufgaben.

Zur autonomen Verwaltung von auf Datenbanken basierenden Software-Stacks sind weitreichendere Mechanismen gefordert. Zentrale Grundvoraussetzung für eine autonome Infrastruktur ist eine verlässliche, globale Daten- bzw. Wissensbasis. Wir erläutern die Bedeutung und anhand einer Klassifikation die Arten von Wissen im autonomen (Datenbank-)Tuning. Dabei zeigen wir, welchen Nutzen das verschiedenartige Wissen über das operative System und die autonomen Prozesse haben und wie es das autonome (Datenbank-)Tuning beeinflussen kann. Wir erarbeiten zudem, wie Performance-Daten über das Verhalten und den Zustand des Systems mit aus dem Data-Warehousing bekannten Techniken gesammelt, konsolidiert, verwaltet und zur Laufzeit ausgewertet werden können. Neben der Architektur und den funktionalen Komponenten eines solchen Performance Data Warehouse wird zudem dessen Datenmodell erläutert und die Anbindung an das vorausgehende Monitoring sowie die nachfolgende Analyse spezifiziert.

Mit dem Ziel, die menschliche Vorgehensweise „nachzuahmen" und somit die Administratoren bei ihren Routine-Tätigkeiten zu entlasten, widmen wir uns der Konzipierung und Beschreibung einer möglichen Infrastruktur zur Automatisierung typischer Tuning-Aufgaben. Wir erarbeiten allgemein und anhand von Beispielen, wie Tuning-Wissen und bewährte Praktiken von DBAs abgebildet, in Form von Workflows formalisiert und zur Laufzeit für die Problemlösung angewendet werden können.

Die erarbeitete Architektur verknüpft Aspekte der Workflow-orientierten Modellierung mit denen einer Ereignis-gesteuerten Ausführung von Tuning-Praktiken unter Beachtung der über eine Workload-Klassifikation ermittelten Arbeitslast. Anhand unseres in einem Kooperationsprojekt mit der IBM entstandenen Prototypen zum autonomen Datenbank-Performance-Tuning, dem Autonomic Tuning Expert, zeigen wir die Realisierbarkeit einer solchen iterativen Monitoring- und Tuning-Architektur sowie anhand von konkreten Experimenten, wie ein DB2-System von dem Ansatz profitieren und optimiert werden kann. Durch die Integration von „best Practices" für das Datenbank-Tuning können sowohl Administratoren als auch autonome Komponenten bei der Analyse, Planung, Entscheidungsfindung und Evaluation zur Laufzeit unterstützt werden.

Inhaltsverzeichnis

Abbildungsverzeichnis

Tabellenverzeichnis

Listings

Kapitel 1

Einleitung

In diesem Kapitel werden nach einer kurzen Darlegung der aktuellen Situation und der einhergehenden Probleme im System-Management sich daraus ergebende Ziele sowie der grobe Aufbau der vorliegenden Arbeit beschrieben.

1.1 Motivation

„... The obstacle is complexity … Dealing with it is the single most important challenge facing the IT industry"
(Paul Horn, Director of Research, IBM, 2001)

In den letzten Jahrzehnten wurden immer leistungsfähigere und zunehmend komplexere Softwaresysteme mit einer heterogenen Komponenten-, Architektur- und Applikationsvielfalt entwickelt, deren zentraler Bestandteil Datenbanken (DB) sind. Mit der zunehmenden Komplexität und Leistungsfähigkeit von IT-Systemen steigen nicht nur die Anforderungen an deren Verwaltung (System-Management), sondern auch die Probleme [GaCo03]. Für viele Unternehmen hat der fehlerfreie Betrieb ihrer IT-Systeme allerhöchste Priorität, da sich Ausfälle oder Leistungseinbußen unmittelbar auf die finanzielle Situation des Unternehmens auswirken. In solchen Fällen müssen Probleme schnellstmöglich erkannt und behoben werden.

Zur Gewährleistung der Zuverlässigkeit, hoher Verfügbarkeit und vor allem Performance dieser Systeme sowie der Einhaltung diverser von Managern oder Fachabteilungen vorgegebener Richtlinien (Service Level Agreements, SLA) sind, neben einer vernünftigen Ressourcenplanung, Installation und Vorab-Konfiguration, die Wartung, Administration und Optimierung unabdingbar. Die heutigen Systemen anlastende Dynamik in Form von sich ständig ändernden Anforderungen sowie Arbeits- und Umgebungsbedingungen macht es erforderlich, im laufenden Betrieb administrativ und optimierend einzugreifen.

Um einen derartigen performanten und fehlerfreien operativen Betrieb dieser Systeme zu gewährleisten, werden insbesondere in großen Unternehmen heutzutage Administratoren auf mehreren Ebenen (System-, Netzwerk-, Datenbank-Administrator, ...) eingesetzt. Deren hauptsächliche Tätigkeit besteht darin, diese Systeme zu überwachen und bei bzw. idealerweise vor auftretenden Problemen adäquat zu reagieren. Dazu müssen die Administratoren schnell und kompetent handeln. Allerdings sind sie eine teure menschliche Ressource und zudem bedingt durch Arbeitszeiten, Pausen, Krankheit etc. nicht rund um die Uhr verfügbar, u.U. nicht sofort zur Stelle und können somit oft nicht umgehend auf abnormale Situationen reagieren. Durch mangelnde Kompetenz und Erfahrung kann es im Fehlerfall zu Fehlentscheidungen und zusätzlich zu Zeitverzögerungen kommen, bis die

Ursachen, Zusammenhänge und Trade-Offs erkannt und eine entsprechende Lösung gefunden werden.

Die immer komplexer werdenden Systeme bestehen aus einer Vielzahl von Schichten und Komponenten mit diversen zu erkennenden und zu berücksichtigenden Abhängigkeiten untereinander. All diese Komponenten müssen überwacht und verwaltet werden. Beispielsweise müssen Datenbank-Administratoren nicht nur tiefgehende Kenntnisse über das eingesetzte Datenbank-Management-System (DBMS), sondern auch über die auf dem DBMS aufbauenden Applikationsschichten sowie über die Applikationsdomäne im Allgemeinen haben. Für viele der Aufgaben sind daher Spezialwissen und Schulungen nötig, um den Administratoren Wissen auf den verschiedenen Ebenen der Systemarchitektur zu vermitteln. Zudem wird bei enormen Daten-Mengen, Workload-Komplexität und vielfältigen Aufgaben schnell etwas übersehen. Indes scheitert es auch oft an der mangelnden Kommunikation und dem Wissens- bzw. Erfahrungsaustausch zwischen den Administratoren. Das liegt u.a. an den verschiedenen Ebenen der Betrachtung und Begriffsbildung. Einem Netzwerkadministrator werden verständlicherweise die Interna eines DBMS eher fremd erscheinen. Auch Nachwuchs ist nur langwierig anlernbar, da die „alten Hasen" oft über Jahre hinweg eigene, undurchschaubare, nicht-standardisierte und spärlich dokumentierte Skripte und Tools für ihr Aufgabenspektrum entwickelt haben.

Die Anforderungen an die Kompetenz, Anzahl und Lernfähigkeit der Fachkräfte und damit die laufenden Betriebskosten steigen somit beträchtlich. Manuelle Administration ist somit nicht nur zeitaufwendig, kostspielig und fehleranfällig, sondern auch in den meisten Fällen nur noch durch erfahrene und gut ausgebildete Spezialisten durchführbar. Erfahrene Administratoren sind jedoch rar und tragen wesentlich zur Total Cost of Ownership (TCO) des Systems bei.

Die systeminhärente Gesamt-Komplexität sowie die schwer übersehbare Vielfalt an Installations- und Konfigurationsoptionen heutiger moderner IT-Landschaften sind also durch bloße menschliche Intervention kaum noch kontrollier- und handhabbar [KeCh03].

Die aufgeführte Problematik moderner IT-Landschaften erfordert daher offensichtlich neben kontinuierlicher manueller Administration und Optimierung im laufenden Betrieb insbesondere den Einsatz intelligenter(er) Tools und Komponenten mit erweiterten Aufgaben zur Unterstützung der automatisierten Systemüberwachung und -verwaltung. Moderne DBMS bringen bereits eine Reihe integrierter automatischer Leistungsverbesserungs- und Administrationsvereinfachungsmechanismen mit. Wir werden in Abschnitt 3.1 dieser Arbeit näher darauf eingehen. Für eine wirklich differenzierte und vor allem das ganze System umfassende Leistungssteigerung reichen diese Mechanismen aber dennoch oft nicht aus.

Ansätze wie das *Autonomic Computing* helfen das Problem der steigenden Komplexität zu adressieren, indem sie Technik nutzen, um Technik zu verwalten [Hor01]. Vorbild des Autonomic Computing ist das vegetative Nervensystem des Menschen. Es kontrolliert wichtige Körperfunktionen, wie Atmung, Puls und Blutdruck, ohne dass wir bewusst eingreifen müssen. Analog zum menschlichen Organismus werden beim Autonomic Computing wichtige Parameter „unwillkürlich" überwacht und es wird bei Abweichungen mit den richtigen Maßnahmen selbstständig (autonom) reagiert. Das (Langzeit-)Ziel ist die Selbstregulation von Computersystemen, die flexibel auf sich ändernde Arbeits- und Umgebungsbedingungen reagieren sollen, ohne ständige Eingriffe menschlicher Administratoren zu erfordern. Dadurch werden Administratoren von ihren Routineaufgaben entlastet und befähigt, sich auf langfristige, strategische Aufgaben zu konzentrieren.

Mit dem Ziel, die menschliche Vorgehensweise „nachzuahmen“ und somit den Datenbank-Administrator bei dessen Routine-Tätigkeiten zu entlasten, wurde 2005 ein Kooperationsprojekt zwischen der IBM Böblingen und dem Lehrstuhl für Datenbanken und Informationssysteme der Friedrich-Schiller-Universität Jena ins Leben gerufen. Unser im Rahmen dieses Projekts entstandener und in der Arbeit beschriebener *Autonomic Tuning Expert* (ATE) kombiniert zentrale Konzepte des Autonomic Computing mit dem Konzept der Formalisierung und der automatischen Ausführung bewährter Standard-Tuning-Praktiken. Auf dieser Grundlage wird es möglich, sowohl das Verhalten eines zu optimierenden Systems als auch das des selbst-optimierenden (Black-Box-)Aufsatzes zu evaluieren und weitere konzeptionelle und prototypische Untersuchungen zum autonomen Datenbank-Tuning anzustellen.

Grundvoraussetzung für eine solche automatisierte bzw. autonome Selbst-Optimierung ist die kontinuierliche bzw. Ereignis-gesteuerte Sammlung, Speicherung und Verwaltung von Performance- und Metadaten über den Zustand sowie das Verhalten des Systems bzw. die Charakteristika und Auswirkungen der Workload. Eben nur durch eine solche den Anforderungen und Gegebenheiten angepasste Monitoring-Strategie wird eine manuelle bzw. automatische Problem-Erkennung und -Diagnose und folglich auch eine effiziente Problem-Auflösung erst ermöglicht.

In der Praxis kommen die für das System-Management relevanten Daten meistens nicht aus einem zentralen System, sondern sind auf verschiedene Subsysteme, oftmals redundant, verteilt. Verschiedenste System-interne bzw. von Drittanbietern angebotene Tools sammeln, visualisieren und speichern die Performance-Daten. Die Abfrage nach bestimmten Informationen ist durch diese Fülle an Daten verschiedenen Ursprungs und Formats schwierig realisierbar und fehlerbehaftet, da verschiedene Datenquellen zudem unterschiedliche Ergebnisse liefern können. Informationsverlust und Widersprüche sind damit nicht ausgeschlossen. Die relevanten Daten aus diesen heterogenen, internen und externen Quellen müssen idealerweise also erst einmal sinnvoll zusammengestellt, bereinigt, geordnet und an zentraler Stelle restrukturiert, redundanzfrei, konsistent bereitgestellt werden, um eine vernünftige und flexible Analyse sowie die Verknüpfung von Informationen zu erlauben. Daher bietet sich die Untersuchung und Übertragung von Techniken und Konzepten des in der Geschäftsdatenwelt und in fast allen Wirtschaftszweigen allgegenwärtigen *Data Warehousing* auf das System- und Performance-Management an. Der Nutzen, den diese Systeme bieten, hat zu ihrer großen Verbreitung beigetragen. Data Warehouses übernehmen in diesem Schritt eine entscheidende Rolle. Sie sollen die unterschiedlichsten (Unternehmens-)Daten für die weitere Verarbeitung und vor allem Entscheidungs-unterstützende Analyse integriert, thematisch gruppiert bereitstellen und so Qualität sowie Integrität des Datenmaterials gewährleisten.

In einem einheitlichen Langzeitarchiv (*Performance Data Warehouse*) soll schließlich eine Historie aller bisherigen Performance-Kennzahlen zusammen mit erkannten Problemen protokolliert und mit den daraufhin ausgeführten Problemlösungen korreliert werden. In Folge können die gewonnenen Daten einem Administrator oder System-verwaltenden Komponenten für weitere Analysen, zur Wissens- und Erfahrungs-gestützten Planung und Entscheidungsfindung zur Verfügung stehen. Um mit der ständig anwachsenden Menge gesammelter und gespeicherter Performance-relevanter Daten zurechtzukommen, werden darüber hinaus automatisierte Verfahren zur Verwaltung und Analyse benötigt. Insbesondere das *Data Mining* bietet Werkzeuge, die scheinbar zusammenhanglosen Daten nach noch nicht bekannten, wissenswerten Zusammenhängen durchsuchen, Daten aufspüren, kombinieren und neue Informationen zur Verfügung stellen.

Die Konzipierung und Beschreibung einer auf einem Performance Data Warehouse basierenden Datensammlungs- und -verwaltungs-Architektur zur Modellierung, Gewinnung, Speicherung, Verwaltung und Anwendung von Performance- und Metadaten für das manuelle bzw. autonome Datenbank-Performance-Tuning sowie zur Analyse und Planung bilden den Schwerpunkt der vorliegenden Arbeit.

1.2 Ziele der Arbeit

Die wichtigsten Ziele und wesentlichen Beiträge der vorliegenden Arbeit können wie folgt zusammengefasst werden:

Heranführen an das autonomes Tuning in und von Datenbank-basierten Systemen

Mit dem Ziel der Unterstützung und Entlastung der Datenbank-Administratoren bei der System-Verwaltung und -Optimierung soll die Arbeit anfänglich die Übertragung der Möglichkeiten zur Selbst-Optimierung auf die Überwachung und Steuerung von Datenbank-basierten Systemen andiskutieren. Hierfür werden zum einen grundlegende Herangehensweisen unterschieden, die eine Selbst-Verwaltung und insbesondere die Selbst-Optimierung von und in Datenbanksystemen ermöglichen. Zum anderen stellen wir ausgewählte Prototypen aus der Forschung sowie autonome Funktionalitäten der bedeutenden Vertreter heutiger relationaler Datenbanksysteme gegenüber.

Identifikation und Klassifikation der Metadaten im ADPT

Es sollen die im autonomen Datenbank-basierten Performance-Tuning (ADPT) anfallenden bzw. notwendigen Metadaten formalisiert, klassifiziert und beschrieben werden. Dabei ist zwischen Metadaten aus der Vergangenheit, Gegenwart und Zukunft über die (1) vergangene, gegenwärtige und erwartete Workload, das (2) zu verwaltende System, seinen Status und sein Verhalten sowie (3) über das Tuning-System selbst zu unterscheiden. Als eine spezielle Form der Metadaten werden hierbei vor allem Performance-Daten (Messwerte) gezielt betrachtet.

Erstellung eines Wissensmodells für das autonome Datenbank-Tuning

Die identifizierten Metadaten sollen als Grundlage für die Konstruktion eines Wissensmodells für das autonome Datenbank-Tuning dienen. Wissen über das System in seiner Umgebung und über die Workload kann als zentraler Bestandteil zum Aufbau und Betrieb eines Systems zum autonomen Tuning sowie zur intelligenten Planung und Entscheidungsfindung angesehen werden.

Einführung in die Formalisierung von Experten-Tuning-Wissen

Wir möchten in den Grundzügen aufzeigen, warum und wie Informationen über bewährte Datenbank-Tuning-Praktiken aus vielfältigen unstrukturierten Quellen formalisiert, materialisiert und maschinell auswertbar Komponenten zum autonomen Tuning zur Verfügung gestellt werden können. Die Idee besteht darin, ein autonomes Tuning-System in Analogie zur menschlichen Vorgehensweise auf erkannte Problem-Situationen mit bewährten Tuning-Maßnahmen reagieren zu lassen.

Beschreibung der Performance-Daten-Gewinnung, -Verwaltung und -Anwendung

Es soll aufgezeigt werden, wie relevante Daten periodisch bzw. Ereignis-gesteuert gesammelt bzw. generiert, um Kontext-Informationen angereichert, von Fehlern oder Inkonsistenzen bereinigt, transformiert und schließlich einem zentralen Repository zugeführt werden können. Das konsolidierende Repository hat die Aufgabe, die zum

Zweck der Analyse adäquat vorbereiteten Daten zu speichern. Zudem sollen Szenarien vorgestellt werden, wie die aufbereiteten, an zentraler Stelle gespeicherten Daten manuell bzw. automatisch gefiltert, ausgewertet und somit dem autonomen Tuning sowie den Administratoren bereitgestellt werden können.

Motivation und Beschreibung eines multidimensionalen Performance Data Warehouse

Als zentraler Aspekt der vorliegenden Arbeit sollen neben der Architektur und den funktionalen Komponenten eines Performance Data Warehouse (PWH) auch dessen Datenmodell erläutert und die Anbindung an das vorausgehende Monitoring sowie die nachfolgende Analyse spezifiziert werden. Das PWH stellt alle Informationen zusammen mit ihrem Kontext zentralisiert, integriert und aktuell in Anwendungsneutraler, wiederverwendbarer Form bereit, um die zugrunde liegenden Ressourcen, vollständig, d.h. nach allen relevanten Perspektiven (Dimensionen), überwachen und analysieren zu können. Über Data Marts bspw. werden je nach Benutzer- und Analyseanforderungen verschiedene Formen der Aufbereitung und Darstellung unterstützt. Darüber hinaus sollen typische Anwendungs-Szenarien zur manuellen bzw. automatischen Nutzung sowie zur flexiblen, multidimensionalen Analyse und Navigation innerhalb der restrukturierten Performance-Daten vorgestellt werden.

Beschreibung der System- und Tuning-unabhängigen Workload-Erkennung

Um ein Workload-orientiertes, adaptives Monitoring und Tuning zu ermöglichen, soll ein vom konkreten System und dem Tuning unabhängiges Verfahren zur Klassifikation der aktuell anliegenden Workload entwickelt und vorgestellt werden.

Skizzierung eines System- und Ressourcen-Modells für das ADPT

Eine entscheidende Charakteristik autonomer, selbst-verwaltender Systeme ist das Bewusstsein über den Aufbau und die Wirkungsweise ihrer Komponenten und der (dynamischen) Umgebung. Dazu soll die Darstellung und Nutzung von u.a. topologischen Informationen über einzelne Ressourcen, ihr Verhalten sowie ihre Abhängigkeiten und Beziehungen untereinander skizziert werden. Ein solches allgemeines Verständnis über die Ressourcen und deren Abhängigkeitsbeziehungen kann als gemeinsame Wissens-Basis für viele der Entscheidungsprozesse dienlich werden.

Nachweis der Realisierbarkeit des Workload- und best-practices-orientierten Datenbank-Tuning

Es soll die Realisierbarkeit der in der Arbeit beschriebenen Infrastruktur zur Automatisierung typischer Datenbank-Tuning-Aufgaben unter Minimierung menschlicher Interaktion anhand eines Prototypen (Autonomic Tuning Expert, ATE) nachgewiesen und ihre Tauglichkeit anhand von Testläufen exemplarisch demonstriert werden. Dazu ist ein Formalisieren, Speichern, Verwalten, individuelles Anpassen und Anwenden von Tuning-Wissen erforderlich. Dieser Ansatz scheint vielversprechend, da vielfältige Informationen über bewährte Datenbank-Tuning-Praktiken (best practices) in elektronischer Form zur Verfügung stehen, z.B. in News-Groups, IT-Blogs, Online Magazinen, „IBM Redbooks“-Veröffentlichungen oder Produkthandbüchern.

Konzeption und Beschreibung einer Monitoring- und Tuning-Referenzarchitektur

Zu den wesentlichen Zielen der Arbeit zählen die Erarbeitung und Vorstellung von Anforderungen und Konzepten für eine mögliche Architektur zum adaptiven, von der Workload und Richtlinien abhängigen Sammeln, Verwalten und Anwenden von Performance-Daten. Ziel ist die Unterstützung der Administratoren bzw. der autonomen Komponenten bei der Analyse, Planung, Entscheidungsfindung und Evalua-

tion. Darüber hinaus soll die Integration in die bestehende ATE-Infrastruktur zum automatisierten Performance-Tuning aufgezeigt werden.

1.3 Gliederung der Arbeit

Im **Kapitel 2** werden zunächst grundlegende Begriffe, Grundlagen und Konzepte zum Performance Management, Autonomic Computing sowie Data Warehousing und Data Mining beschrieben.

Kapitel 3 stellt daraufhin vor, welche Selbstverwaltungs-Funktionalitäten heutige RDBMS-Produkte bereits unterstützen und wie man ein bestehendes (Datenbank-)System um Mechanismen zum autonomen Performance-Monitoring und -Tuning erweitern kann. Daneben werden einige, sowohl in Form vereinzelter Konzepte als auch in Form von Prototypen aus der Forschung, konkret existierende Ansätze kurz vorgestellt und theoretischen Herangehensweisen für eine Selbst-Verwaltung zugeordnet.

Kapitel 4 widmet sich der Identifikation und Klassifizierung von Wissen, als zentralem Grundbaustein im autonomen Datenbank-Tuning. Neben dem Wissen über Probleme und Vorgehensweisen zur Lösung, steht hierbei vor allem das Wissen über das System in seiner Umgebung und über die Workload zur Unterstützung der Planung und Entscheidungsfindung im Mittelpunkt. Die verschiedenen Themen- und Anwendungs-Bereiche werden kategorisiert und in einem Wissensmodell als Grundlage für nachfolgende Implementierungen festgehalten.

Nach dem Einblick in die verschiedenen Performance- und Metadaten zum autonomen Tuning soll das **Kapitel 5** die gängigsten Anforderungen an typische Performance-Daten-Quellen im Datenbank-Umfeld, potentielle Möglichkeiten zur Datengewinnung und Schnittstellen sowie Datenformate umreißen.

Im Anschluss an die Darstellung der Möglichkeiten zur Gewinnung von Performance-Daten aus den verschiedenen Quellsystemen, wird in **Kapitel 6** dargelegt, wo und vor allem wie die Informationen adäquat zu verwalten und Nutzern oder Folgeprozessen aufgabengerecht bereitzustellen sind. Wir werden zunächst die grundsätzlichen Anforderungen und in Folge eine mögliche Umsetzung in Form von Struktur und funktionalen Komponenten eines konsolidierten Repository zur Speicherung und Aufbereitung von Performance-Daten aus verschiedenen, möglicherweise heterogenen Quellen beschreiben. Insbesondere Techniken des Data Warehousing ermöglichen es unserem Performance Warehouse, sämtliche Informationen über das System und sein Verhalten derart zu hinterlegen, dass sie nach allen relevanten Perspektiven im Kontext überwacht und analysiert werden können.

Mittels entsprechender Werkzeuge können die zentralisierten Performance-Daten schließlich flexibel analysiert und zur Entscheidungsunterstützung ausgewertet werden. In **Kapitel 7** werden dahingehend mögliche Anwendungsgebiete und Verknüpfungsmöglichkeiten der Daten eines Performance Warehouse als Planungs- und Entscheidungsgrundlage im Umfeld des manuellen sowie des autonomen Datenbank-Performance-Tuning beschrieben.

Kapitel 8 schließlich fasst die bisherigen Erkenntnisse zusammen und stellt eine Referenz-Architektur für das manuelle bzw. autonome Datenbank-Performance-Monitoring und -Tuning vor. Zusätzlich zur Konzipierung umfasst das Kapitel auch unsere prototypische Realisierung und Evaluierung der entwickelten und dargestellten Konzepte. Der Prototyp namens Autonomic Tuning Expert kombiniert zentrale Konzepte des aus der Kontrolltheorie bzw. dem Autonomic Computing bekannten Feedback-Ansatzes mit der Idee der

Formalisierung bewährter Tuningpraktiken und der System- bzw. Tuning-unabhängigen Klassifikation der Workload, um ein Datenbank-basiertes Software-System autonom zu tunen.

Im Anschluss daran fasst **Kapitel 9** die vorliegende Arbeit sowie gewonnene Erkenntnisse zusammen und gibt einen Ausblick auf mögliche weiterführende Themen.

Hinweise:

Die in der Arbeit präsentierten Vorgehensweisen sind zum Teil am Beispiel von IBM DB2 aufgezeigt, aber nicht auf ein bestimmtes Datenbanksystem beschränkt. Natürlich sind im Detail einige Unterschiede zu finden, die aber nur kleine Änderungen an den vorgestellten Konzepten hervorbringen.

Viele der im Text verwendeten englischen Fachbegriffe werden nicht ins Deutsche übertragen, um die Nachvollziehbarkeit, Wieder-Erkennung und Lesbarkeit nicht zu beeinträchtigen.

Abschließend noch ein Hinweis zur Aktualität der vorliegenden Arbeit: In der Arbeit enthaltene Aussagen über die aktuelle Version von DB2, Oracle, MS SQL Server oder die aktuelle Version eines sonstigen Produkts beziehen sich auf den Stand vom August 2010.

Kapitel 2

Grundlagen

In diesem Kapitel werden die für die weiteren Betrachtungen notwendigen und hilfreichen Begriffe, Grundlagen und Konzepte beschrieben. Nach einem Überblick über das Performance Management und das (Datenbank-)Performance-Tuning im **Abschnitt 2.1**, beschäftigt sich **Abschnitt 2.2** mit dem Autonomic-Computing-Paradigma. Schließlich erfassen die nachfolgenden beiden **Abschnitte 2.3** sowie **2.4** generelle, zum Verständnis beitragende Aspekte des Data Warehousing sowie des Data Mining.

In der nachfolgenden Übersicht wollen wir kurz motivieren, welche eigenen bzw. betreuten Arbeiten bereits im Grundlagen-Kapitel referenziert werden und warum dies geschieht.

Abschnitt 2.1 - Performance Management

In [Wie05] legen wir den Grundstein für die im Abschnitt 2.1 beschriebenen Phasen des Performance Tuning. Hierbei handelt es sich im Grunde um eine Aufarbeitung von Informationen verschiedener Quellen mit dem Ziel einer vereinfachten, verständlichen Darstellung der Sachverhalte.

Im Rahmen von [WiRa07] untersuchen wir für verschiedene Bereiche des Datenbank-Tuning typische Problemsituationen und etwaige Lösungsansätze. Insbesondere in [Hen07] befassen wir uns mit Datenbank-spezifischen Diagnose- und Tuning-Techniken. Darauf basierend wird in Abschnitt 2.1 auf das ermittelte Standard-Vorgehen sowohl im Datenbank-Tuning, als auch in der Medizin und der Versuch der Übertragbarkeit über eine entsprechende Referenz aufmerksam gemacht. In [Ehl07] umreißen wir zunächst Ursachen für Performance-Probleme in Datenbanken sowie Möglichkeiten zur Detektion. Auch in [Roe08] beleuchten wir Szenarien für das Tuning von Datenbank-Lösungen näher. Wir zeigen in Abschnitt 2.1 jedoch lediglich kurz die in der Veröffentlichung ausführlich beschriebenen Möglichkeiten zur Behandlung von Wartesituationen auf Sperren auf.

Die in den Veröffentlichungen und der vorliegenden Arbeit dargelegten Beispiele von konkreten Problemen und Tuning-Algorithmen in DB2 sollen letztlich eine Grundlage zur Definition eines ausreichenden Befehlssatzes zur Beschreibung von Tuning-Plänen (siehe Abschnitt 4.2) darstellen. Die Ergebnisse umfangreicher Tool-Untersuchungen aus [Ehl10, Hen07, Wei10, Wie05] finden zudem Einzug in Abschnitt 5.3, mit dem Fokus auf Performancedaten-Quellen im DB2-Umfeld. Abschnitt 8.3.2 schließlich zeigt sowohl einige der entstandenen Tuning-Plan-Umsetzungen, als auch die Evaluierung derer automatischer Anwendung mittels unseres ATE-Prototypen.

Abschnitt 2.2 - Autonomic Computing

Neben den für das weitere Verständnis notwendigen Grundlagen beinhaltet Abschnitt 2.2 auch einen Verweis auf [Hen07]. Hierbei wird der Leser auf eine umfangreichere Gegenüberstellung des manuellen (Datenbank-)Tuning mit dem Konzept der rückgekoppelten MAPE-Schleife hingewiesen.

Abschnitt 2.3 - Data Warehousing

Mit dem Hintergrund der Übertragbarkeit auf unser Szenario, umfasst [Wie05] ausgewählte Grundlagen des Data Warehousing. Abschnitt 6.3 der vorliegenden Schrift präsentiert

daraufhin das Konzept des Performance Warehouse sowie für uns relevante multidimensionale Strukturen. Erste Vorarbeiten zur letzteren Thematik finden sich bereits in [Wie06, Wie09]. Abschnitt 7.2 zeigt schließlich die Anwendung des eingeführten multidimensionalen Datenmodells im Kontext des autonomen Performance-Tuning zur Diagnose von Performance-Problemen in Aktion.

Abschnitt 2.4 - Data Mining

Die in [Göb08] ergründete Kategorisierung und Vorstellung von Data-Mining-Techniken dient Abschnitt 2.4 als Voraussetzung. Die darin und auch in [Kra08] eingehend beschriebenen Techniken finden vor allem in den Abschnitten 7.3 sowie 7.4 der vorliegenden Arbeit eine Übertragung auf das autonome Datenbank-Tuning, mit dem Fokus auf der Erkennung der anliegenden Workload basierend auf der Klassifikation.

2.1 Performance Management

„Measurements are the key.
If you cannot measure it, you cannot control it.
If you cannot control it, you cannot manage it.
If you cannot manage it, you cannot improve it.“
(J.H. Harrington, New York, 1991)

Nahezu jedes IT-System stellt heute im besonderen Maße hochgradig komplexe Anforderungen an die Leistungsfähigkeit seiner Komponenten. Das mehrere Aktivitäten umfassende Performance Management, als Teildisziplin des System Management, spielt dabei eine entscheidende Rolle zur Leistungssteigerung unter Reduzierung des Einsatzes zusätzlicher Hardware-Komponenten, wie CPU, RAM, Festplatten o.ä. Es wird angewendet, um Anschaffungs- und Instandhaltungskosten gering zu halten, um die Anwenderzufriedenheit und die Akzeptanz der Systeme zu steigern und um finanzielle Vorteile zu erzielen. Nur bei guten Antwortzeiten und hohem Gesamtdurchsatz können Benutzer motiviert und effizient mit der Anwendung arbeiten. Ein langsames System führt zu Ausfallzeiten und Frustration, zu Mehrarbeit oder gar finanziellen Verlusten.

Der nachfolgende Abschnitt führt ein in das Performance Management mit der zugrundeliegenden Terminologie und vertieft die Methodiken des Performance-Tuning sowohl in Theorie als auch anhand ausgewählter Datenbank-spezifischer Performance-Diagnose-Techniken und -Tuning-Szenarien.

2.1.1 Terminologie und Grundlagen

IT-Systeme sind in der Regel mehrschichtig und unterliegen meist einem der **Abbildung 2.1** ähnlichen, modularen Aufbau. Hierbei stellt jede Schicht der darüber liegenden Schicht über eine abstrahierende Schnittstelle ihre Funktionalitäten und demzufolge auch ihre Ressourcen zur Verfügung.

Der Begriff **Ressource** (oder gelegentlich auch Komponente) sei weit gefasst. Darunter sind sowohl *physische* Hardwareressourcen, wie Prozessor (CPU), Hauptspeicher (RAM), Festplatte und Netzwerk, als auch darauf aufbauende *logische* Ressourcen, wie Caches/Puffer, Prozesse, Services, Threads zu verstehen. Zu letzteren zählen u.a. Datenbank-Objekte, wie Tabellen, Indexe, Tablespaces und andere abstrakte Objekte, die auf der jeweiligen Schicht zur Verfügung stehen.

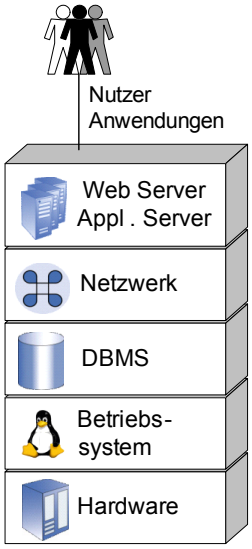


Abbildung 2.1: Aufbau eines typischen Datenbank-basierten Software-Stack

Die **Performance** eines Daten-verarbeitenden Systems beschreibt sich typischerweise durch die Art und Weise, wie seine Komponenten unter einer bestimmten Auslastung und unter gegebenen Anforderungen Aufgaben allgemein oder auf gewisse Weise ausführen. Der Begriff Performance suggeriert bereits, dass neben den Anforderungen und dem Kontext vor allem die Zeit eine wesentliche Rolle spielt und man sich zur Realisierung der Auswirkungen von Performance-Problemen vor Augen zu führen hat, wo und wofür Zeit ver(sch)wendet wird.

Die Performance wird anhand von Performance-Maßzahlen gemessen und durch die vorhandenen Ressourcen, deren Auslastung, die Anzahl und Typen konkurrierender Nutzer sowie deren Workload beeinflusst. Insbesondere sind es die verschiedenen Nutzertypen, die maßgeblich direkten und indirekten Einfluss auf die Leistung eines Systems haben:

- *End User* mit ihren Erwartungen und Anforderungen (komfortable GUI, schnelle Antwortzeiten, permanente Verfügbarkeit, ...)
- *Anwendungsentwickler* mit ihren Qualifikationen (Verständnis für interne Zusammenhänge und Konsequenzen, Vorgehensmethoden, ...)
- *Administratoren* mit ihrer Expertise, ihrem Qualitätsanspruch und ihren Erfahrungen (Planung, Kontrolle, optimierte Ressourcennutzung, Nutzung effizienter Administrator-Werkzeuge, ...)

Ist das Zusammenspiel der einzelnen Ressourcen nicht ideal abgestimmt oder wird die Leistungsgrenze einzelner Komponenten erreicht, können ungewünschte Wartesituationen und Durchsatzprobleme entstehen, die sich negativ auf die Leistung auswirken. Allgemein ist ein derartiges **Performance-Problem** gekennzeichnet durch eine nicht tolerierbare Abweichung (Ist-Zustand) von einem geforderten Soll-Zustand. Das Performance-Problem stellt sich i.d.R. durch eine Reihe von Symptomen dar. Ziel sollte es sein, auf Basis solcher beobacht- und messbarer Eigenschaften, nicht allein die Auswirkungen, sondern vielmehr die Ursache(n) eines Problems (root cause) zu „bekämpfen“.

Der Ist-Zustand einer IT-Lösung wird in der Regel durch eine oder eine Menge von **Performance-Metriken**, sog. Performance-Indikatoren, charakterisiert. Man unterscheidet in der Regel zwischen eher abstrakten, verständlicheren High-Level- und spezialisierten, technisch-orientierten Low-Level-Performance-Maßen. Zu ersterer Gruppe zählen bei-

spielsweise mittlere Antwortzeiten, Durchsatzkennzahlen, Verfügbarkeit und Datenbankzeit.

Unter der *Antwortzeit* versteht man die Zeitspanne vom Absenden der Anfrage bis zur Ausgabe des ersten Ergebnisses. Dabei kann sich der Abarbeitungsweg vom Browser auf dem Client, über das Netzwerk und einen Applikations-Server bis hin zum DBMS und wieder zurück zum Nutzer erstrecken (siehe **Abbildung 2.2**). Der *Durchsatz* ist die Anzahl bearbeiteter Anfragen pro Zeitintervall. Die *Verfügbarkeit* schließlich ist das Maß, zu welchem ein System bestimmte Anforderungen innerhalb eines vereinbarten Zeitrahmens erfüllt. Sie gibt bspw. die Erreichbarkeit des Systems, z.B. bezogen auf ein Jahr, wieder. Um genauer lokalisieren zu können, wo ein Performance-Problem liegt und damit auch, an welcher Stelle eine Optimierung ansetzen sollte, ist das Maß der *Datenbankzeit* zudem recht gebräuchlich. Sie ist nach [DRS+05] definiert als Summe der Zeit, die innerhalb des DBMS für Nutzerprozesse aufgewendet wird. Vom Prinzip her ist sie damit ähnlich der Antwortzeit, allerdings mit der Einschränkung, dass dieses Maß nur die Zeit und Arbeit innerhalb des DBMS, aber nicht etwa im Applikations-Server oder Browser, misst. Im Wesentlichen setzt sich diese Datenbankzeit, neben dem Herstellen/Beenden der Verbindung zur Datenbank, aus der Ausführung von SQL-Anweisungen und der (Rück-)Übertragung der Ergebnisse zusammen.

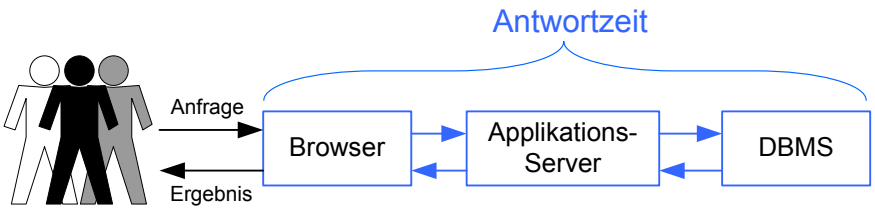


Abbildung 2.2: Veranschaulichung der Antwortzeit

Zu den Low-Level-Performance-Indikatoren in DBMS können u.a. Auslastungsgrade, Füllgrade von Speicherbereichen sowie die (Buffer) Cache Hit Ratio gezählt werden. Letztere gibt an, wieviel Prozent der gelesenen Seiten im Cache gefunden wurden und nicht von einer Festplatte gelesen werden mussten. Da Cache-Zugriffe um ein Vielfaches schneller als I/O-Zugriffe sind, wird im Allgemeinen eine möglichst hohe Buffer Cache Hit Ratio angestrebt.

Soll-Zustände werden im Rahmen von Policies bzw. Service Level Agreements (siehe Abschnitt 4.2.1) durch verbindliche **Performance-Ziele** bzw. Anforderungen an ein System und an dessen Performance-Metriken charakterisiert. Sie garantieren nicht zuletzt dem Nutzer ein bestimmtes Maß an Leistung. Sollte diese nicht (mehr) erbracht werden, müssen die Ursachen identifiziert und die Fehler behoben werden. Alternativ können im Rahmen der Optimierung auch bisher ungenutzte Leistungspotentiale erschlossen werden, um so die Performance-Ziele zu erreichen.

> *„Without well-defined performance objectives, performance is a hit-or-miss exercise, with no way of delivering on any service level agreements that may be negotiated with users."* [AGK+04]

In Anlehnung an [AGK+04] müssen diese Performance-Ziele folgende Eigenschaften aufweisen:

- *Realistisch*, so dass gesetzte Ziele mit dem aktuellen Stand der Technik auch erreicht werden können.

- *Vernünftig*, so dass überhöhte Zielvorstellungen keine unnützen Kosten verursachen und nicht alles technisch Mögliche auch als notwendig und sinnvoll erachtet wird.
- *Quantifizierbar*, so dass bei der Zieldefinition nur quantitative Metriken (Zahlen, Verhältnisse, Prozentsätze) verwendet werden. Qualitative Ausdrücke, wie „gut", „sehr gut" sind eher subjektiv und nicht eindeutig.
- *Messbar*, so dass die Zielkonformität überprüft werden kann. In Abgrenzung zur Quantifizierbarkeit ist hier ausschlaggebend, dass der Zustand der Performance-Ziele auch praktisch messbar ist und nicht nur die theoretischen Voraussetzungen dafür erfüllt sind.

Bei der konzentrierten Nutzung von Low-Level-Performance-Zielen und damit der Bewertung mittels Low-Level-Performance-Metriken besteht die große Gefahr, sich in Details zu verlieren und diese zu optimieren, obwohl ihr Einfluss auf die für die Nutzer relevanten High-Level-Ziele nicht gegeben sein muss. Generell ist deswegen die Verwendung von High-Level-Performance-Zielen anzuraten.

Kriterien für eine gute Performance sind abhängig vom Kontext des überwachten Systems. Mit variierendem Anwendungsfällen können demnach sowohl die betrachteten Systemkenngrößen als auch die dafür zu messenden Werte sehr unterschiedlich sein. Das Ziel eines Online-Versandhauses ist es bspw., den Durchsatz an täglichen Transaktionen zu maximieren. In einem Analyse-System eines Marktforschungsinstituts hingegen ist die Ausführungsdauer der wenigen komplexen Anfragen von essentieller Bedeutung. Diese sollte minimal sein.

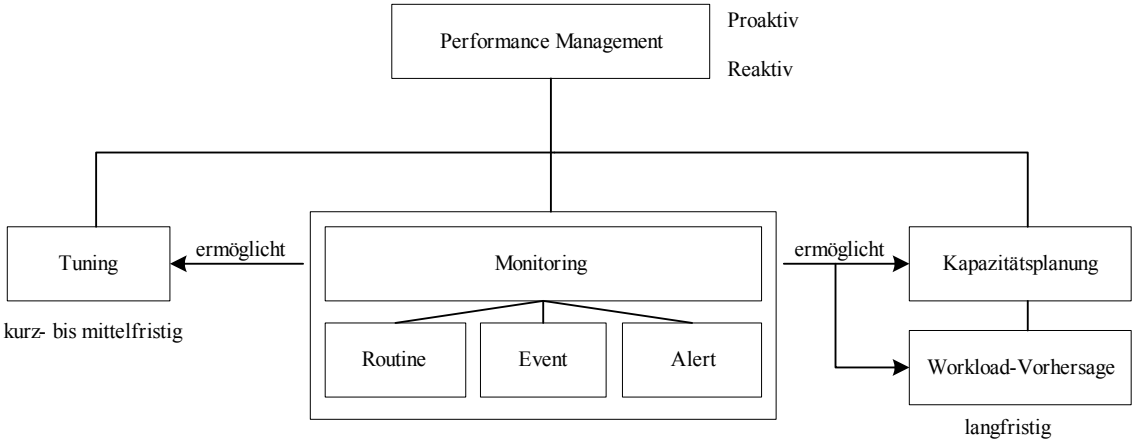


Abbildung 2.3: Performance Management in der Übersicht

Um das System am „Laufen" zu halten, die durch die Erwartungen der Benutzer angestrebte Soll-Performance im operationalen Betrieb zu erreichen, Performance-Probleme rechtzeitig zu erkennen bzw. idealerweise vorherzusagen und schließlich zu beheben, ist daher ein effizientes **Performance Management** essentiell. Von den am System-Management beteiligten Instanzen (Applikations-Entwickler sowie System- und Datenbank-Administratoren) muss eine fundierte, gemeinsame, sowohl durch mittel- bis langfristige Vorausschau proaktive, als auch kurzfristig auf kritische Ereignisse reaktive Strategie im Rahmen des Performance Management entwickelt und eingehalten werden. Wie in **Abbildung 2.3** ersichtlich, beinhaltet es als Kernpunkt und Grundvoraussetzung die Sammlung, Speicherung und Verwaltung von Performance-relevanten Daten (Monitoring). Hierbei können existierende Performance-Tools verwendet, jedoch müssen permanent die aktuellen System-Dynamiken und -Erkenntnisse, aber vor allem verschiedene Zeithorizonte berücksichtigt werden. Kurz- bis mittelfristig sollen durch das darauf aufbauende

Tuning u.a. Anomalien erkannt, ihre Ursachen diagnostiziert und behoben werden. Langfristig sollen der (normale) Systemzustand, Abweichungen, Trends, die Workload und Abhängigkeiten zwischen den Komponenten erkannt bzw. vorhergesagt werden, um dadurch insbesondere frühzeitige (Kapazitäts-)Planungen und Problemlösungen zu ermöglichen.

Abgesehen von diesen generellen und typischen Szenarien können natürlich außerordentliche Umstände ebenso eine Optimierung erfordern, so z.B. das Einbinden von neuer Hard- oder Software sowie eine starke Erhöhung der Anzahl an Systemnutzern.

2.1.2 Performance Tuning

Mit **Performance Tuning** (kurz: Tuning) ist hier das Vorgehen gemeint, wie der Sollzustand, sprich die gewünschte Performance, durch Änderungen an einem System (wieder) erreicht und somit das Verhalten des Systems auf die Workload verbessert werden soll. Tuning wird demnach keineswegs als Selbstzweck betrieben, sondern ist stets darauf ausgerichtet, die Anforderungen der Geschäftsprozesse zu erfüllen, Performance zu erhalten bzw. zu steigern.

> *„One of the main objectives of an IT organization is to ensure that its infrastructure delivers the required performance to ensure that business objectives are continuously met in a constantly evolving and changing business environment."* [AGK+04]

Besonders augenscheinlich spiegelt sich auch im Performance-Tuning das ökonomische Prinzip bei zwei idealisierten Ansätzen wider. Zum einen der „Ansatz des minimalen Arbeitsaufwands", bei welchem eine akzeptable Performance mit minimalem Einsatz bereitgestellt werden soll. Hierbei handelt es sich um einen eher reaktiven Ansatz, bei dem das System sich weitestgehend selbst überlassen wird, solange keine Fehlermeldungen und Beschwerden auftreten. Das Gegenteil stellt der „Ansatz der maximalen Performance" dar, wobei Kosten zweitrangig sind. Hierbei finden Wartung, Kontrolle und gegebenenfalls Optimierung kontinuierlich statt [DGI+03]. Probleme werden möglichst frühzeitig erkannt bzw. es wird proaktiv nach Wegen zur Performance-Steigerung gesucht. Auf welche Variante die Wahl fällt bzw. ob oder wie weit man sich diesen „Grenzfällen" nähert, hängt von der Unternehmenskultur und nicht zuletzt von der Priorität des Systems ab.

Der Schlüssel zum Performance-Tuning liegt hauptsächlich darin, zu wissen, wo man nach Performance-Problemen bzw. deren Ursachen suchen muss. Aus Sicht der Datenbank können Performance-Probleme aus einer Kombination inadäquaten Anwendungs- und Datenbankdesigns sowie suboptimaler Ressourcenausnutzung und -konfiguration hervorgehen. Nach [Nie99] können typischerweise 60% der Datenbank-Performance-Probleme dem Anwendungs- bzw. Programm-Design, 20% dem logischen und physischen Datenbank-Design, 18% der Konfiguration und 2% dem Betriebssystem und der Hardware zugeschrieben werden. Die Einschätzung, dass der Einfluss der Administratoren mit nur zirka 20% der Gesamt-Performance eines Datenbanksystems durchaus eingeschränkt ist, wird u. a. auch in [DGI+03] vertreten.

> *„ ... Performance is only refined by System Administrator and Database Administrator. It is created by the designers and programmers."* [DGI+03]

Dies soll die Notwendigkeit zur Optimierung jedoch keineswegs in Frage stellen. Datenbanken und Anwendungen können noch so gut konzipiert sein; wenn die Implementierung unzureichend umgesetzt bzw. das System schlecht konfiguriert ist, wird die Performance weit hinter den Erwartungen bleiben und den Anwendern nicht genügen.

In einzelnen Schätzungen wird gar behauptet, dass bis zu 40% der auftretenden Fehler oder Performance-Probleme mangelhafter Konfiguration zuzuordnen sind [GaCo03]. Entsprechend groß scheint das Optimierungspotential in diesem Bereich.

Durch gezieltes Tuning und die Optimierung der Nutzung von vorhandenen Ressourcen können Leistungsniveau und Stabilität geschäftskritischer Anwendungen gesteigert werden, ohne zusätzlich in Hardware (Prozessoren, RAM, Plattenplatz) zu investieren. Ein getunetes System befindet sich im „Einklang" mit den verfügbaren Ressourcen. Dabei ist zu unterscheiden zwischen der eher technisch ausgelegten *Optimierung von Ressourcen* und der *Optimierung von Applikationen.*

Im Einzelnen können im Rahmen des *Ressourcen-Tuning* nach der Identifizierung von Ressourcen-Engpässen zum System gehörige Datenstrukturen bzw. Ressourcen hinzugefügt (z.B. Indexe, Tablespaces, Puffer), entfernt oder in ihrer Konfiguration (durch Parameter) angepasst werden um so das Zusammenspiel bzw. die Verteilung der durch die Benutzer erzeugten Last zu verbessern und Performance-Engpässe zu vermeiden bzw. aufzulösen. Das Tuning kann dabei bestimmten Nebenbedingungen unterliegen, die entweder explizit definiert werden (z.B. Beschränkung hinsichtlich der Verwendung oder Typus von DBMS-Ressourcen) oder durch die Umgebung bedingt sind (Beschränkungen der Hardware- und Betriebssystemressourcen, wie z.B. Speicherobergrenzen). Ein Engpass bei CPU, Hauptspeicher, I/O oder Netzwerk lässt sich zwar mittels Monitoring identifizieren, für eine detailliertere Analyse werden jedoch Werkzeuge der Hardware- und Netzwerkanbieter benötigt. Angesichts der unüberschaubaren Anzahl der angebotenen Produkte soll dieser Bereich in der vorliegenden Arbeit ausgeklammert werden.

Während durch das Ressourcen-Tuning die durch die Applikationen erzeugte Last optimal auf das System verteilt wird, ist es Aufgabe des *Applikations-Tuning*, durch den effizienten Einsatz, die durchdachte Entwicklung von Applikationen bzw. Anpassungen ihres Verhaltens unnötige Last auf dem System erst gar nicht entstehen zu lassen bzw. zu korrigieren. In der Implementierungsphase werden Programme oft von unerfahrenen Entwicklern oder unter Zeitdruck geschrieben und mit einem völlig unrepräsentativen Datenbestand getestet. Später im produktiven Einsatz stellt sich dann heraus, dass diese Programme mit wachsendem Datenbestand kontinuierlich an Performance verlieren und schließlich zum Problem für das gesamte System werden. In der Praxis ist das Tuning-Spektrum jedoch durchaus eingeengt, da viele Sachverhalte vorgegeben werden und man häufig weder die SQL-Anweisungen noch die proprietären Anwendungen selbst anpassen kann oder darf. Spätere Eingriffe bzw. nachträgliche Performance-Optimierungen an diesen Stellen sind deswegen, wenn überhaupt, nur mit erheblichen Zeit- und Kostenaufwendungen sowie detailliertem Anwendungswissen und Kenntnis der jeweiligen Semantik bzw. Interna zu realisieren. Nichtsdestoweniger lassen sich Performance-Probleme durch ein systematisiertes Überwachen und Optimieren der Ressourcen auf bestimmte Nutzer oder Applikationen eingrenzen und somit an den entsprechenden Entwickler bzw. Berater adressieren.

Leistungsoptimierung durch Erweiterung der Hardware kann nötig sein, wenn ein ansonsten optimiertes System an seine Leistungsgrenzen stößt. Keinesfalls darf dieses Mittel jedoch zum „goldenen Hammer" und als vermeintliches Patentrezept für alle auftretenden Probleme erkoren werden. Es besteht die Gefahr, die eigentlichen Problem-Ursachen nur ohne wirksame Lösung auszublenden, so dass diese später u.U. wieder in größeren Dimensionen zu Tage treten.

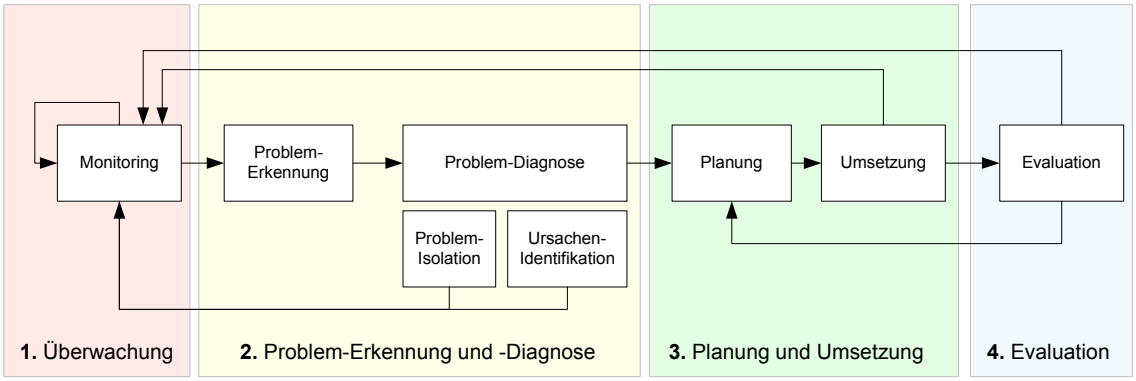


Abbildung 2.4: Die Phasen des Performance Tuning (in Anlehnung an [Wie05])

Aus welchen Gründen und zu welchem Zeitpunkt auch immer man sich für eine Performance-Optimierung (Tuning) entscheidet, der Ablauf entspricht grundsätzlich dem in **Abbildung 2.4** dargestellten iterativen Regelkreis, einem koordinierten Prozess aus 4(+1) untereinander abhängigen Phasen [IBM00, IBM03b, IBM04a, IBM04b].

0. *Festlegen und Quantifizieren der Performance-Ziele(ausserhalb von Abbildung 2.4)*

   Zusammen mit dem Management werden die durch das Tuning zu erreichenden objektivierbaren und sinnvollen Performance-Zielsetzungen sowie die relevanten Metriken vereinbart. Voraussetzung ist ein Verständnis der Geschäftsprozesse sowie deren Abarbeitung im System.

1. *Systematische Überwachung (Monitoring)*

   Um eine reaktive bzw. proaktive Optimierung zu erlauben, müssen die entsprechenden Komponenten bzw. deren Zustand und Verhalten systematisch überwacht bzw. gemessen werden. Die gesammelten Informationen sind im Tuning-Prozess mehrfach von Nutzen. Zum einen müssen natürlich (1) entsprechende Informationen vorliegen, um Aussagen über Konformität bezüglich der Performance-Ziele zu treffen. Durch die Bildung von auf gesammelten Daten basierenden Trends können potentielle Probleme und Ziel-Abweichungen auch schon frühzeitig, vor dem eigentlichen Auftreten, erkannt werden. Sollten nicht alle Ziele erreicht werden, müssen zum anderen während der (2) Analyse die Ursache des Problems sowie (3) Lösungsmöglichkeiten ermittelt werden. Dazu werden ebenfalls die Monitoring-Informationen verwendet und falls nötig weitere Monitoring-Läufe (z.B. das nachfolgend geschilderte Exception-Monitoring) angestoßen. Typischerweise unterscheidet man drei Strategien und damit auch Gründe für die Überwachung (und das Tuning). In welchem Umfang welche Variante verwendet wird, liegt in der Hand des Administrators.

   - *Routine-Überwachung* (Routine Monitoring)
     Darunter verstehen wir die regelmäßige Routine-Daten-Sammlung und -Überwachung des Systems während normaler und Spitzenzeiten. Basierend auf den gesammelten Daten kann ein Profil des (normalen) Systemverhaltens erstellt und zur Kapazitätsplanung, Vorhersage sowie frühzeitigen Problem-Erkennung verwendet werden. Aufgrund der regelmäßigen Natur dieser Überwachung und dem Ziel, möglichst wenig Overhead auf dem System zu erzeugen, werden typischerweise nur die sich mit wachsender Erfahrung als besonders wichtig herausstellenden Performance-Daten (Key Performance Indicators) in sehr groben Zeitabständen erfasst. Eine Stärke

dieser Variante ist die Möglichkeit zur Langzeitanalyse, wodurch sich u.U. Trends erkennen lassen, welche strategische Entscheidungen unterstützen. Paradebeispiel ist die Feststellung einer stetig größer werdenden Workload, auf welche dann rechtzeitig durch Rekonfigurationen oder Hardware-Erweiterung reagiert werden kann.

- *Vorausschauende Überwachung mittels Online Event Monitoring*
  Obwohl das System durch die Routine-Überwachung die meiste Zeit unter Kontrolle steht, können dennoch unerwünschte, jedoch erwartete Fehler/ Ereignisse gelegentlich eintreten. Beispiele hierfür wären Verzögerungen durch Paging/Swapping, „Aussetzer" durch ein überlaufendes File System, eine kurzzeitig nicht verfügbare Datenbank oder auch kurzfristige Trends durch eine hohe Anzahl an Sperrproblemen oder Transaktionen zu (typischen) Workload-Spitzenzeiten. Deadlocks beispielsweise werden vom DBMS selbstständig gelöst, verursachen aber vor allem beim häufigen Auftreten in kurzer Zeit einen Leistungsabfall. Wird beim Event Monitoring eine außergewöhnliche Häufung von Deadlocks festgestellt, kann sofort nach der Ursache gesucht werden. Ziel ist es daher, durch gezielte Datensammlung nach auf Fehler hinweisenden Ereignissen Ausschau zu halten und prompte korrigierende Maßnahmen einzuleiten.
- *Reaktive Überwachung* (Alert Monitoring)
  Der Wunsch hierbei ist die Identifikation unerwarteter und unerwünschter, entweder durch die Administratoren bzw. Endnutzer oder automatisch erkannter Ereignisse/Fehler sowie Ursachen und mögliche Lösungen zur Laufzeit. Dabei werden durchaus mehrere Monitoring-Läufe initiiert, um die Ursache des Problems systematisch mit aktuellen und aus der Vergangenheit zur Verfügung stehenden Daten einzugrenzen. Im Gegensatz zu den anderen beiden geplanten Überwachungsarten mit geringem Overhead, ist es im Falle von akuten Problemen wichtig, schnell zu handeln und zur Bestimmung der Probleme sowie der erforderlichen Maßnahmen die notwendigen Daten detailliert zu sammeln, auch wenn dabei u.U. erhebliche Zusatzlast auf dem System erzeugt wird. Das Alert Monitoring stellt den für das beschriebene iterative Tuning typischen Fall dar.

Zum Erfassen, Speichern, Aggregieren und Bewerten von Performance-Metriken bieten sich Dutzende auf den verschiedenen Ebenen und Komponenten wirkende „Expertenmonitore" als Data Supplier an. Klassische Monitoring-Mechanismen und -Tools überwachen bestenfalls jede Komponente einzeln. Angesichts der Vielzahl von Komponenten, die an einer Lösung beteiligt sein können und miteinander in Beziehung stehen oder kommunizieren, kann es sein, dass jede Komponente für sich korrekt arbeitet, das Gesamtergebnis dem Endanwender beispielsweise aufgrund fehlerhafter Kommunikation dennoch nicht performant oder korrekt zur Verfügung steht. Daher darf sich nicht darauf beschränkt werden, einzelne Hardware- und Software-Komponenten zu betrachten. Vielmehr sollte der Daten- und Informationsfluss über die Komponenten hinweg überwacht und optimiert werden. Idealerweise sollten Überwachungsdaten aller Komponenten, wenn möglich, an zentraler Stelle erfasst und konsolidiert werden. Siehe hierzu auch ausführliche Erläuterungen in den Kapiteln 5 und 6.

2. *Identifizierung und Analyse von Problemen (Problem-Erkennung und -Diagnose)*

   In dieser Phase werden Probleme *identifiziert*, klassifiziert und genauer diagnostiziert. Die Herausforderung bei der Problem-Erkennung ist es, schnellstmöglich

bzw. idealerweise vorausschauend Symptome und Situationen durch die Anforderungen bestimmbaren, abnormalen Verhaltens zu identifizieren und unwichtige Informationen aus der Flut der Daten zu filtern. Ist ein außergewöhnliches Ereignis oder Verhalten erkannt, erfolgt zunächst die Zuweisung einer Dringlichkeitsstufe [AGK+04]. Sporadisch auftretende, unkritische Ereignisse sind nicht dringlich. Sie können weiterhin mittels Routine Monitoring beobachtet werden, um so auch Schlüsse für zukünftige Korrekturen ziehen zu können. Regelmäßig auftretenden, die Geschäftsprozesse störenden Ereignissen muss zügig Aufmerksamkeit geschenkt werden. Schwerwiegende Ereignisse, wie Ausfälle eines kompletten Datenbankservers, hingegen müssen sofort bearbeitet werden.

Die *Problemisolation* beschäftigt sich darauf folgend mit der Aufgabe, das erkannte Problem genauer zu lokalisieren und zu charakterisieren. Dies umfasst u.a. das Feststellen, wann und wo welche Probleme aufgetreten und welche Ressourcen davon betroffen sind. Ein Problem setzt sich meist über viele Schichten im System fort und wird daher bei verschiedenen Komponenten auf unterschiedliche Art erkannt. Bei der *Ursachenidentifikation* werden die dem Problem zugrundeliegenden Faktoren (das „Warum“) und die eigentliche(n) Ursache(n) genauestens bestimmt. Falls ein dringliches Problem aufgetreten ist, muss die Fehlerursache systematisch eingegrenzt werden. Dazu werden Hypothesen über die mögliche Ursache aufgestellt. Alternativ kann man Hypothesen auch entsprechend einer Checkliste von typischen Ursachen für Performance-Probleme aufstellen und abarbeiten. Zum Bestätigen bzw. Widerlegen der Hypothesen werden Informationen des Routine- und Event-Monitoring verwendet. Ungenügende Informationen in dieser Phase führen dazu, dass eine weitergehende Überwachung nötig ist und mittels Alert Monitoring noch detailliertere bzw. relevantere Informationen gewonnen werden können.

3. *Planung und Umsetzung von Optimierungsmaßnahmen*

   In dieser Phase werden die zur Lösung bzw. Behebung des Problems und zur Rückkehr zum normalen Betriebszustand aussichtsreichsten Tuning-Maßnahmen bestimmt und ausgeführt. In der Regel kommen zur Behebung eines Problems mehrere Lösungen in Betracht, die in einer bestimmten Reihenfolge und im Hinblick auf lokale Gegebenheiten ausgeführt und aufeinander abgestimmt werden sollten. Das Vorgehen der Administratoren beruht dabei maßgeblich auf Erfahrung und bewährten Verfahren. Primär sollte die eigentliche Ursache (problem resolution) und nicht nur die Symptome des Problems (problem recovery) adressiert werden, so dass sich Fehlsituationen nicht notwendigerweise wiederholen. In der Praxis wählt man aus Gründen des akuten Handlungsbedarfs jedoch häufig einen Symptom-behebenden bzw. -lindernden Workaround und beschäftigt sich anschließend in Ruhe mit der zugrundeliegenden Ursache.

4. *Evaluierung*

   In der Evaluierungsphase wird anschließend (durch Überwachungsdaten) verifiziert, ob die durch den Administrator eingeleitenden Maßnahmen den gewünschten Erfolg erzielt haben und ob ggf. eine erneute bzw. angepasste Planung notwendig ist. Sollte keine Verbesserung eintreten, können die vorgenommenen Änderungen rückgängig gemacht oder neue Maßnahmen ermittelt werden. Sowohl jede erfolgreiche als auch jede erfolglose Änderung sollte dokumentiert werden, um Anhaltspunkte für zukünftige Optimierungsprozesse zu haben.

Dieses naheliegende Vorgehen ist im Übrigen in ähnlicher Art und Weise u.a. auch im 4-Phasen-Fehlerprotokoll [Ech90] zur Fehler-Erkennung (detection), -Eindämmung (containment), -Klassifizierung (taxonomy) und -Behandlung (recovery) sowie in den 4 MAPE-Phasen des Autonomic Computing (siehe Abschnitt 2.2) wiederzufinden.

Die Prozesse bilden einen iterativen Zyklus, der einerseits durch interne Anforderungen bzw. Anpassungen getrieben, andererseits aber auch von externen DBA-Wünschen und Vorgaben (Richtlinien, Policies) und insbesondere der dynamischen Workload beeinflusst wird. In dem zyklischen Verlauf wird immer wieder geprüft, ob die gemessene Leistung den Vorgaben der Performance-Ziele gerecht wird. Ist dies nicht der Fall, werden entsprechende Anpassungen eingeleitet. Pro Iteration sollte jedoch nur eine Änderung vorgenommen werden, um die Kausalität zwischen angewandter Optimierungsmaßnahme und Performance- bzw. Zustands-Änderung zweifelsfrei belegen zu können. Es wäre andernfalls schwer möglich zu sagen, welche aus einer Vielzahl von Veränderungen nun für die Leistungssteigerung verantwortlich war. Schlimmer noch, Modifikationen könnten sich in ihrer Wirkung aufheben und Leistungspotentiale verschenkt werden. Eine Ausnahme von der Regel sei jedoch an dieser Stelle erwähnt. Bei der sogenannten „All-at-once-Methode“ werden mehrere Veränderungen auf einmal vorgenommen. Sinnvoll bzw. nötig ist dies beispielsweise im Rahmen einer Neuinstallation, bei welcher die Parameter auf Standardwerte gesetzt werden. Das System soll so einen stabilen Ausgangszustand erreichen und kann mit dem iterativen Ansatz weiter optimiert werden. Im Allgemeinen ist von dieser Methode aber abzuraten, da die Gesamtheit der Auswirkungen und Interdependenzen nicht zu überblicken sind.

2.1.3 Datenbank-spezifische Diagnose- und Tuning-Techniken

Gegenwärtig erscheint die Identifizierung und Lösung von Performance-Problemen oftmals keiner erprobten Methodik oder Disziplin zu gehorchen [Sne03]. Heutzutage werden die Performance-Problem-Diagnose (Identifizierung von Performance-Problemen) und das Performance-Tuning (Lösung von Performance-Problemen) meist manuell durch die DBA durchgeführt. Unterstützung erhalten sie dabei von diversen Tools, vom DBMS selbst oder von eigenen Skripten, die Möglichkeiten zur teilautomatisierten Systemüberwachung liefern. Bei der Analyse der Überwachungsdaten endet die Unterstützung und „Handarbeit“ ist notwendig. Jeder DBA entwickelt hierin seine eigene, auf bisherigen Erfahrungen und Intuition basierende Vorgehensweise. Während erfahrene DBA den Ruf eines die „schwarze Magie“ beherrschenden „Guru“ genießen, wird es aufgrund der steigenden Komplexität der DBMS für „neue“ DBA, die noch nicht viel Erfahrung sammeln konnten, umso schwerer, den Einstieg zu meistern.

Die „Administratoren“ im Gesundheitswesen, die Ärzte, folgen hingegen festgelegten Regeln und teilen gemeinsame Elemente der Philosophie, Bildung, Erfahrung, bewährte, althergebrachte Verfahren der Diagnose und Therapie von Beschwerden der Patienten und etablierte Methoden der Problemlösung [Sne03]. Die medizinischen Abläufe sind festgelegt und werden größtenteils von Institutionen zwecks Qualitätssicherung überwacht. Die Aufgabenverteilung zwischen Ärzten, Assistenten, Patienten, Schwestern etc. ist klar vorgegeben. Infolgedessen stellen wir uns in [Hen07] die Frage, ob sich bestimmte Aspekte der medizinischen Diagnose und Therapie auch auf die Performance-Problem-Diagnose übertragen lassen. Die medizinische Diagnose ist das Ergebnis eines kritischen Denkprozesses mit dem Ziel, aus der Menge von Symptomen eines Patienten, seiner Krankheit einen „Namen“ zu geben [Hel93]. Symptome (Probleme) sind dabei beobachtbare bzw. messbare Anzeichen, die auf eine Krankheit oder Verletzung (Ursache, root cause) hinweisen. Krankheiten teilen dementsprechend mit Performance-Problemen die Gemeinsam-

keit, dass sie mit Beschwerden verbunden sind und anhand bestimmter Symptome erkannt werden können. Die Therapie bzw. das Performance-Tuning stellt eine Heilungsmethode dar, die zur Linderung oder Beseitigung der Symptome führt. Eine exakte Diagnose führt zur Ursache der Krankheit bzw. des Performance-Problems, deren Beseitigung primär im Vordergrund steht. In [Hen07] widmen wir uns ausführlicher dieser Thematik und zeigen darin sowohl Parallelen der medizinischen zur informationstechnischen Diagnose sowie verschiedene Verfahrensweisen, die auf unser Szenario übertragen, entweder einzeln oder aber kombiniert zum Einsatz kommen können.

Wie bereits dargelegt, gibt es in einem System viele Aspekte, die optimiert werden können und auf die die Diagnose und das Tuning gesondert ausgerichtet sind. Im Folgenden sollen **drei fundamentale Diagnose-Methoden** im Umfeld des Datenbank-Tuning vorgestellt werden. Ausgangspunkt sei hierfür eine durch die Endnutzer verzeichnete, durch Diagnose genauer zu spezifierende und mittels Tuning zu behebende Leistungseinbuße, wie beispielsweise verlängerte Antwortzeiten. Detaillierte Informationen zu dieser Thematik finden sich in [Hen07].

Bei der klassischen **(1) Kennzahlen-Analyse** stellen über die Zeit ermittelte Performance-Metriken Indikatoren für Performance-Probleme dar [Sch03]. Anhand von Erfahrungen oder auch bewährten Vorgehensweisen sowie unter Einbeziehung zusätzlicher Kontext-Informationen über die Umgebung und die Workload können DBA schließlich einschätzen, ob eine Ressource anhand ihrer Kennzahlen eine gute oder schlechte Performance aufweist. Dabei ist es möglich, dass eine einzelne Ressource eine schlechte Performance aufweisen kann, aber keine schlechte Gesamt-Performance implizieren muss. Darüber hinaus sagen Kennzahlen nichts über die eigentlichen Ursachen oder die Zufriedenheit der Endnutzer, im Sinne etwa von Antwortzeiten, aus. Diese Form der Performance-Problem-Diagnose wird jedoch seit Jahren von den DBA genutzt, wohl weil sie mitunter eine übergreifende und gesamtheitliche Sichtweise auf die DBMS-Ressourcen erlaubt.

Bezeichnet wird dieses gebräuchliche Vorgehen von Cary Millsap in [MiHo03] als „Method C“, wobei sich das „C“ aus dem englischen Conventional ableitet. Er umreißt die Methode dabei anhand von vier Schritten:

1. Das Aufstellen der Hypothese, dass eine Performance-Metrik M einen inakzeptablen Wert hat.
2. M iterativ verbessern unter Rückgängigmachen jedes Versuchs, der die Performance bemerkbar verschlechtert.
3. Falls der DBA oder der Endnutzer keine zufriedenstellende Verbesserung (z.B. anhand der Antwortzeit) wahrnimmt, gehe zurück zu Schritt 1.
4. Falls die Performance-Verbesserung zufriedenstellend ist, gehe trotzdem zu Schritt 1 zurück, da es möglich sein könnte, weitere Performance-Verbesserungen zu erreichen.

Diese Methode wird auch als Trial-and-Error-Methode bezeichnet, da die Wahl der zu optimierenden Metrik bzw. der Komponente eher zufällig gefällt wird und keinen nennenswerten Gesetzmäßigkeiten folgt.

Sofern Anwendungen oder Prozesse gerade keine Tätigkeit verrichten, warten sie auf ihren Einsatz. Bei der **(2) Bottleneck-Analyse** steht der Zeitverbrauch einzelner Aktionen in

bzw. durch die Komponenten im Vordergrund [Sch03]. Ein häufig wartender bzw. lang wartender Prozess kann ein Indikator für ein Performance-Problem und damit ein prinzipieller Performance-Engpass (Bottleneck) sein. Bei der Suche solcher Ressourcen-Engpässe bietet sich i.d.R. ein Top-Down-basiertes Vorgehen, von der problematischen Anwendung bis zu einzelnen verantwortlichen Ressourcen, die betroffene Daten bearbeiten und letztlich ungewöhnlich lange Wartezeiten hervorrufen, an.

Dieser von in [MiHo03] als „Method R" („R" für Response Time, Antwortzeit) getaufte und weiterentwickelte, analytische Ansatz zur Verbesserung der Performance orientiert sich an einer Optimierung der Antwortzeit. Die vier Schritte der Methode sind:

1. Die Identifikation und Auswahl der Benutzeraktionen, für die eine Verbesserung erzielt werden soll. Bei der Auswahl spielen vielmehr unternehmensrelevante (High-Level-)Aspekte eine Rolle, als für den Endnutzer unverständliche Low-Level-Kenngrößen.
2. Das Sammeln geeigneter Statistiken für jede der Aktionen zu den Zeiten, in denen sie nicht optimal arbeiten und einen hohen Anteil der Antwortzeit einnehmen.
3. Die (Re-)Priorisierung und Ausführung der Optimierungsmöglichkeiten nach Kosten-Nutzen-Verhältnis, so lange, bis die gewünschte Performance-Verbesserung erreicht ist.
4. Gehe zu Schritt 1.

Zu beachten ist dabei, dass es sich keinesfalls um eine Endlos-Schleife handelt, wie Schritt vier suggerieren mag. Gibt es in Schritt eins keine relevanten Gründe mehr für weitere Performance-Verbesserungen, so ist die Optimierung abgeschlossen. Letzteres ist auch erforderlich, sofern weitere Verbesserungsversuche vom Kosten-Nutzen-Verhältnis nicht vertretbar sind.

Der Kernpunkt und in der Regel der zeitaufwendigste Aspekt dieser Methode ist es, die richtigen statistischen Daten zu sammeln und geeignet zu analysieren. Die darauf folgende Umsetzung der Maßnahmen ist anschließend als weit weniger problematisch anzusehen.

Das Messen, Erfassen sowie das graphische Visualisieren von DB-Wartezeiten und das Herunterbrechen auf einzelne Teilschritte einer Query verdeutlicht direkt, wie gut die Applikationen agieren und erlaubt die gezielte Beantwortung mitunter folgender Fragen:

- Welche individuellen SQL-Statements bzw. kritischen Ressourcen sind verantwortlich für Verzögerungen innerhalb der betrachteten (aktuellen oder historischen) Zeitspanne?
- Wie lange warten bestimmte Transaktionen, Nutzer, Applikationen und Systeme auf Datenbank-Ressourcen zu bestimmten Zeiten?
- Welche Datenbank-Ressourcen sind die größten Bottlenecks und wieviel Zeit gewinnt man vermutlich nach einem Fix dieser Problemstellen?
- Kann das Bottleneck durch reines Datenbank-Tuning adressiert werden, oder muss es an andere Verantwortlichkeiten, wie z.B. einen Entwickler oder auch Netzwerk-Administrator, delegiert werden?
- Welche Applikationen und Ressourcen haben über die letzten Wochen bzw. Monate Performance-Änderungen aufzuweisen? (Trending)

Im Unterschied zur Bottleneck-Analyse wird die zu untersuchende und optimierende Größe nicht mehr oder weniger zufällig ausgewählt, sondern auf Grundlage einer Analyse der gesammelten Daten. Die Vorgehensweise ermöglicht, die Probleme eines Systems gezielt identifizieren und damit auch angemessen (auf-)lösen zu können. Der Erfolg wird

an der für die Endnutzer sichtbaren Kenngröße Antwortzeit bzw. Datenbankzeit gemessen. Eine Verbesserung bringt daher in jedem Fall eine merkliche Leistungssteigerung mit sich und beschränkt sich nicht nur auf eine interne Optimierung einer beliebigen Kennzahl.

Darüber hinaus kann eine umfangreiche **(3) Workload-Analyse** bei der Überprüfung anwendungsspezifischer Vorgänge mit dem Ziel der Reduzierung des Ressourcenverbrauchs angewendet werden. Hierbei erstellt man zur Eingrenzung einer Liste aller problematischen, das System durch eine hohe Last stark beanspruchenden Applikationen, Transaktionen oder SQL-Anweisungen. Kriterien zur Auswahl sind bspw. Häufigkeit der Ausführung, Dauer der Ausführung, CPU-Zeit, I/O-Zeit etc. Das darauf folgende Ermitteln und Optimieren der durch die identifizierte Workload einer hohen Auslastung unterliegenden Ressourcen hat wiederum hohe Performance-Auswirkungen auf die Anwendungen. Um die Analyse effizienter zu gestalten und den Problemraum zu verringern, empfiehlt es sich, die Anweisungen hinsichtlich ihrer Ähnlichkeit zu gruppieren (Statement Equalization) und nicht auf Basis einer einzelnen Anweisung, sondern für eine Gruppe ähnlicher Anweisungen die Analyse durchführen [Hay05].

Für eine aussagekräftige, realistische Performance-Problem-Diagnose und die sich anschließende unverzügliche -Auflösung sollten die Diagnosetechniken miteinander kombiniert werden [Sch03, Hen07]. Die Workload-Analyse kann anfänglich einen Überblick der Problemauswirkungen auf oberster Ebene vermitteln. Darauf basierend lassen sich die beteiligten Ressourcen bestimmen. Mit Hilfe der Kennzahlen-Analyse kann der Administrator in Folge einen Überblick über wichtige Performance-Metriken gewinnen und eine genauere Problem-Eingrenzung vornehmen. Die detaillierte Bottleneck-Analyse sollte letztlich aufzeigen, wo die eigentlichen Engpässe im System bestehen.

Im Verlaufe der vorliegenden Arbeit werden noch weitere Beispiele folgen, die erkennen lassen, dass es im Rahmen des Autonomic Database Performance Tuning sinnvoll erscheint, die Diagnosemethoden miteinander zu kombinieren. Effiziente Diagnosen sind machbar, wenn zum richtigen Zeitpunkt die richtige Methode gewählt wird und die dafür richtigen Daten vorhanden sind. Je nach Situation wird mehr oder weniger Aufwand in die Diagnose investiert, damit Performance-Tuning-Maßnahmen entsprechend schnell eingeleitet werden können.

2.1.4 Datenbank-Performance-Tuning-Szenarien

Am Beispiel von „IBM DB2 for Linux, Unix and Windows“ (DB2 LUW[1]) sollen im Überblick ausgewählte Tuning-Szenarien und Vorgehensweisen zur Verbesserung der Leistungsfähigkeit für den Zugriff und die Verarbeitung durch das DBMS aufgezeigt werden.

Typisch beim Einsatz von IBM DB2 sind die folgenden, nach ansteigender Häufigkeit aufgelisteten, die Statistiken aus Abschnitt 2.1.2. widerspiegelnden **Probleme**:

- Externe Ressourcenengpässe (CPU, I/O, Speicher, Netzwerk-Bandbreite)
- Interne Ressourcenengpässe
    - Sperr- und Deadlock-Konflikte
    - Unzureichender „DB Shared Memory“ (Bufferpools, Sort Heaps, Application Global Memory, Catalog/Package Caches etc.)

[1] http://www-01.ibm.com/software/data/db2/linux-unix-windows/

- Suboptimale bzw. nicht-vorhandene (adäquate) Indexe
- Mangelhafte Maintenance (runstats, reorg)
- Unausgereifte Applikationen, zugehöriges DB-Design oder auch ineffizienter SQL-Code

Die Überwachungs-Werkzeuge in IBM DB2 beispielsweise lassen sich in zwei Kategorien einteilen [Ree04, Ree05]: Werkzeuge, die lediglich Informationen liefern und Werkzeuge, die zusätzlich Analysemöglichkeiten zur Verfügung stellen. Zur ersten Kategorie kann man den Database System Monitor [IBM06i] mit Snapshot-, Event- und Health Monitor sowie Explain Snapshots, Protokolldateien und das Tracing zählen. Zur zweiten Kategorie zählen Activity Monitor, Memory Visualizer, Health Center, Event Analyzer, Explain Tools und das Problem Determination Tool. Daneben existieren noch eine Menge proprietäre Tools von Drittherstellern, die das Monitoring und damit die Diagnose unterstützen. Für nähere Details sei auf Abschnitt 5.3. und auf die Arbeiten [Ehl07, Hen07] verwiesen.

Zur Auflösung eines Problems gibt es in der Regel vielfältige Strategien, die je nach Ursache in unterschiedlicher Reihenfolge anzuwenden sind. Bei diagnostizierten DB2-Problemen können u.a. die nachfolgend exemplarisch aufgelisteten, wesentlichen **Tuning-Möglichkeiten** auf Ebene der Ressourcen und der Applikationen durch die Administratoren in Betracht gezogen werden:

- System-technische Aktivitäten
  - Einstellen der Konfigurationsparameter (z.B. Anzahl paralleler Threads, Log-/Lock-Management, Optimizer-Suchtiefe)
  - Optimieren des DB-Layouts: Physische DB2-Objekte und deren Optionen (database, tablespace, index, bufferpool)
  - Re-Organisation der physischen Datenspeicherung
  - Permanente Überwachung des Systemverhaltens und Starten von Utilities (z.B. runstats)
- Anwendungs-bezogene Maßnahmen
  - Logische und physische Datenmodellierung (Festlegung der Objekte)
  - Einsatzentscheidungen für Tabellen, Sichten etc.
  - Anlegen/Ändern/Löschen von Indexen
  - Festlegung und Test von SQL-Statements
  - Umschreiben von Queries in effizientere Form
  - Einbau von Commits zur Vermeidung von lang laufenden Transaktionen
  - Ändern des Isolation Level zum gezielteren (Ent-)Sperren
  - (Re-)Definition von Constraints, Triggern, UDFs und Prozeduren

Im Rahmen des vorliegenden Kooperationsprojekts mit der IBM wurden eine Fülle an weiteren bekannten DB2-spezifischen Problemen sowie systemtechnischen und Anwendungs-bezogenen, bewährten Tuning-Maßnahmen und Check-Listen für einen auf DB2 basierenden Software-Stack identifiziert und ausführlich in [Ehl07, Hen07, Roe08, WiRa07, WRRA08, WRRA09] beschrieben. Es sollen dadurch primär Ideen geliefert werden, welche Vorgänge für ein autonomes DB-Tuning formalisiert und automatisch abarbeitbar gemacht werden sollten (siehe Kapitel 4). Ziel der Formalisierung solcher Expertenanalysen und -vorgehen ist es, auch Benutzern (oder eben auch automatischen Routinen) von DB2, die nicht als Experten auf diesem Gebiet bezeichnet werden können, zu erlauben, ein akzeptables Performance-Tuning an ihren Datenbanken durchzuführen.

2.1.5 Fazit

Zur Gewährleistung adäquater Performance sind, wie in der Motivation und in diesem Abschnitt 2.1. erläutert, System- und Datenbank-Administratoren gemeinsam verantwortlich. Schon bei einfacheren Systemen verbringen die Administratoren einen nicht unerheblichen Teil ihrer Arbeit mit wiederkehrenden, zeitintensiven Aufgaben, wie Komponenteninstallation, Konfiguration, Tuning etc.

Zu den konkreten Aufgaben von DBA gehören neben obligatorischen Tätigkeiten, wie dem Anlegen und Verwalten von DB-Objekten, dem Exportieren und Laden/Importieren von Daten, Backup und Recovery auch das Erstellen, Implementieren, Überwachen von Autorisierungsmaßnahmen sowie Pflege- und Wartungsarbeiten (Pflege der Statistiken, Reorganisation der Daten, Rebind von Programmpaketen etc.). Insbesondere im Rahmen des Performance-Tuning sind auch die Formulierung von Performance-Zielen, die Überwachung und Verwaltung der benötigten Systemressourcen, die Analyse von gewonnenen Daten sowie das Erkennen und Lösen von Performance-Problemen von hoher Bedeutung.

Diese umfangreichen Aufgaben erfordern bei der derzeitigen Komplexität, der Vielzahl an Stellschrauben und möglichen Problembereichen eines (Datenbank-)Systems trotz weniger unterstützender Tools und selbst geschriebener Skripte einen immensen Zeitaufwand und vornehmlich sehr viel Erfahrung und Routine. Viele langjährige DBA sind zu Experten auf dem Gebiet der Performance-Verbesserung von Datenbanken geworden und haben über die Zeit und mit wachsender Erfahrung der Leistungsverbesserung von DB dienliche Expertenstrategien entwickelt.

Inzwischen sind die Anforderungen an die Verwaltung von IT-Systemen allerdings gewaltig geworden und selbst hoch-qualifizierte Administratoren sehen sich mit zunehmend unüberschaubaren Aufgaben konfrontiert. Ein bestmöglicher Ablauf und damit ein universelles Vorgehen sind bei derartigen Optimierungsaufgaben aufgrund der Vielzahl an Abhängigkeiten zwischen den einzelnen Aufgaben und der Dynamik des zugrundeliegenden Systems bei weitem nicht (einfach) zu ermitteln. Es ist dem Menschen praktisch nicht ohne weiteres möglich, alle Trade-Offs zu (er-)kennen und zu berücksichtigen.

Selbst wenn aus ökonomischer Sicht mit hohen Kosten verbundenes zusätzliches Personal in ausreichend großer Zahl zur Verfügung stünde, wäre der Arbeitsaufwand enorm und es würde den Systemadministratoren noch immer Spezialwissen abverlangt werden, das sie teilweise nicht besitzen bzw. in das sie eingearbeitet werden müssen. Von einer anhaltenden Zunahme der Komplexität in IT-Systemen im Allgemeinen und in DBMS im Speziellen kann ausgegangen werden. Hinzu kommt die Tatsache, dass die Systemverwaltung durch den Menschen fehleranfällig und aufgrund von Urlaubs- und Pausenzeiten teilweise im laufenden Betrieb nicht möglich ist. Die Folge sind verschenkte Optimierungspotentiale oder im schlimmsten Fall Performance-Einbußen. In der Natur des Menschen liegt es auch, eher reaktiv und oftmals stark zeitverzögert zu handeln. Gründe dafür liegen in meist hoher Arbeitsbelastung sowie dem langsamen und u.U. fehlerbehafteten Prozess von der Problem-Erkennung bis hin zur Problem-Auflösung.

Zusammenfassend ist festzustellen, dass Administratoren mit wachsender Systemkomplexität zunehmend überfordert sind, einen großen Anteil der TCO ausmachen und somit eine Automatisierung der „Wertschöpfungskette“ administrativer Aufgaben sehr zu wünschen ist.

2.2 Autonomic Computing

Das durch die rasche Entwicklung vielschichtiger, heterogener Umgebungen schwerwiegende Problem der Komplexität wurde auch durch Paul Horn, IBM Senior Vice President und Director of Research, in [Hor01] erkannt und beschrieben.

> *„More than any other I/T problem, this one - if it remains unsolved - will actually prevent us from moving to the next era of computing. ... The obstacle is complexity. Dealing with it is the single most important challenge facing the I/T industry."*

Will man dennoch die Möglichkeiten neuer Technologien weiterhin nutzen, eine Überforderung der Administratoren vermeiden sowie die heterogenen und vielfältigen Systeme effizient verwalten, so sollte man den Menschen entlasten, indem man ihm unterstützende Werkzeuge bereitstellt. Idealerweise werden IT-Systeme angepasst bzw. um Komponenten erweitert, die eine automatisierte Systemüberwachung und -verwaltung unterstützen. Ansätze, wie die im Oktober 2001 von IBM hervorgerufene Initiative des **Autonomic Computing**[2]**,** sollen helfen, das Problem der steigenden Komplexität zu adressieren, indem sie Technik nutzen, um Technik zu verwalten [IBM06e].

Im Rahmen dieses Abschnitts soll kurz erklärt und verdeutlicht werden, worum es sich beim Autonomic Computing genau handelt, auf welchen Grundsätzen es basiert und was die Ziele der sich davon gewünschten Selbst-Verwaltung sind. Darüber hinaus wird ein konkreter Implementierungsansatz eines Autonomic-Computing-Systems nach den Vorstellungen der Firma IBM erläutert. Diese Ausführungen basieren auf [IBM06e, IBM07, KeCh03, Mur04, PH05].

2.2.1 Die Vision (und die Ziele)

Ziel des Autonomic Computing ist die Realisierung von IT-Systemen, die sich (weitestgehend) ohne Eingriffe des Menschen selbst verwalten können, um so ihre Komplexität sowie die Details des Systembetriebs und der Wartung vor dem Menschen zu verbergen. Der Begriff „autonomic" ist bewusst in Anlehnung an das vegetative Nervensystem (auch autonomes Nervensystem genannt) des Menschen gewählt. Dieses soll als Leitbild dienen, da es in der Lage ist, die menschlichen Körperfunktionen selbstständig zu steuern und sich ständig ändernden Umweltbedingungen anzupassen, ohne dass der Mensch aktiv eingreifen muss. Das vegetative Nervensystem regelt völlig autonom lebensnotwendige Routinefunktionen, indem es permanent ohne Wissen des Menschen diverse lebenswichtige Vital-Indikatoren, wie beispielsweise Herzschlagfrequenz, Atmung, Puls, Körpertemperatur, Blutzuckerspiegel und Blutdruck, überwacht [IBM06e]. Sobald eine gravierende Differenz zu den üblichen Werten festgestellt wird, kann der Körper mit entsprechenden Maßnahmen automatisch darauf reagieren. So werden bspw. der Puls und die Atmung umgehend den jeweiligen Belastungssituationen (bspw. Ruhe oder Sport) angepasst. Auf diese Weise kann möglichen ernsthaften Problemen vorgebeugt oder im konkreten Fall darauf reagiert werden.

Diese Methode soll analog auf IT-Systeme übertragen werden [Hor01]. Ebenso wie das vegetative Nervensystem sollen autonome Systeme Routineaufgaben selbst übernehmen, die eigenen Komponenten ständig überwachen, Probleme erkennen und nötige Änderungen mit möglichst wenig menschlicher Interaktion vornehmen können. Benutzer oder Administratoren werden entlastet und bei derartigen Systemen nur noch in wenigen Ausnahmefällen Veranlassung haben, manuell in den laufenden Betrieb eingreifen zu

[2] http://www.research.ibm.com/autonomic

müssen. Vielmehr verschiebt sich ihr Aufgabenspektrum weg von den „niederen", repetetiven Aufgaben hin zu den eigentlich wichtigen, nicht delegierbaren, langfristig-orientierten, strategischen Tätigkeiten, wie bspw. die Formulierung und Formalisierung von grundsätzlichen Strategien und Systemrichtlinien, anhand derer das System agiert.

„This allows you to think about what you want to do, and not how you'll do it" [Hor01]

Ein grundlegender Unterschied im Vergleich zum vegetativen Nervensystem besteht allerdings in der Tatsache, dass der Administrator frei entscheiden kann, welche Aufgaben und Funktionen er an das System delegiert und gemäß welcher Unternehmensziele, Richtlinien und Bedingungen diese ausgeführt werden sollen [IBM06e]. Paradoxerweise muss erst ein u.U. noch komplexeres System geschaffen werden, damit den Administratoren und den Nutzern der Umgang mit IT vereinfacht wird [Hor01].

An dieser Stelle soll darauf hingewiesen werden, dass der Begriff des Autonomic Computing u.a. von IBM verwendet wird. Autonome Computersysteme sind schon länger unter dem Namen „Selbstverwaltende Systeme" oder „Selbstorganisierende Systeme" bekannt, erhalten aber in den letzten zehn Jahren wieder gesonderte Aufmerksamkeit.

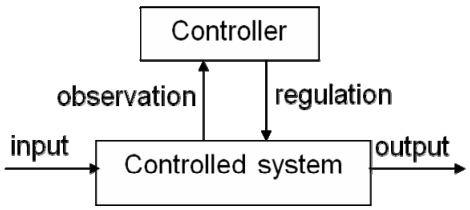


Abbildung 2.5: Controller auf einem überwachten System [DHK+05]

Insbesondere in der Regelungs-/Steuerungs- und Automatisierungstechnik dient ein auf dem zu überwachenden System aufgesetzter Controller (siehe **Abbildung 2.5**) zur Umsetzung eines sogenannten Regelkreises, der permanent kontrolliert, dass der gewünschte Effekt (Soll-Zustand) erreicht wird [DHK+05]. Zur Aufdeckung und Auflösung von Abweichungen vom Standardverhalten beobachtet und misst er dafür kontinuierlich den aktuellen Zustand des Systems und initiiert im Problemfall angemessene Aktionen, um das vorgegebene Ziel unter den Randbedingungen (wieder) zu erfüllen. Die Übertragung der Funktionsweise eines solchen Regelkreises wird für unseren Kontext in Abschnitt 2.2.4 erneut thematisiert.

Forschungen und Entwicklungen auf dem Gebiet des Autonomic Computing bzw. der Selbst-Verwaltung wurden und werden auch von anderen IT-Firmen durchgeführt, die dafür teils ihre eigenen Begriffe verwenden (z.B. Microsoft - Dynamic Systems Initiative, HP - Adaptive Enterprise). Eine detaillierte Beschreibung ausgewählter Forschungsprojekte findet man in [DHK+05, MOKW06].

2.2.2 Kernelemente autonomer Systeme

Das sog. „Autonomic Computing Manifesto" [IBM01] diente als erster grober Entwurf zur Identifikation und Präzision der Schlüsseleigenschaften, die ein als autonom betrachtetes Computersystem charakterisieren. Ein autonomes System

1. muss sich **selbst kennen**, Informationen über die vorhandenen Komponenten, deren Zustand, Kapazitäten und alle Abhängigkeiten zu anderen Systemen besitzen. Des Weiteren ist auch Wissen über zusätzlich verfügbare Ressourcen vonnöten, welche z.B. bei Engpässen eingebunden bzw. mit anderen Systemen geteilt werden können.

2. muss sich **selbst (re-)konfigurieren** und automatisch an sich ändernde, unvorhersehbare Umstände anpassen können. Diese Änderungen beruhen auf speziellen, im Vorfeld von IT-Fachkräften festgelegten Richtlinien und können sowohl die Integration neuer Komponenten, als auch die Anpassung (in den Systemeinstellungen) und Entfernung von bereits vorhandenen betreffen. Eine kontinuierliche Anpassung an die aktuellen Situationen hilft, die Stärke und die Produktivität der IT-Systeme sicherzustellen und die Flexibilität aufrechtzuerhalten. Paradebeispiel für Selbstkonfiguration ist die USB-Architektur, welche die dynamische Einbindung der unterschiedlichsten Komponenten ermöglicht.
3. muss in der Lage sein, sich **selbst heilen** zu können. Im Falle einer erkannten Problemsituation bzw. Fehlfunktion müssen alternative Wege gefunden bzw. eine geeignete Rekonfiguration zur Behebung durchgeführt werden, ohne dabei den normalen Betrieb zu unterbrechen bzw. merklich zu stören. Praktisch könnte dies bspw. eine ständige Überwachung mittels Monitoring bedeuten, wobei beim Auftreten von Fehlern auf redundante oder wenig ausgelastete Komponenten (etwa im Standby) ausgewichen wird. Fehlerhafte Komponenten können so isoliert und nach Möglichkeit repariert werden. Nur falls dies nicht möglich ist, z.B. bei einem Hardwaredefekt ohne weitere Redundanz, müssen die Administratoren verständigt werden. Systeme werden so belastbarer und die Gefahr von Systemausfällen sowie die Fehleranfälligkeit werden reduziert.
4. muss stets nach Methoden suchen, sich **selbst** zu **optimieren** und zu verbessern, ohne sich mit dem Status quo zufrieden zu geben. Der Optimierungsprozess orientiert sich dabei an den jeweiligen Service Level Agreements, um Optimierungsziele zu erreichen und den Bedürfnissen von Nutzern und Geschäftsprozessen gerecht zu werden. Beispielhaft wäre die Anpassung an eine sich verändernde Workload. Hierbei lässt sich bereits erahnen, dass zur Realisierung der Optimierung unter anderem auch auf die Fähigkeiten zur selbstständigen Rekonfiguration und Heilung zurückgegriffen wird.
5. muss sich **selbst** gegen schädliches Verhalten oder gar Angriffe von außen bzw. innen **schützen** können. Nach außen richtet sich der Schutz in erster Linie gegen die Bedrohung durch Viren und gezielte Angriffe durch Hacker. Nach innen bedeutet es die Nutzerverwaltung mittels Authentifizierung und Autorisierung. Systemsicherheit und -integrität sowie die Umsetzung von Datenschutzrichtlinien werden so selbstständig durch das System aufgrund vorgegebener Regeln gewährleistet.
6. **kennt seine System-Umgebung/-umwelt** und den Kontext, in welchem es dementsprechend nach sinnvollen Regeln optimal mit Nachbarsystemen interagiert. Ziel ist es, nicht eine reine Datenflut, sondern nützliche Informationen zur richtigen Zeit zu liefern. Ein anschauliches Beispiel ist der Aufruf einer komplexen Website über ein einfaches Handy. Ein autonomes System ordnet den Kommunikationspartner richtig ein und stellt eine sinnvolle Menge an Informationen der Website zur Verfügung.
7. muss in einer **heterogenen Systemlandschaft** und nicht ausschließlich in einer abgekapselten Umgebung funktionieren. Alle an das System angeschlossenen Komponenten, unabhängig davon, wie speziell die Aufgabe des Gerätes oder der Software auch sein mag, müssen über einheitliche Schnittstellen angepasst und reguliert werden können. Daher ist die Unterstützung offener Standards unumgänglich.

8. **verbirgt die Komplexität** des Systems und seiner autonomen Eigenschaften vor dem Endnutzer. Eine vom Nutzer gestellte Aufgabe muss von einem autonomen System selbstständig gelöst werden, ohne dass sich der Benutzer mit Details der Lösungsmethode oder der Implementation auseinanderzusetzen hat.

Einige der Aspekte, wie bspw. Kenntnisse über sich selbst und die Umwelt, sind vom Prinzip her bereits aus dem Bereich der künstlichen Intelligenz bekannt und dort intensiver Forschungsgegenstand gewesen. Allerdings sind sie im Kontext des Autonomic Computing in dieser Form als neu anzusehen [StBu03]. Hierbei sei allerdings auch zu beachten, dass es sich lediglich um allgemeine Anforderungen handelt, die ein autonomes System ausmachen, in der Praxis allerdings nicht immer einheitlich verwendet werden. Diese idealen Eigenschaften eines autonomen Systems sind derzeit vor allem auf Grund der hohen Dichte von proprietären Lösungen innerhalb einer großen IT-Infrastruktur noch weitgehend unverwirklichbar. Doch bereits die grundsätzliche Vision und die darauf basierenden ersten Realisierungsbestrebungen vieler IT-Unternehmen sind ein wichtiger Schritt nach vorne.

Es sei auch darauf hingewiesen, dass diese einzelnen Eigenschaften nicht streng voneinander getrennt werden können, sondern an den Grenzen ineinander übergehen. Einzelne Komponenten bzw. Funktionen eines Systems können mehreren Merkmalen zugeordnet werden. So kann bspw. die automatische Zuweisung von Speicher sowohl als Konfigurations- als auch als Optimierungsproblem verstanden werden. Derart umfasst die Fähigkeit, sich selbst zu heilen häufig auch die Erkennung und Beseitigung von Effekten von Angriffen auf ein System. Des Weiteren können Angriffe bei schlechtem Schutz beispielsweise Probleme verursachen, die wiederum andere Konzepte (Heilung oder sogar Optimierung) beeinflussen.

Nichtsdestotrotz umreißen diese Merkmale sinnvoll die Anforderungen an ein autonomes System. Sie lassen sich zur leichteren Abbildung bestehender Systeme und zur besseren Einschätzung der Autonomiestufe nach [IBM06e] auf vier wesentliche Kerneigenschaften reduzieren:

- Selbst-Konfiguration (Self-Configuring),
- Selbst-Heilung (Self-Healing),
- Selbst-Optimierung (Self-Optimizing) und
- Selbst-Schutz (Self-Protecting).

Aufgrund der Anfangsbuchstaben der englischen Begriffe ist in diesem Zusammenhang auch oftmals von dem Akronym **CHOP** zu lesen [KeCh03]. In [EPBM03] werden diese um die Selbst-Organisation (Self-Organizing) und die Selbst-Kontrolle (Self-Inspecting) ergänzt, um insbesondere den Anforderungen an Datenbank-Management-Systeme gerecht zu werden. Obwohl zwischen den Bereichen des Selbst-Management keine Unabhängigkeit besteht und die Übergänge oft fließend sind, erscheint eine solche funktionelle Grobeinteilung sinnvoll, da die Intention, eine Änderung am System vorzunehmen, von Bereich zu Bereich verschiedene Ursachen hat.

2.2.3 Automatisierung durch Evolution (statt Revolution)

Das große Ziel des Autonomic Computing ist die Beherrschung der Komplexität von IT-Systemen durch Selbst-Verwaltung. Dass dies nicht über Nacht erreicht werden kann, versteht sich von selbst. Prinzipiell lassen sich zwei mögliche Herangehensweisen unterscheiden, ein autonomes System zu entwickeln und zu etablieren. Zum einen kann ein

autonomes System von Grund auf neu entwickelt und anschließend in Betrieb genommen werden. Alternativ können bestehende Systeme schrittweise um autonome Mechanismen für Prozesse, Komponenten und Teilsysteme angereichert werden. Dabei ist von Vorteil, dass Unternehmen bzw. Entwickler nicht komplette IT-Infrastrukturen umstellen müssen, sondern klein anfangen, autonome Mechanismen bei Bedarf und mit wachsenden Erfahrungen in vorhandene Strukturen einbinden können, um somit eine immer größere Anzahl autonomer Eigenschaften abzudecken.

Auch IBM propagiert die iterative, langwierige Vorgehensweise, an deren Ende die Selbst-Verwaltung von IT-Systemen steht und spricht in dem Zusammenhang von einem evolutionären Prozess autonomer Computersysteme [GaCo03]. Bis jedoch der höchste Reifegrad der Autonomie erreicht ist, müssen unterschiedliche Zwischenziele realisiert und eine Reihe von Evolutionsstufen durchlaufen werden [Wor04]. Systeme können daher nach Autonomiegraden unterschieden und eingeteilt werden. Die in [IBM06e] postulierten fünf Stufen auf dem Weg zum Autonomic Computing sind in **Abbildung 2.6** dargestellt und können zusammengefasst folgendermaßen charakterisiert werden.

Das grundlegende System (**Basic Level** 1) repräsentiert die gegenwärtige Situation, erfüllt die gestellten Mindestanforderungen in Bezug auf die Funktionalität und bildet den Ausgangspunkt für die weiteren Stufen. Auf dem Basic Level wird das Management ausschließlich manuell durch hochqualifiziertes IT-Personal übernommen. Administratoren verwalten jedes Teilelement des Systems in Abhängigkeit von Produktdokumentationen und Systemberichten für sich, richten es ein, überwachen es und ersetzen es gegebenenfalls durch eine andere Komponente. Sie sind somit für die Sammlung und Analyse von Daten sowie die daraus zu ziehenden Schlüsse und Maßnahmen selbst verantwortlich.

Ein verwaltetes System (**Managed Level)** wird durch Verwaltungswerkzeuge und erste automatisierte Mechanismen unterstützt und stellt einen ersten Schritt in Richtung einfacherer Administrierbarkeit dar. Auf dem Managed Level sind vor allem Monitoring-Aufgaben automatisiert, um selbstständig Informationen über die unterschiedlichen Ressourcen bzw. Systeme zu sammeln. Dadurch können die notwendige Zeit zum Sammeln und Darstellen der Informationen reduziert, somit die Administratoren in diesem Aufgabenbereich entlastet und die Produktivität erhöht werden. Darüber hinaus weiß das System zunehmend mehr über sich selbst. Dennoch muss das IT-Personal weiterhin eine Vielzahl an Analysen durchführen und große Teile der Aufgaben manuell bearbeiten.

Ein vorhersagendes System (**Predictive Level**) ist in der Lage, seine Ressourcen und die Umwelt im Detail durch Sammlung und Analyse von Daten zu erfassen, auf der Ebene einzelner Komponenten automatisch Interdependenzen zwischen Entscheidungen und evtl. bestimmte Muster zu erkennen und daraus Handlungsempfehlungen abzuleiten. Diese können dem Nutzer schließlich zur Bewertung oder etwaigen Genehmigung bzw. Ausführung vorgeschlagen werden und helfen, schnellere, bessere Entscheidungen zu treffen. Die Systeme prüfen lediglich die Vorschläge und veranlassen bei Einverständnis deren automatische Ausführung. Dadurch verringert sich die Abhängigkeit von hochqualifiziertem und damit teurem Personal, es sind jedoch erweiterte Maßnahmen zur Einführung von Schnittstellen und Technologien zur Vereinheitlichung unterschiedlicher Systemkomponenten notwendig.

Bei dem adaptiven System (**Adaptive Level**) werden die autonomen Mechanismen dahingehend erweitert, dass neben der automatisierten Datensammlung und -analyse auch notwendige (Re-)Aktionen vom System selbst umgesetzt werden. Die nur noch ein Minimum an menschlichem Eingreifen benötigenden Entscheidungen basieren auf den gesammelten Informationen, dem verfügbaren Wissen über die relevanten Geschehnisse,

aber vor allem auf den durch die Administratoren spezifizierten (Vorgehens-)Richtlinien, den sog. Service Level Agreements.

Mit der fünften und letzten Stufe ist schließlich das Ziel des autonomen Systems (**Autonomic Level**) erreicht. Es umfasst die Funktionalität des Adaptive Level, orientiert sich jedoch an abstrakten Strategien, wie z.B. Geschäftspolitiken bzw. Unternehmensrichtlinien und -zielen von Unternehmen, zur Steuerung sämtlicher Aktionen im IT-System. Die Aufgabe des IT-Personals ist es diese Richtlinien festzulegen bzw. anzupassen. Manuelles Eingreifen, etwa zum Zweck der Konfiguration oder Optimierung, ist nicht mehr notwendig.

Diese Einteilung kann abgesehen von ganzen Systemen natürlich auch herangezogen werden, um die Autonomität von Teilsystemen mit ihren entsprechenden Subkomponenten zu bewerten. Mit jeder zunehmenden Ebene nimmt auch der Automatisierungsgrad zu und umso mehr Verwaltungsaufgaben kann das System selbstständig bewerkstelligen. Gleichzeitig sinkt damit der manuelle Aufwand, der durch IT-Experten realisiert werden muss. Die vollständige Umsetzung eines selbstverwaltenden Systems liegt zwar noch in weiter Ferne, die Integration von bereits umsetzbaren autonomen Eigenschaften in einzelnen Softwarekomponenten und Subsystemen geschieht jedoch in einer Vielzahl von Produkten [StHi05]. Die Mehrzahl der heutigen Systeme kann bereits zwischen Level 2 und 3 angesiedelt werden. Obwohl diese Klassifizierung eine gute Einordnung heutiger Systeme ermöglicht, wird sie von anderen Herstellern und Autoren eher selten übernommen.

	Basic Level 1	Managed Level 2	Predictive Level 3	Adaptive Level 4	Autonomic Level 5
Characteristics	Rely on system reports, product documentation, and manual actions to configure, optimize, heal and protect individual IT components	Management software in place to provide consolidation, facilitation and automation of IT tasks	Individual IT components and systems able to monitor, correlate and analyze the environment and recommend actions	IT components, individually and collectively, able to monitor, correlate, analyze and take action with minimal human intervention	Integrated IT components are collectively and dynamically managed by business rules and policies
Skills	Requires *extensive, highly skilled* IT staff	IT staff *analyzes and takes actions*	IT staff *approves and initiates actions*	IT staff *manages performance* against SLAs	IT staff *focuses* on enabling business needs
Benefits	Basic requirements addressed	Greater system awareness Improved productivity	Reduced dependency on deep skills Faster/better decision making	Balanced human/system interaction IT agility and resiliency	Business policy drives IT management Business agility and resiliency
	Manual				Autonomic

Abbildung 2.6: IBM Maturity Model - Der Weg zum Autonomic Computing [IBM03a]

2.2.4 Referenz-Architektur für das Autonomic Computing

Das autonome Management eines ganzen Systems offenbart eine Menge an Herausforderungen, wie das Einbeziehen von vielen heterogenen Komponenten, Ressourcen und Teilsystemen sowie die Sicherstellung, dass alle Prozesse und getroffenen Entscheidungen mit den festgelegten Geschäftsregeln und Performance-Zielen konform gehen. Der in [IBM06e] vorgeschlagene Architekturentwurf für autonome Computersysteme bildet sowohl die konzeptionelle Grundlage vieler universitärer Forschungsprojekte als auch die

Basis für Weiterentwicklungen und Adaptionen in der IT-Industrie. Entwicklungen der Firma IBM repräsentieren in diesem Bereich quasi den technologischen Vorreiter und unternehmen vielversprechende Anstrengungen in Richtung eines ausgereiften und weitgehend autonomen Gesamtsystems.

Zentraler Baustein autonomer IT-Systeme ist demzufolge der sogenannte **Autonomic Manager**. Dieser kleinste Baustein autonomer Systeme ist verantwortlich für die optimierte Nutzung einer einzelnen, dedizierten Ressource. Bei diesen verwalteten Ressourcen kann es sich um weitere Autonomic Manager oder Ressourcen in Form von Hard- und Softwarekomponenten unterschiedlichster Granularität handeln, z.B. Datenbank-Seiten, Datenbanken, eine Netzwerkkarte oder aber auch eine Menge allokierten Hauptspeichers. Jede kontrollierte Ressource stellt zu diesem Zweck zwei Arten von Schnittstellen (Touchpoints) zur Verfügung: **Sensoren** und **Effektoren**. Sensoren stellen Mechanismen zur Erfassung von Informationen über den Status und Statuswandel des zu verwaltenden Elements bereit. Sensoren können aufgerufen werden, um den aktuellen Status wiederzugeben oder sie geben unverlangt die Informationen wieder, wenn sich der Zustand des Elements signifikant ändert. Effektoren bieten Mechanismen zur Änderung des (Konfigurations-)Zustands des zu verwaltenden Elements.

Sensoren und Effektoren unterstützen nach [KrSt05] vier Arten der Interaktion.

- *Request-Response* (polling) entspricht dem klassischen Anfrage-Antwort-Schema. Über einen Sensor wird ein aktueller Wert abgefragt, der an den Anfragesteller als Antwort zurückgegeben wird.
- *Send-Notification* (pushing) wird verwendet, wenn eine bestimmte Instanz umgehend von einem speziellen Vorfall informiert werden soll. Die Veranlassung dazu erfolgt durch die verwaltete Ressource selbst. Sie stellt fest, dass ein spezielles Ereignis aufgetreten ist und benachrichtigt daraufhin eine zuvor bei der Ressource registrierte Instanz.
- Mit Hilfe der *Perform-Operation* können Änderungen an den Ressourceneinstellungen vorgenommen und damit das Verhalten einer bestimmten Ressource beeinflusst werden.
- Bei der *Solicit-Response*, einer Kombination aus Send-Notification und Perform-Operation, hingegen wird die Kommunikation von der Ressource aus veranlasst. Diese fragt bei einer ihr übergeordneten Stelle nach Informationen und erhält diese daraufhin.

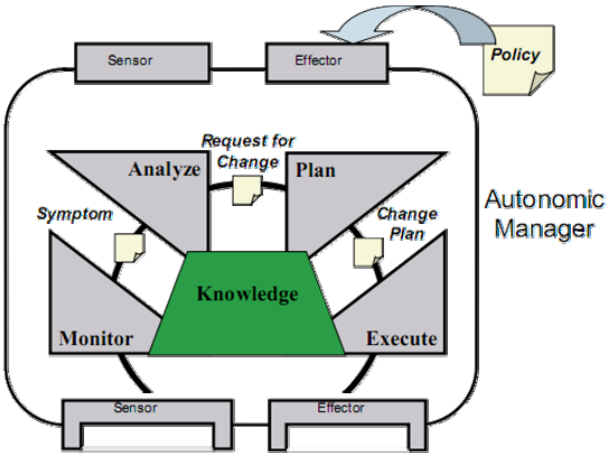


Abbildung 2.7: Funktionale Details eines Autonomic Manager [Mil05a]

Um autonom agieren zu können, muss ein System Funktionen bereitstellen, um Daten/Informationen über das System zu sammeln, diese bei Auftreten von Problem-Situationen zu analysieren und schließlich entsprechende Schritte einleiten, um darauf angemessen zu reagieren. Dieses Verhalten wird durch den Autonomic Manager mittels dem aus der Kontrolltheorie bekannten und in Abschnitt 2.2.1. vorgestellten Konzept einer rückgekoppelten Kontrollschleife implementiert, die kontinuierlich folgende vier in **Abbildung 2.7** dargestellten Phasen durchläuft. Man bezeichnet diese Kontrollschleife auch kurz nach den englischen Phasenbezeichnungen (Monitor, Analyze, Plan, Execute) als **MAPE**-Schleife [IBM03a]. Hierbei sei auf die offensichtlichen Parallelen der MAPE-Loop zum bereits bekannten iterativen, jedoch manuellen und vornehmlich reaktiven Vorgehen eines DBAs bei der Performance-Optimierung (Abschnitt 2.1.2) hingewiesen, die anhand unserer Ausführungen in [Hen07] detailliert nachgelesen werden können.

In der **Monitor**-Phase, dem Ausgangspunkt eines jeden Durchlaufs der Kontrollschleife, werden mit Hilfe der Sensoren kontinuierlich Informationen über die Konfiguration, den Zustand/Status und das Laufzeitverhalten der überwachten Ressourcen ausgelesen. Dies können je nach Typ bspw. Hardware-Informationen (CPU-Geschwindigkeit), Performance-Indikatoren (buffer hit ratio) oder Umgebungs-Informationen (Systemtemperatur) sein. Diese Daten werden gespeichert, aggregiert, korreliert und gefiltert, damit Probleme verursachende Symptome identifiziert und im Anschluss von der Analyze-Komponente ausgewertet werden können. Symptome sind in der Wissensbasis gespeicherte Indikatoren für potentielle Probleme, auf die unter Umständen reagiert werden soll. Ein exemplarisches Symptom aus dem Datenbankbereich ist beispielsweise, dass ein Deadlock auftrat.

Die ermittelten Symptome werden in der **Analyze**-Phase dahingehend weiterverarbeitet und ausgewertet, ob sie tatsächlich auf Probleme hinweisen und ob Änderungen zu deren Behebung notwendig sind. Im Falle signifikanter Zustands-Änderungen durch vorherrschende Probleme wird nach Ursachen gesucht und ob ähnliche Probleme schon einmal bestanden beziehungsweise ob und wie diese erfolgreich behoben wurden. Die Analyze-Phase bietet dafür Mechanismen zur Korrelation, Modellierung und Interpretation komplexer Situationen. Sie ist darüber hinaus in der Lage, zukünftiges Verhalten (Trends) vorherzusagen, indem sie durch die Ergründung der System-Umgebung und den in der Monitor-Phase gesammelten (historischen) Daten neues Wissen erlernt. Soll-Ist-Abweichungen von angestrebten Vorgaben können also für die Gegenwart und für die Zukunft ermittelt werden. Ist dies der Fall, wird eine Änderungsanforderung (Change Request) generiert und an die nächste Phase/Komponente gesendet. Dabei handelt es sich um eine Art Anforderungsliste, welche Maßnahmen von Nöten sind und ausgeführt werden sollten, um den kritischen Systemzustand abzuwenden.

Angestoßen durch die Änderungsanfrage werden aus den bisher aufbereiteten Informationen in der **Plan**-Phase konkrete Handlungen und Änderungen (Change Plan) zur Behebung des Problems der verwalteten Ressource bzw. zum Erreichen der von IT-Spezialisten vorgegebenen Zielrichtlinien (Policies) abgeleitet. Dies können Änderungen an der System-Konfiguration sein, aber auch Komponenten, die zusätzlich zur Verfügung gestellt oder solche, die abgestellt werden müssen. Um die gewünschten Änderungen auszuführen, werden mit Hilfe von Regeln derartige Aktionspläne erstellt oder aus einem Repository gewählt. Diese sollen das System wieder in einen laut Policy akzeptablen Zustand bringen bzw. zukünftiges Verlassen dieses Bereichs frühzeitig verhindern. Bei der Planung werden Abhängigkeiten zwischen den Systemkomponenten in die Betrachtung einbezogen, um das System stabil zu halten und suboptimale Zwischenzustände (z.B. Oszillationen) zu vermeiden.

Die **Execute**-Phase stellt Mechanismen zur Ausführung und Überwachung der in der Planungs-Phase bestimmten Aktionspläne zur Verfügung. Über die Effektoren der Ressourcen werden die notwendigen Änderungen vorgenommen bzw. zur Umsetzung zusammen mit der Reihenfolge der auszuführenden Aktionen und Informationen bezüglich des richtigen Timings an die Ressource weitergegeben.

Die Auswirkungen dieser Änderungen werden wiederum durch die Monitor-Phase überwacht. Damit schließt sich der Kreis und der Prozess beginnt erneut. Ein (potentiell unendlicher) Zyklus dieser Phasen ist von Nöten, da nach einer Änderung am System wiederum überprüft werden muss, ob das System nun bessere Leistungsmerkmale aufweist oder sich das Systemverhalten gar weiter verschlechtert hat und somit eine erneute Anpassung der Konfiguration erforderlich ist. Sollte ein Problem weder durch die (wiederholt) getätigten Maßnahmen noch durch das Nachrüsten von Hardware behoben werden können, so ist es evtl. notwendig, die Vorstellungen und Kriterien des Soll-Zustands anzupassen, da sie im Vorfeld vielleicht nicht realistisch genug abgeschätzt wurden. Die Grenzen der Optimierung sind in diesem Fall erreicht und man muss Performanceeinbußen in Kauf nehmen.

Ein wichtiger Bestandteil eines jeden sowohl reaktiv als auch proaktiv agierenden Autonomic Manager ist das **Wissen**, das zur selbsttätigen Verwaltung der Ressource(n) notwendig ist. Dieses Wissen muss in geeigneter Weise formalisiert, repräsentiert und in einer Wissensbasis verfügbar gemacht werden [WRRA08]. Die Phasen der MAPE-Schleife tauschen akkumuliertes Wissen und Daten über die Wissensbasis aus, um die Funktionalität des Regelkreises zu gewährleisten [Rab06].

Dieses gemeinsame, geteilte Wissen steht allen vier Phasen des MAPE-Kreislaufs zur Verfügung und beinhaltet u.a. Informationen über das zu überwachende System, Log-Daten vergangener Aktionen und Probleme, aber auch Metriken, die definierten Symptome sowie Regeln und Richtlinien, wie in bestimmten Situationen zu verfahren ist. Üblicherweise unterscheidet man die folgenden Wissens-Bereiche [Mil05b]:

- **Solution Topology Knowledge**: Beinhaltet Informationen über die einzelnen verwalteten Ressourcen, deren Aufbau, Installation und Konfiguration sowie über deren Verbindungen zur Umwelt.
- **Policy Knowledge**: In diese Kategorie fallen sämtliche (von Experten oder auch dem System erstellten bzw. angepassten) Regeln und Richtlinien zur Entscheidung, ob und auf welche Art und Weise Modifikationen am System vorgenommen werden müssen, wenn in der Monitor-Phase eine Änderung des Systemverhaltens festgestellt wurde.
- **Problem Determination Knowledge**: Neben Daten von Monitoring-Läufen, Symptom-Definitionen und erkannten Problem-Mustern findet sich hierin auch (erlerntes) Wissen bezüglich der Auswirkungen getroffener Entscheidungen.

In der Regel können diese Informationen sowohl von außen durch IT-Spezialisten, als auch (zur Laufzeit) von innen durch alle an der MAPE-Schleife beteiligten Komponenten zur Verfügung gestellt bzw. angereichert oder aktualisiert werden. Die Wissensbasis ist indes nicht auf einen bestimmten Autonomic Manager begrenzt, sondern könnte auch von mehreren bzw. allen Autonomic Manager genutzt werden. In der Konsequenz bedeutet dies einen einheitlichen Wissensstand für alle Komponenten und u.U. die Vermeidung von Mehrfachausführungen gleicher Aufgaben.

Die Betrachtung der Problembereiche durch die Autonomic Manager erfolgt lokal. Zur bereichsübergreifenden Problem-Erkennung und -auflösung ist die Einführung von übergeordneten Autonomic Manager bzw. einer Kommunikationsstruktur möglich.

Die Kommunikation erfolgt typischerweise auf Basis einer oder einer Mischung der nachfolgenden Topologien. **Hierarchisch** angeordnete Autonome Manager bspw. unterstehen einem hierarchisch übergeordneten Root Manager, der sie überwacht und Daten von ihnen sammelt bzw. Daten gezielt verteilt. Der Root Manager an der Spitze einer Hierarchie bietet eine Schnittstelle zum IT-Administrator, der die globalen Geschäftsziele und Policies eingibt, die ihrerseits auf jeder Ebene in ein für das das nächst niedrigere Level verständliches und sinnvolles Detaillierungs- bzw. Abstraktionsniveau übersetzt werden. Über eine **Peer Group** miteinander verbundene Autonomic Manager sind, wie der Name andeutet, gleichgestellt. Informationen können so innerhalb der Gruppe schnell verteilt werden [BMM+03]. Ein dem Root Manager ähnliches Konstrukt für die Verteilung bzw. Freigabe von Daten für bestimmte Autonomic Manager ist somit nicht nötig.

In [IBM06e] ist ein konzeptioneller Vorschlag für eine auf den beschriebenen Topologien basierende Architektur zur Umsetzung des Selbst-Management dargestellt. Demnach wird ein Autonomic-Computing-System in mehrere hierarchisch organisierte und miteinander verbundene Schichten unterteilt. Die hierarchische Topologie spiegelt sich darin zwischen den verschiedenen Ebenen wider, während die Peer-Group-Topologie innerhalb einer Ebene angewendet wird. Der Architektur-Ansatz ist unabhängig von einem konkreten Verwaltungs-Protokoll oder einer Technologie zur Instrumentierung und jede der Schichten kann durch eine Vielzahl von Technologien bzw. Standards umgesetzt werden.

Bis heute existiert noch keine konkrete Implementierung der beschriebenen Architektur in Form eines Gesamtsystems. Eine Vielzahl an Produkten zur Umsetzung der erforderlichen (selbstverwaltenden) Eigenschaften einzelner Komponenten jedoch erlaubt ein kooperatives und sich gegenseitig ergänzendes Miteinander-Arbeiten in einem gemeinsamen Netzwerk.

2.2.5 Standards im Autonomic Computing

„People can't share knowledge if they don't speak a common language."(T. Davenport]

Aufgrund der Natur der Autonomic-Computing-Systeme kann es gesamtheitlich keine proprietäre autonome Lösung eines Herstellers geben. Unternehmen sind sowohl mit heterogenen internen IT-Infrastrukturen als auch mit heterogenen Umgebungen über Plattform-, Unternehmens- und Ländergrenzen hinweg konfrontiert, um markt- und wettbewerbsfähig zu bleiben und den sich ständig wechselnden Anforderungen gerecht zu werden. Damit heterogene Komponenten (Ressourcen, Manager, Wissensquellen etc.), Applikationen und Systeme verschiedener Hersteller in ein autonomes System integriert werden und interagieren können, ist es daher ratsam, an allen vorhandenen Schnittstellen offene Standards zu verwenden. Dies betrifft u.a. Aspekte der Kommunikation und des Datenaustauschs, die Art und Struktur von Sensoren und Effektoren der verwalteten Elemente sowie die Form der Wissensbasis.

Diese Orientierung am Stand der Technik und Wissenschaft, die Nutzung vorhandener und praxiserprobter Vorgehensmodelle sowie die methodische Vereinheitlichung fördern:

- Interoperabilität: Unabhängige, heterogene Systeme arbeiten möglichst nahtlos, ohne gesonderte Absprachen untereinander, zusammen.

- Kommunikation: Vereinfachte Kommunikation verschiedener Komponenten untereinander durch das „Sprechen einer Sprache“. Steht ein einziges, standardisiertes Protokoll für die Kommunikation zur Verfügung, wird es einfacher, Ressourcen dynamisch zu ermitteln, die Management-Fähigkeiten von Ressourcen offen zu legen und schließlich mit ihnen zu interagieren.
- Austauschbarkeit: Versionen, Tools oder ganze Komponenten können sich ändern, die Schnittstellen bleiben davon unberührt.
- Universalität, Flexibilität: Instrumentierung und Überwachung eines beliebigen Systems.
- Abstraktion: Die Abstraktion von technischen Details (Implementierung, Lokation der Daten, Anbieter etc.) erlaubt adäquatere Nachvollziehbarkeit.
- Einsparen von Kosten und Ressourcen: Standards ermöglichen es, den Ressourcenverbrauch und die Ausgangskenngrößen einheitlich offen zu legen. Die komplexe Administration einzelner Applikationen entfällt, die Ressourcenverwaltung wird einfacher und effizienter.

Standards	Einsatzgebiete	
	Autonomic Manager	**Schnittstellen**
Distributed Management Task Force (DMTF)		
Common Information Model (CIM)		x
Web Services Common Information Model (WS-CIM)		x
Applications Working Group		x
Utility Computing Working Group		x
Server Management Working Group		x
Internet Engineering Task Force (IETF)		
Policy - Core Information Model (RFC3060)	x	
Simple Network Management Protocol (SNMP)		x
Organization for the Advancement of Structured Information Standards (OASIS)		
Web Services Security (WS-Security)	x	x
Web Services Distributed Management (WS-DM)	x	x
Web Services Resource Framework (WS-RF)	x	x
Web Services Notifcation (WS-N)	x	x
Java™ Community Process		
Java™ Management Extensions (JSR3, JMX)	x	
Logging API Specifcation (JSR47)		x
Java™ Agent Services (JSR87)		x
Portlet Specifcation (JSR168)	x	
Storage Networking Industry Association (SNIA)		
Storage Management Initiative Specifcation (SMI-S)		x
Global Grid Forum (GGF)		
Open Grid Services Architecture (OGSA)	x	
Open Grid Services Infrastructure (OGSI)	x	
Open Grid Services Common Management Model (CMM-Working Group)	x	x
Grid Resource Allocation Agreement Protocol (GRAAP-Working Group)	x	
The Open Group		
Application Response Measurement (ARM)		x
World Wide Web Consortium (W3C)		
Solution Install Schema	x	

Tabelle 2.1: Organisationen und Standards für das Autonomic Computing (nach [TM06])

Viele der vorherrschenden Standards im IT-Umfeld greifen auch bei der Entwicklung bzw. Integration von Produkten mit autonomen Fähigkeiten. Neue offene Standards für die „Vereinheitlichung“ in heterogenen System-Umgebungen sind jedoch nötig. Zu den wichtigsten/sichtbarsten bereits vorhandenen und in der Entstehung begriffenen Standards speziell für das Autonomic Computing gehören u.a. die Bestrebungen der Distributed

Management Taskforce (DMTF) sowie der Organization for the Advancement of Structured Information Standards (OASIS). Die Mehrzahl der für das Autonomic Computing relevanten Standards sowie die verantwortlichen Organisationen sind ausführlich in [IBM06e, IBM06n, TM06] beschrieben und in **Tabelle 2.1** im Überblick zusammengefasst.

Service-orientierte Architekturen (SOA) und Web Services als technische Grundlage haben sich inzwischen auf breiter Front durchgesetzt und gelten als Mittel der Wahl, wenn es um die architektonische Basis verteilter unternehmensweiter bzw. -übergreifender Anwendungen und die Interaktion zwischen Implementierungen verschiedener Hersteller geht. Bei dem Entwurf der Technologie wurde von den namhaften Beteiligten konsequent auf die Verwendung offener Standards und möglichst einfacher und damit leicht umzusetzender Protokolle geachtet. Aufgrund der wachsenden Akzeptanz, Unterstützung und Tools im SOA- und Autonomic-Computing-Umfeld erscheint es pragmatisch und wünschenswert, die gleichen oder kompatible Web-Service-basierten Standards in beiden Domänen zu verwenden. Eine der großen aufkommenden Standardisierungs-Tendenzen für das Autonomic Computing basiert nunmehr auf dem sprach-, plattform- und herstellerunabhängigen Ansatz der Verwaltung von Web Services als technische Grundlage. Historisch sind zwei grundlegende Bestrebungen zur Beschreibung von Mechanismen und Standards für die Offenlegung von Management-Schnittstellen bzw. -Informationen und das dadurch ermöglichte System-Management, sprich die Überwachung, den event-basierten Nachrichtenaustausch und die Regelung, mittels Web Services entstanden: die Web-Services-Management-Spezifikation der DMTF sowie die Web-Services-Distributed-Management-Spezifikation (WSDM) des OASIS-Konsortiums.

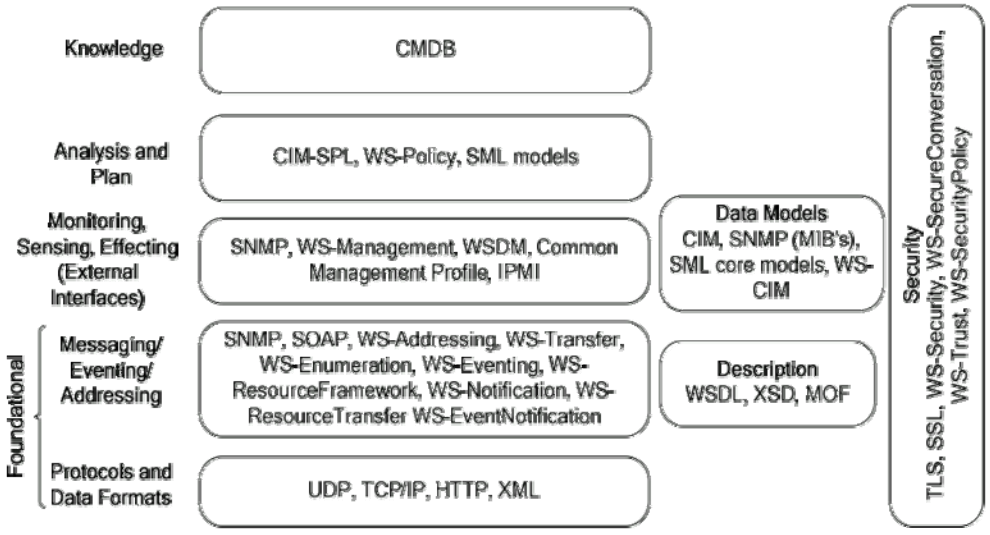


Abbildung 2.8: Einordnung der Standards im Autonomic Computing (nach [TM06])

Abbildung 2.8 unternimmt den Versuch einer groben, nicht vollständigen, veranschaulichenden Zuordnung diverser Standards auf Funktionen und Bereiche der Architektur der Autonomic Manager. Der Nachrichtenaustausch zwischen den autonomen Komponenten basiert auf einer einheitlichen Menge an Protokollen und Daten-Formaten. Damit die strukturierten Informationen zwischen den Elementen ausgetauscht werden können, muss jede Schnittstelle mit ihren angebotenen Operationen, Funktionen, Daten, Datentypen und Austauschprotokollen zunächst in einer maschinenauswertbaren, Plattform-, Programmiersprachen- und Protokoll-unabhängigen Form beschrieben werden.

Auf jeder Ebene des Systems müssen die Objekte und ihre Daten abstrakt und strukturiert repräsentiert werden. Insbesondere die Autonomic Manager benötigen umfassende, einheitliche Datenmodelle der von ihnen überwachten und kontrollierten Ressourcen. Auch das für ihre Analyse- und Planphase benötigte Wissen zur Problem-Erkennung und Entscheidungsfindung muss in geeigneter Form beschrieben und repräsentiert werden können.

Aus Sicht des Autonomic Computing der letzten Jahre stellte das HP Utility Data Center [HP03] eine gewagte und fortschrittliche Unternehmung dar. In einer hochgradig heterogenen Systemlandschaft sollte das aus einer Kombination von Hard- und Software bestehende System ein weitgehend zentrales und teils automatisiertes System-Management von unterschiedlichsten Komponenten (Server, Datenbanken, Netzwerk-Komponenten, Sicherheitssystemen) ermöglichen [Fab03]. Der Misserfolg des Projekts, aufgrund der hohen Einstiegskosten und der wenigen gewonnenen Kunden, unterstreicht dagegen den Eindruck, dass die Zeit für ein solches Gesamtsystem noch nicht reif gewesen zu sein schien. Der umständliche Versuch der Vereinheitlichung von Grund auf unterschiedlicher Systeme durch eine zentrale, übergeordnete Struktur trug wesentlich zum Scheitern bei. Im Nachhinein ist klar, dass die gesamte IT-Landschaft vielmehr durch verstärkte Zusammenarbeit und durch den massiven Einsatz offener Standards homogener werden musste.

2.3 Data Warehousing

„Knowledge is power only if a man knows what facts not to bother with" (Robert Lynd)

Als Basis für Entscheidungs-unterstützende sowie Analyse-orientierte Systeme (Decision Support Systems, DSS [TuAr98]) kann den Data Warehouses eine zunehmend wichtige Rolle in modernen, unternehmensweiten Informationssystemen zugeschrieben werden [ChDa97, Col96]. Im Unterschied zum klassischen Expertensystem, das eine Entscheidung *für* den Benutzer trifft (bzw. dies versucht), soll ein DSS durch Gewährleistung und Aufbereitung notwendiger Informationen eine Entscheidung *mit* dem Nutzer treffen. Dieser Abschnitt soll einen Überblick über Terminologie, Charakteristika, Methodiken und Prozesse im Data-Warehousing-Umfeld geben und so insbesondere die notwendigen Grundlagen für Kapitel 6 und unser im Mittelpunkt der Performance-Daten-Sammlung und Speicherung stehendes Performance Warehouse schaffen.

2.3.1 Terminologie und Abgrenzung zu transaktionalen Systemen

Der Data-Warehouse-Begriff wurde Mitte der 1980er Jahre bei IBM geprägt und mit Information Warehouse bezeichnet. Der Terminus Data Warehouse wurde erstmals 1988 von Devlin verwendet. Eine der Pioniere in diesem Gebiet, William Inmon, prägte Anfang der 1990er den Begriff des Data Warehouse folgendermaßen: *„A warehouse is a subject-oriented, integrated, time-variant and non-volatile collection of data in support of management's decision making process"* [Inm92]. Diese häufig zitierte Definition von Inmon lässt sich in vier Merkmale untergliedern, die ein Data Warehouse besonders auszeichnen und die alle der Entscheidungsunterstützung dienen:

- **Subjektorientierte Daten (thematische Ausrichtung/Zielorientierung)**: Die Datenorganisation und -speicherung ist im Gegensatz zu Online-Transaktionsverarbeitungs-Anwendungen (OLTP), deren Fokus auf bestimmte Geschäftstransaktionen und damit der Erfüllung einer Aufgabe begrenzt ist, auf ein bestimmtes Subjektgebiet (z.B. Umsatz) ausgerichtet. Das Ziel ist demnach nicht

die Unterstützung einzelner Geschäftsprozesse, sondern vielmehr aus allen Datenquellen die Zusammenstellung aller Informationen einer Organisation zu einem bestimmten Thema mit dem Ziel der Unterstützung der Analyse (der Ergebnisse der Transaktionen). Für eine Analyse sind nicht alle möglichen Daten notwendig, sondern nur genau die Daten des Entscheidungsgebiets des Verantwortlichen, für das er in adäquater Zeit Informationen benötigt. Der Anwender braucht also ein Informationssystem, das mehrere Datenquellen vereinigt und einen expliziten Bezug zum jeweiligen Anwendungsfall hat.

- **Integrierte Datenbasis (Vereinheitlichung)**: Die Datenverarbeitung findet auf integrierten Daten aus verschiedenen, nicht notwendigerweise homogenen Datenquellen statt. Die im Data Warehouse als „Datenziel" vorgehaltenen Daten folgen einer einheitlichen Struktur und sind idealerweise frei von syntaktischen und semantischen Inkonsistenzen (aufgrund entsprechender Transformationen, Bereinigungen).
- **Nicht-flüchtige Datenbasis (Dauerhaftigkeit)**: Daten, die einmal in das Data Warehouse übernommen wurden, werden permanent gespeichert und dürfen durch die auf ihnen stattfindenden Analysen nicht mehr verändert oder gar gelöscht werden. Es können daher idealerweise nur neue Daten in das Data Warehouse aufgenommen werden, ohne die bereits vorhandenen zu überschreiben.
- **Historische Daten (Zeitorientierung)**: Die Daten im Data Warehouse repräsentieren nicht nur einen Zeitpunkt, sondern einen zeitlich-historischen Verlauf und werden dafür über einen längeren Zeitraum (u.U. verdichtet) gehalten. Somit sind vor allem Vergleiche über die Zeit möglich.

Es gibt diverse weitere Definitionen des Begriffs Data Warehouse. Die vorherrschende Vielfalt resultiert aus dem Dualismus der zwei grundlegenden Bereiche, die diesen Begriff geprägt haben und denen Fachtermini eigen sind: die technische Seite der Informatik gegenüber der betriebswirtschaftlichen Anwendungs-Seite mit Anforderungen aus der Nutzungsperspektive. Zahlreiche Bestrebungen zur Vereinheitlichung des Begriffs waren wenig erfolgreich. Die anderen Definitionen, bspw. in [BaGu01, Kim96, Leh03], unterscheiden sich vor allem im generellen Zweck eines Data Warehouse sowie im Umfang und Umgang der Daten.

In jüngerer Zeit werden Data-Warehouse-Systeme auch als Business-Warehouse-Systeme oder als Business-Intelligence-Systeme bezeichnet, wodurch die geschäftliche Bedeutung betont werden soll. Die typischen Charakteristika werden jedoch erst deutlich, wenn man Data Warehouse und traditionelle transaktionale, operative Systeme (Online Transactional Processing, OLTP) anhand diverser Kriterien vergleichend gegenüberstellt. In Anlehnung an die Ausführungen in [BaGu01, Inf96] stellen wir in **Tabelle 2.2** die beiden Welten bewusst „schwarz-weiß" gegenüber. In der Realität sind die Grenzen eher fließend und Systeme lassen sich nicht ausschließlich einer Seite zuordnen. Jüngstes Beispiel sind Forschungen und Entwicklungen zum Real-Time-Warehousing [SaCa09].

Der primäre Zweck von transaktionalen bzw. analyseorientierten Systemen hat einen großen Einfluss auf die Organisation und Speicherform der Daten und damit auch auf die Anfrageverarbeitung. Online Transactional Processing (OLTP) bezeichnet die transaktionsorientierte Verarbeitung von Daten. Darunter fallen Aufgaben, wie Flugbuchungen, Bestellungen oder Kontoführung. Allgemein sind dies Bestandteile des operationalen Tagesgeschäfts eines Unternehmens. Die vielen Nutzer (Sachbearbeiter) klassischer transaktionaler Systeme lesen, modifizieren oder löschen Tagesgeschäftsdaten in kurzen und

einfachen Transaktionen, die meist nur wenige Datensätze betreffen. Die wenigen Anwender analyseorientierter Data-Warehouse-Systeme (Manager, Controller, Analysten) hingegen gewinnen häufig ad hoc Informationen aus den Daten, die sie zur Entscheidungsunterstützung benötigen, insbesondere durch langlaufende Lesetransaktionen mit komplexen, berechnenden Anfragen auf Millionen von Datensätzen.

	Transaktional ausgerichtete Anwendungs-Systeme (OLTP)	**Analytisch orientierte Data-Warehouse-Systeme (OLAP)**
Typische Nutzer / Anwendertyp	Ein-/Ausgabe durch Sachbearbeiter, Administrator	Auswertungen durch Manager, Controller, Analysten
Primärer Zweck	Tagesgeschäft, Täglicher Geschäftsbetrieb (Erfassung von Daten)	Analyse, Planung, Entscheidungsunterstützung (Information Retrieval)
Design-Ziel	Performance (hoher Durchsatz), Verfügbarkeit	Intuitive, einfache und flexible Nutzung (hohe Flexibilität)
Interaktionstyp und –dauer	Kurze, einfache Lese-/ Schreibtransaktionen	Lange, komplexe Lese-/ Aggregationsanfragen (periodische Inserts)
Datenvolumen	MB-GB	GB-TB-...
Antwortzeiten	Millisekundenbereich	Mehrere Sekunden bis Minuten
Zugriff	wiederholend/repetetiv, vorhersagbar; weniger Benutzer-orientiert	Ad hoc, jedoch überwiegend bestimmten Mustern folgend
Änderungen/Aktualität Stabilität	Sehr häufig / stets aktuell dynamisch	Periodisch Statisch mit periodischer Ergänzung
Anzahl an (Online-) Nutzern	Viele (1.000+)	Wenige (<100)
Tabellengrößen	Klein	Groß
Bereich einer Anfrage	wenige Datensätze (überwiegend Einzeltupelzugriff)	viele Datensätze (überwiegend Bereichsanfragen)
Datenquellen	Meist nur eine (zentraler Datenbestand)	Viele unabhängige
Typ der Daten/Inhalte	Nicht abgeleitet/sehr detailliert, aktuell, isoliert	Abgeleitet/verdichtet/aggregiert, historisch bis aktuell, integriert, konsolidiert
Normalisierung	Hoch (3NF), redundanzarm	Gering (1NF), z.T. denormalisiert
Datenmodell	Anfage-flexibel/neutral, Applikations-orientiert	Themen-orientiert, Analyse-orientiert

Tabelle 2.2: Vergleich von Anwendungs-Systemen (in Erweiterung von [BaGu01])

Beide Arten von Systemen arbeiten auf Daten aus der Produktionsumgebung eines Unternehmens. Die gesammelten und abgeleiteten Daten eines Data Warehouse stammen jedoch physisch aus einer oder mehreren operativen Datenbanken und anderen Datenquellen (Dateien, Spreadsheets etc.). Die Daten sind konsolidiert, integriert, unveränderlich und meist aggregiert. Durch die Unveränderlichkeit der Daten und Vereinigung mehrerer Datenquellen wächst das Datenvolumen von Mega- oder Gigabyte in transaktionalen Anwendungen auf Datenvolumina bis in den Terabyte-Bereich bei Data-Warehouse-Systemen. Im transaktionalen Betrieb sind die Daten meist nicht abgeleitet, zeitaktuell, aus einer Datenquelle und andauernden Modifikationen (dynamisch) unterlegen.

Die unterschiedlichen Anfragemuster und Datenvolumina erfordern zudem ein an den Einsatzzweck angepasstes Datenbank-Design im Sinne eines Wandels des DB-Schemas von der transaktionalen, Anfrage-flexiblen Applikations-Orientiertheit hin zur Themen- und Analyse-Orientierung sowie die adäquate Anwendung von Zugriffs-Strukturen. Während Anfragen im Data-Warehousing-Fall i.d.R. einen Großteil des Datenbestands betreffen, finden im transaktionalen Fall weitestgehend Einzeltupel-Zugriffe statt. In beiden Anwendungsfällen wird eine kurze Antwortzeit erwartet. Bei einer Anwendung, die sehr große Daten-Mengen in komplexen Anfragen verwendet, kann eine für transaktionale Systeme inakzeptable Forderung nach Antwortzeiten im Sekunden- bis Minutenbereich durchaus akzeptabel sein.

2.3.2 Eine (Referenz-)Architektur für das Data Warehousing

Ein Data Warehouse kann all seine o.g. Eigenschaften selten alleine zur Verfügung stellen. Deshalb ist ein Data Warehouse in ein Data-Warehouse-System eingebettet. Es umfasst alle für die Datenbeschaffung, Integration, Speicherung und Analyse notwendigen Komponenten. Der Data-Warehouse-Prozess, auch Data Warehousing (DWH) genannt, beschreibt den dynamischen Vorgang, angefangen beim Datenbeschaffungs-Prozess über das Speichern bis zur Analyse der Daten, d.h. den Fluss und die Verarbeitung der Daten aus den Datenquellen bis zum Analyse-Ergebnis beim Anwender. Ein Data-Warehouse-System ist also mehr als die Summe seiner Komponenten, d.h., erst mit dem Data-Warehouse-Prozess kann es seine Aufgaben erfüllen [BaGu01].

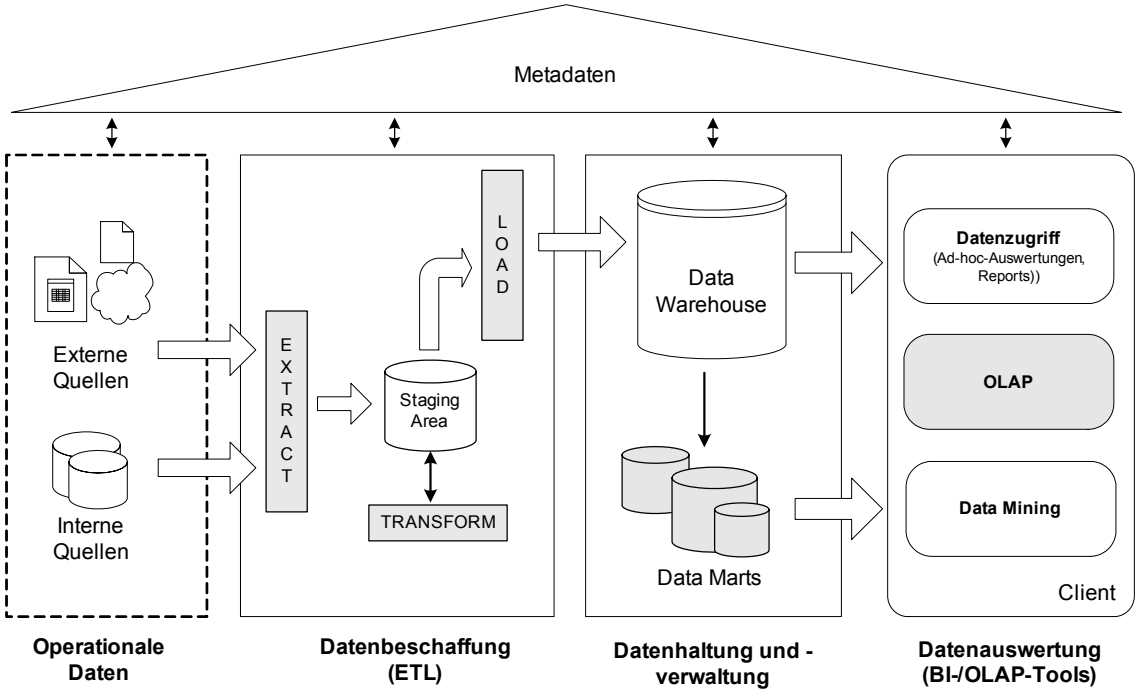


Abbildung 2.9: Phasen und Komponenten des Data Warehousing (angelehnt an [Wie05])

In Analogie zu einem Lager in der Material-Logistik kann ein Data Warehouse in der Informations-Logistik als Datenlager [BaGu01] angesehen werden. Im ersten Fall werden gekaufte und eigene Produkte gelagert, im Data Warehouse werden betriebsinterne und gegebenenfalls externe Daten abgelegt. Die Entwicklung eines Data-Warehouse-Systems kann ebenso mit der Errichtung eines physischen Warenlagers verglichen werden [Perk04]. Zur Bewältigung der vorherrschenden Komplexität ist ein modularer Infrastruktur-Ansatz förderlich. Ähnlich wie ein Gebäude nach einem technischen Bauplan gefertigt wird, so soll

eine DWH-Referenz-Architektur die Gesamtstruktur mit ihren Komponenten, deren Kommunikation und Koordination sowie die Abarbeitung vorgeben, um den Aufbau einer DWH-Lösung einerseits zu vereinheitlichen und andererseits zu vereinfachen. Die Zerlegung des komplexen DWH-Monolithen in kleine, besser verwalt- und wartbare Komponenten dient zum einen der Flexibilität, der Verständlichkeit und der vereinfachten Anpassung. Zum anderen lassen sich dadurch verschiedene Systeme objektiv vergleichen, um die Schwächen und Stärken der Systeme besser einzuordnen.

Die Referenz-Architektur genügt dem Prinzip der Abstraktion, d.h. dass hier von konkreten Details abgesehen wird, um allgemeine Merkmale und Eigenschaften darzustellen und zu erkennen. In diesem Fall abstrahiert sie von technischen Details, von Inhalten des Data Warehouse und vom spezifischen Zweck. Wie bei allen Formen der Modellierung, gibt es nicht die eine exklusive und einzig richtige (Architektur-)Lösung. Vielmehr sind viele Instantiierungen und Varianten denkbar. Im Folgenden soll daher eine mögliche Referenz-Architektur von Data-Warehouse-Systemen erläutert werden. Die Architektur lässt sich zunächst in vier Ebenen untergliedern: eine (a) Datenerfassungs-Ebene mit der Schnittstelle zu den operativen Systemen, eine (b) Datenhaltungs-Ebene mit dem eigentlichen Data Warehouse und eine (c) Datenbereitstellungs-Ebene mit den Schnittstellen zu den Endanwendungen und den Präsentationswerkzeugen. Allen Ebenen sind Administrationsfunktionen zugeordnet, die durch eine (d) Metadaten-Ebene unterstützt werden. Einen zusammenfassenden Überblick über vorhandene Komponenten sowie deren grobe Interaktion gibt die **Abbildung 2.9**.

A. Datenbeschaffung (Extract-Transform-Load)

In erster Linie dient eine Data-Warehouse-Architektur der Zusammenführung heterogener Quellsysteme bzw. deren Daten zu einem integrierten und analysefähigen Gesamtdatenbestand. Die daran beteiligten Komponenten bilden zusammen den Datenbeschaffungsbereich. Dies umfasst vor allem die Extraktion der relevanten Daten aus den Quellsystemen, Transformation und gegebenenfalls Datenbereinigung in einem temporären Arbeitsbereich sowie Laden in das Data Warehouse. Dieser Schritt wird auch Extract-Transform-Load-Prozess (ETL-Prozess) genannt.

Der Datenfluss im Data-Warehouse-System geht von den operationalen, in Bezug auf Struktur, Inhalt und Schnittstellen meist heterogenen **Datenquellen** mit den darin enthaltenen Primärdaten aus. Aus mehreren zur Verfügung stehenden Datenquellen müssen die für die späteren Analysen geeigneten und ausreichenden Quelldaten ausfindig gemacht werden. Der Auswahl der Datenquellen und der Qualität der Daten aus den Quellsystemen kommt daher eine besondere Bedeutung zu. Faktoren für diese Auswahl können neben dem Zweck des Data Warehouse, der rechtlichen oder technischen Verfügbarkeit, dem Preis für den Erwerb von (Unternehmens-externen) Daten vor allem die Qualität der Quelldaten sein. Die Datenquellen lassen sich zudem nicht ausschließlich nach ihrer Qualität und der Herkunft (intern, extern), sondern auch nach weiteren in Abschnitt 4.1.1 gelisteten Kriterien unterscheiden.

Anschließend werden die relevanten Daten aus den verschiedenen ermittelten Datenquellen mittels **Extraktion** in einen temporären, nicht-persistenten Arbeitsbereich (engl. Staging Area) übertragen und transformiert. Die Extraktion kann dabei periodisch, auf Anfrage, Ereignis-gesteuert (z.B. bei Erreichen einer definierten Anzahl von Änderungen) oder auch sofort erfolgen.

Da es aufgrund der typischerweise großen Datenvolumina und Historisierung der Daten eher unpraktikabel erscheint, alle Daten immer wieder auf einmal zu extrahieren, sollten Aktualisierungen des Data Warehouse inkrementell erfolgen. Sogenannte Monitore

unterstützen diesen Ablauf (changed data capture) durch die Identifikation und Auswahl der geeigneten, geänderten Daten in den Datenquellen. Nach [VGD99] können Trigger-, Replikations-, Zeitstempel-, Log- und Snapshot-basierte Monitoring-Strategien unterschieden werden.

Da in Unternehmen häufig mehrere (Quell-)Systeme eingesetzt werden, liegen zum großen Teil unterschiedliche Datenstrukturen vor, so dass die Daten nach der Extraktion zunächst durch eine **Transformation** (Aufbereitung und Anpassung für das Laden) in einen einheitlichen Zustand gebracht werden müssen. Das bezieht sich sowohl auf die strukturellen Aspekte, wie *Schema-Integration,* als auch auf inhaltliche, wie *Daten-Integration* und Datenbereinigung. Es müssen Schemakonflikte (z.B. Synonyme/Homonyme oder strukturelle Differenzen) erkannt, aufgelöst und Daten in ein einheitliches Format (Datentypen, Datumsangaben, Maßeinheiten, Kodierungen) überführt werden. Weiterhin sind durch fehlerhafte oder fehlende Werte bzw. Redundanzen entstandene Verunreinigungen im Datenbestand aufzuspüren und zu beseitigen. Eine derartige Pflege von Daten wird als Datenbereinigung (Data Cleaning) bezeichnet und lässt sich weiterhin noch unterteilen in das Data Scrubbing (Ausnutzung Domänen-spezifischen Wissens zum Erkennen von Verunreinigungen) und das Data Auditing (Anwendung von Data-Mining-Verfahren zum Aufdecken von Regeln und Aufspüren von Abweichungen).

Um die Quellen und das Data Warehouse zu entlasten und nicht zu beeinflussen, wird die Transformation und Integration in einem temporären Arbeitsbereich, der sog. **Data Staging Area** ausgeführt. Nach Abschluss befinden sich in dieser zentralen Datenhaltungskomponente des Datenbeschaffungsbereichs die für die spätere Speicherung und Auswertung geeignet bereinigten und angemessen aufbereiteten Daten zur Weiterleitung in das Data Warehouse.

Der eigentliche **Lade**-Vorgang kann dabei online oder offline erfolgen und macht sich zumeist spezielle Ladewerkzeuge (z.B. Bulk Load) des jeweils zugrundeliegenden Datenbank-Management-Systems zu Nutze, um die extrahierten Änderungen in den Quellen in Form von großen Daten-Mengen in das Data Warehouse zu übertragen, ohne existierende Data-Warehouse-Daten zu überschreiben. Generell unterscheidet man zwischen dem ersten Laden zur Initialisierung des Data Warehouse, bei dem alle Daten aus den Quellsystemen übertragen werden müssen und den späteren, regelmäßigen Aktualisierungen, bei denen nur die geänderten Daten geladen werden müssen.

B. Datenhaltung

In dem **Data Warehouse** liegen die Schemata und Daten der verschiedenen Quellsysteme integriert und langfristig gespeichert vor. Es stellt somit eine flexible, erweiterbare und in Struktur an die Analyse-Bedürfnisse orientierte (meist multidimensionale) Sicht auf die Daten bereit. Um Forderungen an Performance, Unabhängigkeit der Daten und langfristiger Verfügbarkeit einhalten zu können, werden die Daten separat physisch (und damit bewusst redundant) abgelegt, anstatt direkt auf die Datenquelle zuzugreifen.

Ein **Data Mart** (DM) baut auf dem Datenbestand des Data Warehouse auf, dient der Bereitstellung einer, z.B. für einzelne Abteilungen sinnvollen, inhaltlich auf einen Betriebsbereich oder wenige Betriebsbereiche beschränkten Sicht (Teilmenge) auf das Data Warehouse und fungiert als eine Art Cache. Es wird keine weitere Bereinigung oder Normalisierung der Daten benötigt, so dass die inhaltliche und strukturelle Konsistenz der Data-Warehouse-Daten und der DM-Ergebnisse gewährleistet wird. **Tabelle 2.3** fasst die Unterschiede zwischen beiden Konzepten zusammen.

Für die logische oder auch physische (Last-)Verteilung der Data-Warehouse-Daten sprechen Gründe wie Performance, Eigenständigkeit und Datenschutz sowie eine schnelle und kostengünstigere Realisierung. Dies erkauft man sich wiederum durch zusätzliche Redundanz und Transformationsaufwand. Typischerweise beinhalten (abhängige) DMs die Granularität verringernde, verdichtete Extrakte des Data Warehouse, da in den meisten Anwendungsfällen nicht dessen Feingranularität benötigt wird. Zudem kann eine strukturelle Teilmenge des Data-Warehouse-Schema in Analogie zur relationalen Projektion oder auch eine datenbasierte/inhaltliche Teilmenge in Analogie zur relationalen Selektion in den DM repliziert werden. Der Vergleich zu den relationalen materialisierten Sichten liegt daher nahe. Data Marts können aber auch unabhängig und die Vorstufe zu einem unternehmensweiten Data Warehouse sein, indem für einen Unternehmensbereich ein Data Mart aufgebaut wird, in das nach und nach andere Unternehmensbereiche integriert werden.

Merkmale	Data Warehouse	Data Mart
Philosophie	Anwendungs-neutral; zentralisiert	Anwendungs-orientiert; dezentralisiert
Ausrichtung	Unternehmensweit	Einzelne Abteilungen, Nutzer(gruppen)
Datenbanktechnologie	Meist relational	Meist multidimensional
Datenmenge	Hoch (großer Zeit-Horizont))	Niedrig (wenige historische Daten)
Detaillierungsgrad / Granularität	Hoch	Niedrig
Optimierungsziel	Umgang mit großen Daten-Mengen	Abfragegeschwindigkeit
Anzahl	Eines bzw. sehr wenige	Mehrere
Weitere Charakteristika	Flexibel Lang-Zeit, strategisch Groß Einzelne komplexe Struktur	Restriktiv Kurz-Zeit, taktisch Beginnt Klein, wird größer Viele semi-komplexe Strukturen, die gesamt komplex sind

Tabelle 2.3: Gegenüberstellung von Data Warehouse und Data Mart (nach [BaGu01])

Ähnlich wie die Daten aus den Quellen regelmäßig aktualisiert und ins Data Warehouse transferiert werden müssen, ist eine (vereinfachte ETL-)Strategie für DMs zu entwickeln. Eine gute Übersicht und ein Vergleich der Techniken zur Erstellung und Aktualisierung von DM ist in [Liu99] zu finden. Die Autoren argumentieren einerseits, dass der DM als eine natürliche und leistungsfähige Erweiterung zum Data Warehouse, jedoch nicht als Alternative gesehen werden kann. Andererseits zeigen sie auch auf, dass die Grenzen zwischen beiden Konstrukten, ähnlich wie zwischen klassischen OLTP und OLAP-Systemen, immer mehr verschwimmen.

C. Datenauswertung

Der Wert des Data Warehouse wird durch seinen integrierten Datenspeicher und die darin enthaltenen Informationen repräsentiert. Die Datenauswertung erfolgt durch die Analyse der Daten im jeweiligen Data Mart bzw. durch die Versorgung nachgelagerter Anwendungs-Systeme. Derartige Analysewerkzeuge, sog. Business Intelligence Tools, bilden die Schnittstelle der Data-Warehouse-Anwendung zum Endanwender und dienen der Präsentation der gesammelten und durch den erheblichen Aufwand des Aufbaus und der Modellierung des Data Warehouse integrierten Daten mit interaktiven Navigations- und Analysemöglichkeiten. Neben der reinen Darstellung der Daten in Form von Tabellen,

Graphen, Diagrammen spielen einfache Tools und Analysewerkzeuge für **Ad-Hoc**-Anfragen, für das **Reporting** und insbesondere **OLAP**- bzw. **Data-Mining**-Tools für interaktive mehrdimensionale Analysen, Navigation, Gruppierung bzw. statistische Berechnungen und Muster-Erkennung eine wichtige Rolle.

Traditionell werden Data Warehouses mittels relationaler DBMS-Technologien implementiert. Relationale **OLAP Server** (ROLAP) bieten den Tools daher, falls nötig, eine multidimensionale Zugriffs-Schicht auf die Daten. Mit Hilfe einer solchen (intuitiven) multidimensionalen Schnittstelle hat der Endbenutzer die Möglichkeit, seine Abfragen ohne detaillierte EDV-Kenntnisse zu formulieren und an ein Data Warehouse weiterzuschicken. Die multidimensionalen Abfragen werden in entsprechende SQL-Statements und mittels entsprechender (Hilfs-)Strukturen umgesetzt. Das relationale Anfrage-Ergebnis wird im Anschluss wieder (in ein multidimensionales Konstrukt) zurücktransformiert und kann dann in der gewünschten Form zur Verfügung gestellt werden. Zu den Haupt-Anforderungen der relationalen Speicherung gehören zum einen die Vermeidung des Verlustes Anwendungs-bezogener Semantik aus dem multidimensionalen Modell (z.B. Klassifikationshierarchien). Zum anderen hat die effiziente Transformation und Verarbeitung multidimensionaler Anfragen in relationale Repräsentationen unter Berücksichtigung der Anfragecharakteristik, des Datenvolumens von Analyseanwendungen und dem Einsatz spezieller Optimierungstechniken adäquat zu erfolgen. Vorteile der relationalen Lösung sind insbesondere die vereinfachte Integration in bestehende relationale Informationssysteme, SQL-Kompatibilität, die hohe Skalierbarkeit, die Unterstützung transaktionaler Konzepte und die platzsparendere Speicherung in relationalen Strukturen.

D. Metadaten und Management

Die zentrale Komponente eines Data-Warehouse-Systems bildet der Data-Warehouse-Manager. Er ist zuständig für die Initiierung, Steuerung und Überwachung der einzelnen Prozesse sowie für die Verwendung, den Zugriff und die Aktualisierung der dafür notwendigen Metadaten. Um die Metadaten verwenden, aktualisieren und austauschen zu können, greifen die verschiedenen Architekturkomponenten über eine vom Manager bereitzustellende Schnittstelle lesend und schreibend auf das Metadaten-Repository zu. Unter Metadaten werden alle für den Aufbau, den Betrieb, die Wartung und die Administration des Data-Warehouse-Systems notwendigen Informationen verstanden, wie beispielsweise über die Herkunft, Zusammensetzung oder auch Regeln für die Transformation und Verdichtung der Daten.

Von besonderem Interesse laut [DoRa00] ist dabei die Konzeption eines umfassenden Metamodells für die einheitliche Repräsentation von allen Warehouse-Metadaten für eine bessere Interoperabilität. Essentiell hierfür ist die konsistente Verwaltung von gemeinsam genutzten Metadaten sowie von Nutzer-spezifischen Metadaten für das Warehouse-Verständnis und die Entscheidungsunterstützung. Das gestaltet sich jedoch aufgrund der Koexistenz vieler heterogener, lokaler Repositories sowie proprietärer, meistens zugeschnittener und damit inkompatibler Metamodelle und der lokalen Nutzung von Metadaten aktuell als schwierig. Die Autoren entwickeln daher ein Shared Repository für die zentrale Verwaltung von gemeinsamen Metadaten und zeigen damit eine verbesserte Metadaten-Interoperabilität durch weniger und einheitliche Austauschschnittstellen und einen kontrollierbareren Metadaten-Fluss und -replikation auf.

2.3.3 Das multidimensionale Datenmodell

Ein Data-Warehouse-System wird meist angelegt, um als Grundlage für eine Vielzahl spezifischer Anwendungs-Systeme zur Entscheidungs-unterstützenden Analyse zu dienen. Das Hauptproblem, das hierbei auftaucht, ist das Design eines geeigneten logischen Schemas, das die natürliche Denkweise des Anwenders/Entscheidungsträgers in Dimensionen und (Klassifikations-)Hierarchien widerspiegelt. Um die Daten insbesondere für die explorative, interaktive Daten-Analyse im Kontext des Online Analytical Processing, kurz OLAP [CCS93], derart zu strukturieren, hat sich das multidimensionale Datenmodell [EdEd97, Kim96, KRRT98] als sehr zweckmäßig erwiesen. Es stellt im Gegensatz zu anderen Datenmodellen besondere Strukturen und Auswertungsmöglichkeiten zur Verfügung, die schon bei der Modellierung einen Analysekontext schaffen.

Wesentlich für das multidimensionale Datenmodell ist die Unterscheidung in qualifizierende und quantifizierende Daten [Shos82]. Die quantifizierenden Daten, oft als **Fakten** oder Maßzahlen bezeichnet, sind die bei der Daten-Analyse in Entscheidungsprozessen im Mittelpunkt stehenden betriebswirtschaftlichen, numerischen Kennzahlen (Erlöse, Gewinne, Verluste etc.). Dabei kann es sich um Basisgrößen (Werte) oder abgeleitete Zahlen (berechnete Werte) handeln. Die qualifizierenden Daten beschreiben die Benutzersicht auf die Daten und werden als **Dimensionen** bezeichnet. Dimensionen verdeutlichen Aspekte des Auswertungskontextes und erlauben die Betrachtung der Kennzahlen aus unterschiedlichen Perspektiven (z.B. zeitlich, regional, produktbezogen). Sie enthalten Attribute, welche die Dimension möglichst genau beschreiben (Dimensions-Elemente) und die hierarchisch aufgefasst eine weitere Unterteilung der Auswertedimensionen in Form von Hierarchien bzw. Konsolidierungsebenen ermöglichen.

Dimensionen dienen demnach der orthogonalen Strukturierung des Datenraums. Der reine Zahlenwert einer Kennzahl ist ohne semantischen Bezug, sprich ohne die ihn klassifizierenden Dimensionen, nichtssagend. Es ist jedoch auch eine multidimensionale Struktur ohne Kennzahlen vorstellbar. Dies kann dann erforderlich sein, wenn nur das Auftreten eines bestimmten Ereignisses aufgezeichnet werden soll, z.B. wenn nur das Faktum, dass ein bestimmter Kunde ein bestimmtes Produkt zu einer bestimmten Zeit gekauft hat, von Interesse ist.

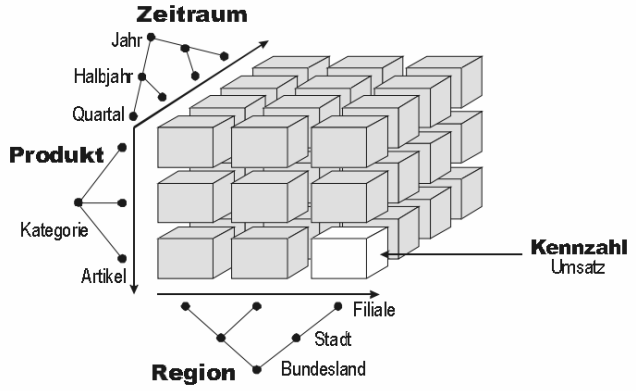


Abbildung 2.10: Darstellung eines exemplarischen Datenwürfels [BaGu01]

Das einem Data Warehouse zugrunde liegende Modell lässt sich am besten mit der „Würfel-Metapher“ veranschaulichen. Die Kombination aus Fakten und Dimensionen wird als (Hyper-)**Würfel** (Cube) bezeichnet. Die Kennzeichen eines Datenwürfels sind seine Kanten, die durch Dimensionen aufgespannt werden und seine Zellen, die eine oder mehrere Kenngrößen beinhalten. Zur Modellierung von Verkäufen in einem Unternehmen

beispielsweise stellen der Zeitraum, die Filialen und Produkte die Dimensionen, der Umsatz der verkauften Artikel in diesem Dimensionskontext den Fakt dar. Der Würfel in **Abbildung 2.10** veranschaulicht ein derartiges Szenario anhand von drei Filialen und drei Artikeln in einem Jahr.

Die Kenngröße ergibt sich somit als eine Funktion der Dimensionen und wird von ihren Werten bestimmt. Die Anzahl der Dimensionen des Datenwürfels wird als seine Dimensionalität bezeichnet. Die Kombination von zwei Dimensionen liefert ein Rechteck, das auch als Tabelle dargestellt werden kann. Im drei-dimensionalen Fall kann der Würfel ebenso anschaulich visualisiert werden. In der Praxis haben sich daneben auch sog. Pivot- bzw. Kreuz-Tabellen zur tabellarischen Repräsentation von Cubes mittels Verschachtelung mehrerer Dimensionen durchgesetzt [BaGu01].

Zur Erfüllung einer der wichtigsten Aufgaben eines Data Warehouse, der Speicherung verdichteter Daten, können Dimensionen mehrere hierarchisch zusammenhängende Merkmale besitzen, die eine weitere Unterteilung der Auswertedimensionen und damit andere Betrachtungsweisen in Form von **Hierarchien** ermöglichen (z.B. Unterteilung der Dimension Zeit in Jahr, Halbjahr, Quartal wie in Abbildung 2.10). Um Informationen mit angemessenem Detaillierungsgrad auf verschiedenen Verdichtungsstufen zur Verfügung stellen zu können, werden auf den Dimensionen jene (Klassifikations-)Hierarchien gebildet, indem die Elemente von Dimensionen gruppiert werden. Elemente mit gleichem Detaillierungsgrad werden als Dimensionsebene oder Level bezeichnet. Dabei enthält die höhere Hierarchieebene (Parent) die aggregierten Werte genau einer niedrigeren Hierarchiestufe (Child). Der oberste Knoten (Top/All) enthält die Verdichtung auf einen einzelnen Wert der Dimension. Auf einer Dimension können eine oder mehrere (unabhängige) Hierarchien festgelegt werden. Generell werden 1:1- oder 1:n-Beziehungen zwischen einzelnen Merkmalen innerhalb einer Dimension in Form von Hierarchien und n:m-Beziehungen in eigene Dimensionen untergebracht. Die n:m-Beziehung zwischen Filiale und Produkt sollte beispielsweise mittels zwei Dimensionen realisiert werden, da jedes Produkt in jeder Filiale gekauft werden kann.

Durch die Dimensionen identifizierte und beschriebene Kennzahlen können durch Anwendung arithmetischer Operationen, Skalarfunktionen oder auch **Aggregationsfunktionen** aus anderen Kennzahlen konstruiert werden. Letztere umfassen die Verdichtung eines Datenbestands durch Ermittlung eines Aggregatwerts aus n Einzelwerten. Als Beispiele seien die auch in SQL vorzufindenden Funktionen SUM, AVG, MIN, MAX, COUNT genannt. Dabei ist zu beachten, dass für jede Kennzahl einzeln die erlaubten Aggregations-Operationen zu definieren sind. Es kann demnach Kennzahlen geben, für die eine Aggregation entweder gar nicht oder nur hinsichtlich ausgewählter Dimensionen möglich bzw. sinnvoll ist.

Insbesondere durch die Größe des Data-Warehouse-Datenbestands und durch die Komplexität der Queries bedingte lange Anfragezeiten besteht Bedarf für **Anfrage-Optimierungstechniken**, wie bspw. Indexierungsmechanismen [GHRU97, ONQu97, ONGr95] oder auch die gebräuchliche Technik zur nicht trivialen Auswahl und Vorberechnung häufig gestellter Queries (materialisierte Sichten, [CKPS95, GHQ95, LeMS95, TeUl97, YaLa87]). In Data Cubes sind die Werte vieler Zellen aus dem Inhalt anderer Zellen berechenbar. [HRU96] nennt drei Möglichkeiten der Implementierung von Datenwürfeln. Im einfachsten Fall können Aggregationen aus Detaildaten zur Laufzeit, sprich bei Anfrage von Zellen, die Werte einer höheren, aggregierten Klassifikationsstufe repräsentieren, berechnet werden. Es wird kein zusätzlicher Speicherplatz benötigt, die Antwortzeiten werden jedoch (ähnlich wie bei der Berechnung normaler Sichten) nachteilig

beeinflusst. Alternativ kann der gesamte Würfel direkt nach jeder Übernahme der Detaildaten in das Data Warehouse oder zu bestimmten Zeitpunkten im Voraus materialisiert werden. Je mehr Zellen vorberechnet und materialisiert werden, umso besser wird die Query-Performance. Obgleich dieser Ansatz die besten Antwortzeiten auf Queries liefert, scheint er für große Würfel keine akzeptable Alternative, da der benötigte Speicherplatz erhöht und die Aktualität u.U. verringert wird. Alternativ kann es auch ausreichend sein, nur einen kleinen Teil des Würfels vorzuberechnen. Die Entscheidung für eine der Alternativen muss komplementäre Anforderungen an Aktualität, Zugriffsgeschwindigkeit und Speicherplatzausnutzung in Betracht ziehen.

Die in einem DWH gespeicherten Daten dienen u.a. der Analyse von historischen Entwicklungen, weshalb der Zeitbezug von wesentlicher Bedeutung ist [ChSt98a, ChSt98b, Inmo96]. Um historische Daten speichern und auswerten zu können, enthält nahezu jeder vorstellbare Datenwürfel eine **Zeit-Dimension**. Nun kann es vorkommen, dass diese Daten allerdings nicht von den Quellsystemen geliefert werden bzw. werden können, da diese in einigen Bereichen nur den jeweils aktuellen „Zustand" des Unternehmens darstellen. In solchen Fällen erzeugt man die historischen Daten bei der Übernahme ins Data Warehouse „selbst", basierend auf dem Intervall der Ladevorgänge.

Im Rahmen der Historisierung stellt vor allem die Strukturdynamik das zentrale Problem dar. Die Fakten besitzen durch die ihnen zugeordnete Dimension „Zeit" implizit eine fortwährende Gültigkeit. Im Gegensatz dazu haben die Dimensionselemente in herkömmlichen Ansätzen jedoch immer den Charakter einer Schnappschuss-Datenbank. Vor allem sich im zeitlichen Verlauf ändernde Klassifikationsstrukturen sind von diesem Problem betroffen. Bereits Kimball [Kimb96] weist im Rahmen von „slowly changing dimensions" auf Dimensionen hin, die logisch von der Zeitdimension abhängig sind. Beispielsweise verändert sich im Laufe der Zeit das Artikelsortiment oder ein Artikel wechselt die Zugehörigkeit zur Artikelgruppe. Eine typische OLAP-Anforderung wäre jetzt der Vergleich einer aktuellen Gruppierung mit einer früheren [AnMu97]. Erste Ansätze zur temporalen Datenhaltung und Versionierung in DWHs sind in [ChSt98a, Kimb96] vorgenommen und in der Arbeit von [Herd99] erweitert worden. Dabei wird u.a. durch Einführung eines Gültigkeitszeitraums und eines künstlichen (surrogate) oder zusammengesetzten Schlüssels versucht, den temporalen Anforderungen zu genügen.

Aufbauend auf einem Datenwürfel lassen sich neben den aus dem herkömmlichen Berichtswesen bekannten Auswertungen auch **OLAP-Analysen** durchführen. Die Daten-Analyse mittels eines OLAP-Tools ist ein dynamischer, interaktiver, explorativer Prozess [Arb95, CCS93, Cla98, Fin95, Tho97], bei dem der Nutzer zur Navigation durch die multidimensionale Datenstruktur eine beliebige Kombination der nachfolgenden orthogonalen, multidimensionalen Operatoren anwenden kann. **Abbildung 2.11** stellt exemplarisch am Beispiel von Studenten- und Semester-Daten verschiedene Analyse-Möglichkeiten vor. Die Hauptoperationen sind:

- **Selektion**
  Die Selektion erfüllt eine Filterfunktionalität und dient der Auswahl einzelner Würfeldaten, z.B. finde die drei Studiengänge mit den höchsten Studierendenzahlen im Grundstudium während des Sommersemesters 1998 (BWL, VWL und WI).
- **Slice /Dice**
  Die beiden Operatoren stellen Spezialfälle der Selektion zur Erstellung individueller Ausschnitte auf den multidimensionalen Daten dar. Dem Anwender wird so eine individuelle und flexible Sichtweise auf die Daten geboten je nach seinen speziellen Anforderungen. Der *Slice*-Operator dient dem Herausschneiden von „Scheiben" aus dem Würfel. Der Cube wird dabei durch Beschränkung auf einen bestimmten

(Dimensions-)Wert um eine Dimension reduziert. Die *Dice*-Operation erlaubt das Herausschneiden einen „Teilwürfels“ unter Erhaltung der Dimensionalität. Die Ansicht beschränkt sich hierbei auf eine Teilmenge der Daten, bei der alle Dimensionen erhalten bleiben.

- **Rotation** (Pivotierung)
  Das Drehen des Würfels durch Vertauschen der Dimensionen erlaubt das Refokussieren des Blicks des Betrachters und damit die Analyse der Daten aus verschiedenen Perspektiven. Typischerweise folgt darauf eine Slice-Operation.
- **Drill Down / Roll Up**
  Die Operationen Roll Up und Drill Down erlauben das Durchlaufen von Konsolidierungspfaden/Hierarchien und damit die Variation des Verdichtungsgrades von Daten. *Drill Down* beschreibt die abwärtsgerichtete Navigation ausgehend von einem Aggregationsniveau auf die jeweils nächst tiefere und detailliertere Verdichtungsstufe. Die *Roll-Up*-Operation stellt die Komplementäroperation zum Drill Down dar und erzeugt durch den Wechsel zur jeweils höheren Verdichtungsebene innerhalb einer Dimensionshierarchie neue aggregierte Informationen.
- **Drill Across**
  Drill Across bewirkt den Wechsel von einem Würfel zu einem anderen.
- **Nest**
  Der Nest-Operator gestattet die komprimierte Darstellung des Würfels in Form einer zweidimensionalen Matrix (Kreuztabelle), bei der verschiedene Hierarchiestufen einer oder mehrerer Dimensionen auf einer Achse (Spalte oder Zeile) geschachtelt präsentiert werden.
- **Join**
  In Analogie zur relationalen Verbund-Operation versucht der OLAP-Join eine Verbindung mehrerer Hypercubes herzustellen. Die gemeinsamen Dimensionen zwischen den zu verknüpfenden Datenwürfeln gelten jedoch als Voraussetzung.

Der mit der SQL-Norm von 1999 unternommene Ansatz zur Erweiterung um spezielle OLAP-Funktionen (wie beispielsweise ROLLUP, CUBE, WINDOW, RANK sowie erweiterte Aggregatfunktionen [Mog05, Wie05]) erlaubt komplexere OLAP-Anfragen in SQL effizienter ausführen zu können und damit eine in SQL92 nur schwer/umständlich formulierbare Vielzahl statistischer Auswertungen auf dem Data-Warehouse-Datenbestand überhaupt erst möglich zu machen. Einige Beispiele sind zeitlich kumulierte Verkäufe für alle Monate des Jahres oder laufende Durchschnitte, Ranglisten der Produkte basierend auf den Verkaufszahlen der jeweiligen Produktgruppe oder die prozentuale Umsatzveränderung von diesem Monat zum Vorjahresmonat.

Die Konstrukte aus dem multidimensionalen Datenmodell werden in einer relationalen Datenbank-Lösung typischerweise durch das Star- oder durch das Snowflake-Schema umgesetzt. Ralph Kimball hat in [Kim96] den Terminus **Star Schema** als Bezeichnung für den Prozess der Denormalisierung bekannt gemacht, der die Struktur einer multidimensionalen Datenbank simuliert. Er beschreibt die Denormalisierung als Vorverknüpfung von Tabellen, die folglich zur Laufzeit nicht mehr verknüpft werden brauchen. Im Zentrum des Star Schema steht die Faktentabelle, die sternförmig von einer Reihe sog. Dimensionstabellen umgeben wird (siehe **Abbildung 2.12**). Sie enthält neben den Kenngrößen (Maßzahlen, Measures), Fremdschlüssel zu den Primärschlüsseln der

jeweiligen Dimensionen. Die Gesamtheit der Fremdschlüssel bildet den zusammengesetzten Primärschlüssel für die Faktentabelle. Im Sinne der Denormalisierung und einer schnellen Anfrageverarbeitung gibt es für jede Dimension genau eine Dimensionstabelle. Die denormalisierten Dimensionstabellen führen jedoch zu Redundanzen und der damit verbundenen Gefahr von Änderungsanomalien. Letztere sind jedoch im vornehmlich lesenden Data-Warehouse-Betrieb beinahe auszuschließen. Änderungen in einer Klassifikation treten sehr selten auf und werden meist unter kontrollierten Bedingungen mit Werkzeugen vorgenommen. Vorteile des Star Schema sind hingegen die einfache Struktur und der schnelle Datenzugriff.

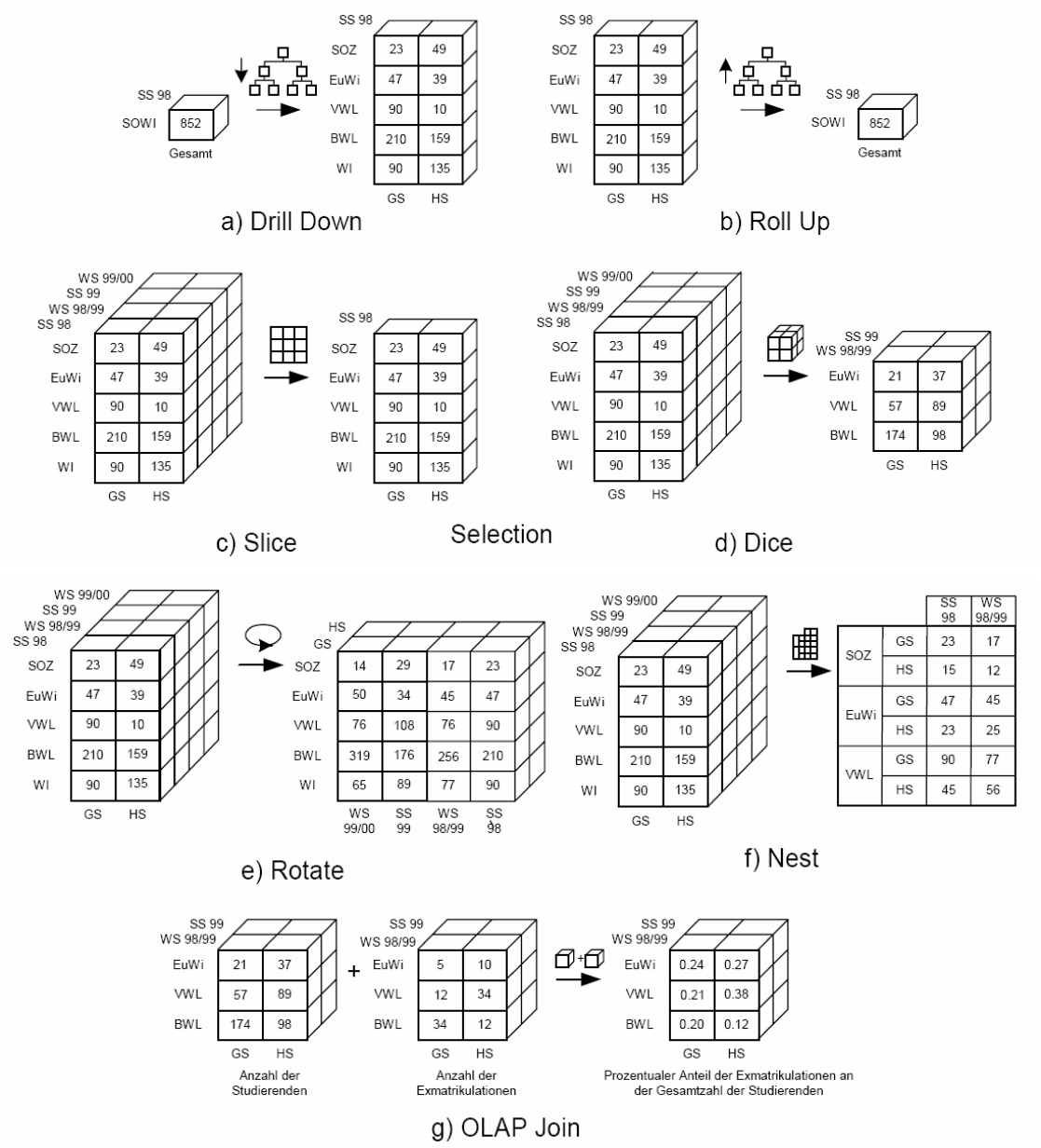


Abbildung 2.11: Typische OLAP-Operatoren [BoEn00]

Die Denormalisierung der Dimensionstabellen führt zu redundanter Datenspeicherung. Außerdem repräsentiert das Star Schema nicht explizit die Attribute-Hierarchie der Dimensionen. Im **Snowflake Schema** wird diese Schwäche überwunden, indem die Dimensionstabellen normalisiert werden. Bei dem Snowflake Schema wird, wie in **Abbildung 2.13** ersichtlich, für jede Klassifikationsstufe eine eigene Tabelle mit der ID für

den Klassifikationsknoten, den beschreibenden Attributen und Fremdschlüsseln der direkt übergeordneten Klassifikationsstufen angelegt. Die Kenngrößen eines Datenwürfels werden wie beim Star Schema in einer normalisierten Faktentabelle verwaltet. Sie enthält neben den Spalten für die Kenngrößen auch Fremdschlüssel zu den jeweils niedrigsten Klassifikationsstufen aller Dimensionen. Analog wird der Primärschlüssel in der Faktentabelle durch Zusammenschluss aller Fremdschlüssel gebildet. Der Vorteil des Snowflake Schemas liegt in der Vermeidung von Änderungsanomalien und Redundanzen. Die vielen resultierenden kleinen Tabellen müssen jedoch bei Anfragen jeweils miteinander verbunden werden. Da solche Verbundoperationen teuer sind, wird oft in ROLAP-basierten Data-Warehouse-Systemen das Star-Schema-Entwurfsmuster verwendet.

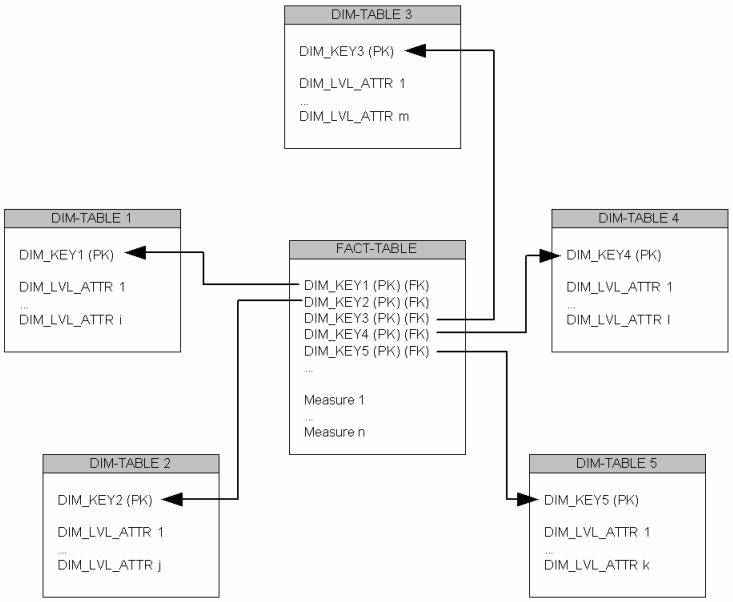


Abbildung 2.12: Struktur und Beispiel eines Star Schema

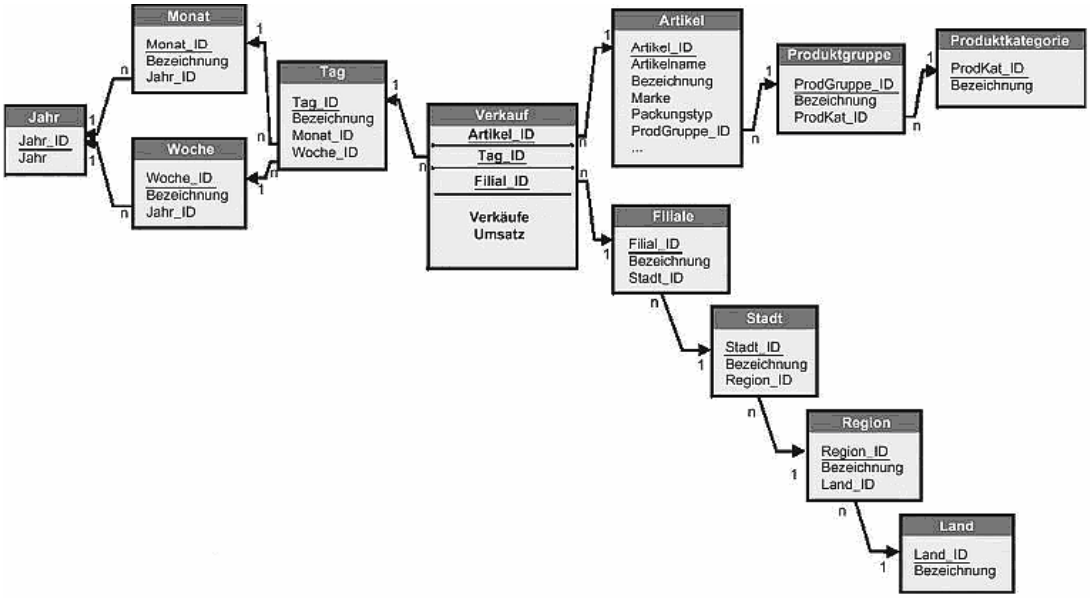


Abbildung 2.13: Struktur eines Snowflake Schema [BaGu01]

Abschließend lässt sich sagen, dass die Entscheidung über eines der beiden Schemata stark von den konkreten Daten- und Anfragecharakteristiken abhängt und sich deshalb keine allgemeine Aussage darüber machen lässt, welches der beiden Schemata besser geeignet ist. Beispielsweise ist es auch denkbar, wenngleich wenig ökonomisch, alle Informationen über die Dimensionen zusammen mit den Kennzahlen komplett denormalisiert in einer einzigen Tabelle unterzubringen. In der Praxis werden oft Mischformen bzw. leichte proprietäre Abwandlungen, wie das Galaxy Schema, Fact Constellation Schema, MicroStrategySchema, Informix Schema oder auch das Developed Star Schema verwendet [SCH04a, Leh03].

2.4 Data Mining

„We are drowning in information, but starving for knowledge." (John Naisbett)

Die rasch in ihrer Größe anwachsenden, für Menschen unüberschaubaren Datenbanken, insbesondere im Data Warehousing, enthalten oft verborgene bzw. auf den ersten Blick nicht oder nur schwer erkennbare Informationen und wecken somit das Bedürfnis nach automatischen Auswerte-Mechanismen.

Ziel dieses Abschnitts ist es, dem Leser eine kurze Einführung in das Thema Data Mining zu geben. Hierzu werden in Anlehnung an [AlNi00, Ban04, ChBu97, FPSS96a, FPSS96b, FPSSU96, Pet05] gängige Definitionen und die Ziele des Data Mining sowie die vier wichtigsten Techniken Klassifikation, Clustering, Regression und Assoziation kurz vorgestellt. Ziel ist es ein grundlegendes Verständnis für die in Kapitel 7 beschriebenen Anwendungs-Szenarien der Performance-Daten zu vermitteln.

2.4.1 Definition und Ziele des Data Mining

Der Prozess zur Vorverarbeitung der Ausgangsdaten, Erkennung und automatischen oder semi-automatischen Extraktion sowie die adäquate Präsentation von Wissen in Datenbanken [FPSS96, Kri00] wird als „Knowledge Discovery in Databases" (KDD) bezeichnet und ist in **Abbildung 2.14** dargestellt. Das Data Mining steht im Mittelpunkt dieses Verfahrens.

Da die Wissenschaft rund um das Data Mining noch relativ jung ist und sich Fachleute noch nicht auf eine einheitliche Definition des Begriffes geeinigt haben, wurde Data Mining zum Schlagwort in nahezu jeder Entwicklung, die Manipulation oder Analyse von Daten betreibt. Der Begriff wird heutzutage häufig überstrapaziert und missverstanden, in der Literatur findet man eine Vielzahl von Definitionen.

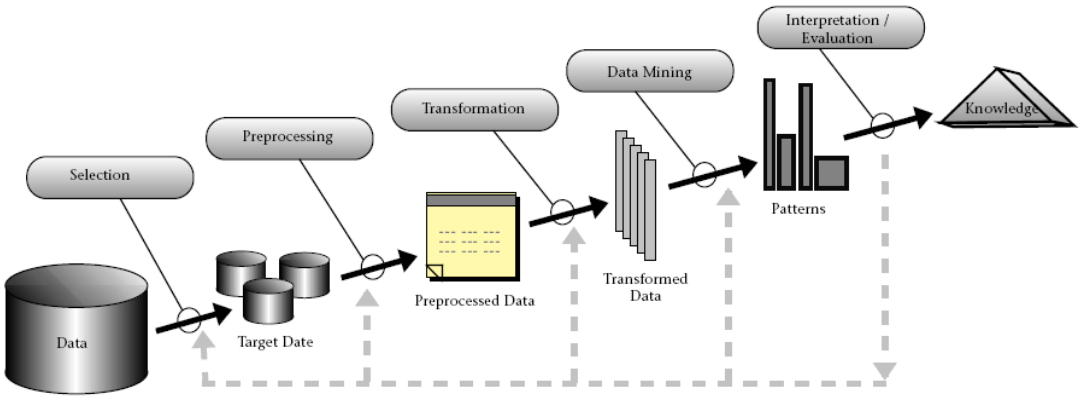


Abbildung 2.14: Stufen des KDD-Prozesses [FPSS96b]

Die Gartner Group[3] bspw. definiert Data Mining als „*Prozess des Erkennens bedeutungsvoller neuer Korrelationen, Muster und Verläufe durch Sieben großer, in Speichern abgelegter, Daten-Mengen unter Benutzung von Technologien der Muster-Erkennung sowie statistischer und mathematischer Techniken*". Diese akzeptable Definition beinhaltet sowohl eine Beschreibung der Resultate des Data Mining, als auch die Information, dass verschiedene Fachgebiete zusammenarbeiten müssen, um diesen Prozess erfolgreich zu bestreiten. Ob der Erkenntnisgewinn aus der Analyse der zuvor aufbereiteten Daten dabei manuell oder automatisch erfolgt, sei dahingestellt. Im Gegensatz zu gerichteten Analysen erlauben Data-Mining-Verfahren bisher unentdeckte Zusammenhänge aus den Daten zu extrahieren und aufzudecken, ohne die Formulierung einer exakten Fragestellung.

In der Literatur findet man unterschiedlichste Synonyme, die mit diesem Forschungsgebiet gleichgesetzt werden können: *Knowledge Mining from Databases, Knowledge Extraction, Data Archaeology, Data Dredging, Data Analysis* etc. Doch in der Regel wird der Ausdruck Data Mining bevorzugt.

Der Data-Mining-Prozess ist weiterhin meist durch einen effektiven Einsatz von visuellen Analysen gekennzeichnet. So sind neben den anspruchsvollen statistischen Techniken vor allem interaktive, leistungsstarke und visuelle Schnittstellen ein Markenzeichen moderner Data-Mining-Werkzeuge, wie SPSS Clementine[4], SAS Enterprise Miner[5], IBM DB2 Intelligent Miner[6], Rapid Miner[7] oder Weka[8]. Durch sie werden Daten angezeigt, temporär angepasst und auf verschiedene Art analysiert.

2.4.2 Methoden und Techniken des Data Mining

Das sehr breit gefächerte Gebiet des Data Mining bietet ein umfangreiches Spektrum an originären und weiterentwickelten Methoden und Techniken zur Analyse umfangreicher Datenbestände. Laut [HKMW01] stammen die Wurzeln der Methoden des Data Mining aus mindestens sechs verschiedenen Disziplinen: Traditionelle Statistik und Datenanalyse, Künstliche Intelligenz, Muster-Erkennung, Datenbanktheorie und -praxis, Computer-linguistik und Information Retrieval sowie Computergraphik. Gute Einführungen zu den dabei verwendeten Techniken aus Sicht der einzelnen Gebiete sind in [BeHa03, CHY96, Mit97] auffindbar.

Die vorgestellten und in gängigen Tools eingesetzten Techniken und Algorithmen sind meist Teil eines größeren Prozesses und interagieren mit anderen Produkten. Eingabedaten liegen somit in unterschiedlichen Formen vor und Ausgaben werden in verschiedenen Formen benötigt. Die Hersteller einigten sich daher auf eine Menge von Standards, mit dem Ziel den Ablauf des Data Mining zu regeln, den Austausch von Data-Mining-Modellen zwischen verschiedenen Produkten zu vereinfachen und Data Mining in Anwendungen zu integrieren Zu den wichtigsten Standards/Schnittstellen zählen der CRoss-Industry Standard Process for Data Mining (CRISP) [CCK+00], die Predictive Model Markup Language (PMML) [RAS04] sowie die SQL Multimedia and Application Packages (SQL/MM) [MeEi01].

---

[3] http://www.gartner.com
[4] http://www.spss.com/de/clementine/
[5] http://www.sas.com/technologies/analytics/datamining/miner/
[6] http://www-306.ibm.com/software/data/infosphere/warehouse/mining.html
[7] http://rapid-i.com/content/view/64/66/lang,de/
[8] http://www.cs.waikato.ac.nz/ml/index.html

Die Analyse im Data Mining wird durch die in **Abbildung 2.15** anhand ihrer Zielsetzung gruppierten Methoden sowie ihnen zugewiesene, stellvertretende Techniken unterstützt. Dabei haben wir uns auf eine geringe Menge der entstandenen großen Anzahl von Data-Mining-Techniken auf Grundlage von [Alpar00, KZ02, IBM06d, ZHA99] beschränkt. Details zu den nachfolgend beschriebenen Verfahrensweisen können in den Grundlagenpapieren und teilweise auch in unseren Arbeiten [Böt07, Göb08, Kap08] studiert werden.

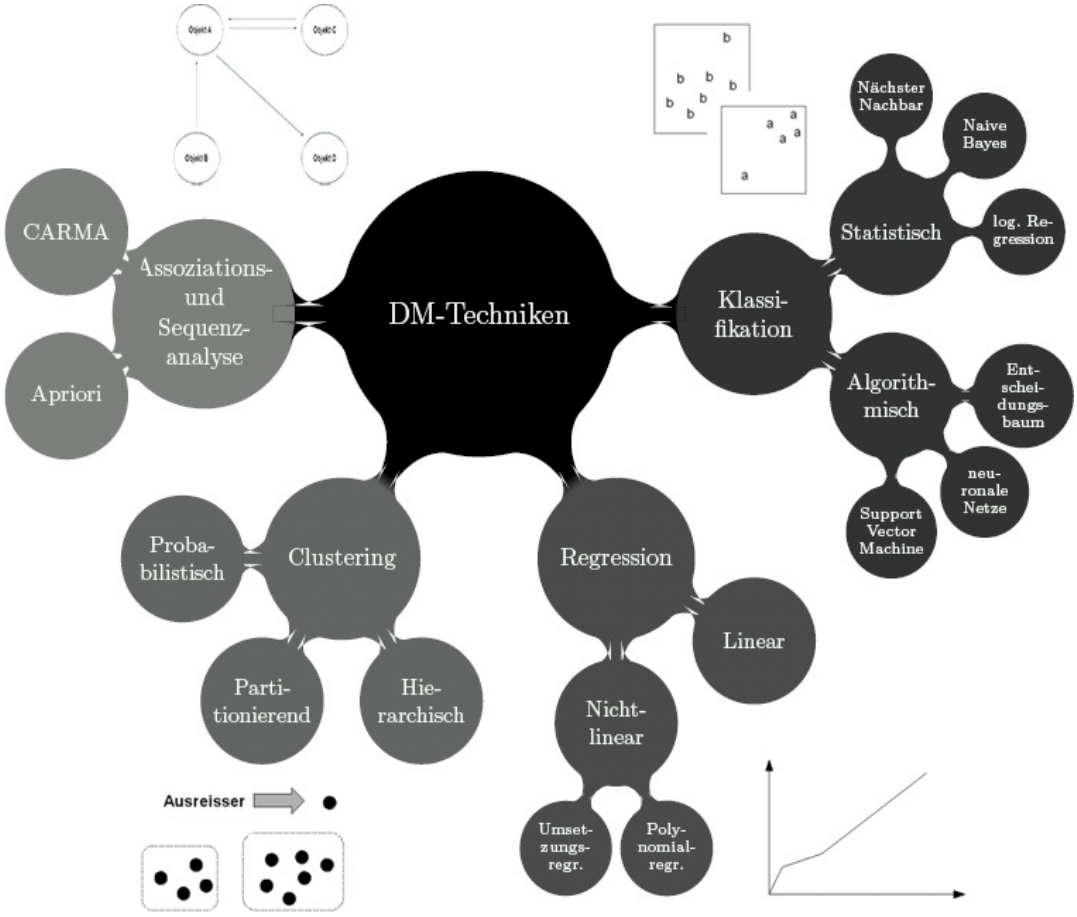


Abbildung 2.15: Übersicht wichtiger Data-Mining-Techniken (nach [Göb08])

Klassifikation

Die Zielsetzung der Klassifikation besteht darin, den zugrunde liegenden Datenbestand vorgegebenen Klassen zuzuordnen. In der initialen Lernphase der Klassifikation wird aus einer Reihe von Datensätzen, deren Klassenzugehörigkeit bekannt ist, automatisch ein Modell erstellt. Das Klassifikationsverfahren analysiert diese so genannten Trainingsdatensätze (Stichproben) und erstellt ein Profil für die Merkmale/Attribute von Datensätzen aller vorhandenen Klassen. Anschließend erlaubt das Klassifikations-Modell, die Klasse unklassifizierter Datensätze vorherzusagen. Beispielsweise verfolgen Versicherungsgesellschaften bei der Klassifikation von Kunden hinsichtlich ihres Schadensrisikos ein solches Ziel.

Um das Klassifikationsproblem zu lösen und ein Modell zu erstellen, gibt es im Wesentlichen die in Abbildung 2.15 ersichtlichen statistischen oder algorithmischen Verfahren. Zu den wichtigsten Techniken zählen die Entscheidungsbaumklassifikation, Naive-Bayes-Klassifikation und Klassifikation durch logistische Regression [Göb08].

Ein *Entscheidungsbaum* bspw. ist ein Graph mit Baumstruktur, bei dem die Wurzel und jeder innere Knoten ein Merkmal eines zu klassifizierenden Objekts auf die Erfüllung einer Bedingung testet. Die von den Knoten ausgehenden Verzweigungen entsprechen den Werten oder Wertebereichen, die diese Bedingung überprüft und beschreiben die für die Klassifizierung bzw. Segmentierung relevanten Ausprägungen der Merkmale. Den Blättern des Baums sind die Klassen zugeordnet. Eine Klasse bzw. ein Merkmal kann dabei mehreren Blättern bzw. Entscheidungsknoten zugewiesen werden. Um ein noch nicht klassifiziertes Objekt einzugruppieren, werden die Merkmalswerte dieses Objektes gegen die inneren Knoten des Entscheidungsbaums getestet. Somit wird ein Pfad über alle Stufen des Baumes durchlaufen, der in einem Blatt endet und so das analysierte Objekt eingruppiert.

Jeder der Klassifikationsalgorithmen erzeugt immer ein Modell der Daten, den sog. Klassifikator. Ein solches Modell kann auf zwei Arten genutzt werden. Zum einen für die Vorhersage bzw. Bestimmung der Klasse bzw. Kategorie eines bisher unbekannten Objektes einzig aufgrund seiner Attributwerte und der anderen Attribute. So wäre es z.B. möglich, aufgrund des Alters, des Geschlechts und der Tätigkeit eines Kunden seine (wahrscheinliche) Einkommensgruppe vorherzusagen. Zum anderen stellt das Modell eine Abstraktion der Eingabedaten dar und dient dem Erkenntnisgewinn bzw. der Generierung von Wissen über die Struktur der Eingabedaten und über die Klassen.

Bei der Nutzung eines solchen Modells sollte jedoch besonders beachtet werden, dass der Klassifikator im Grunde den naiven Versuch darstellt, aus den empirisch gewonnenen Eigenschaften einer kleinen Teilmenge auf die der Gesamtmenge zu schließen. Er ist lediglich für die Trainingsdaten optimiert, liefert also für die Gesamtheit der Daten eventuell unbrauchbare Ergebnisse, da die Kategorisierung nicht zu 100 Prozent richtig sein muss. Sie spiegelt lediglich die wahrscheinlichste Kategorie bei den gegebenen Attributen wider. Daher wird das Modell im Laufe der Zeit angepasst oder neu erstellt werden müssen. Zudem wird in der Regel die Genauigkeit des Klassifikations-Modells durch Klassifikation größerer Datensätze erhöht. Zur Beurteilung der Qualität eines gelernten Klassifikators hat sich die „Train and Test"-Methode etabliert, die die Testobjekte in eine Trainingsmenge zum Lernen des Klassifikators und in eine geringere Testmenge zum Schätzen des Klassifikationsfehlers unterteilt.

Clusterbildung / Entdeckung von Ausreißern

Im Unterschied zur Klassifikation liegt bei der Clusterbildung das Klassifikationsschema und die Klassenzugehörigkeit eines Objektes nicht fest, sondern wird durch das Verfahren erst festgelegt bzw. definiert. Die Aufgabenstellung der Clusterbildung besteht damit in der explorativen Partitionierung der Daten in unterschiedliche Klassen (Cluster, Gruppen). Die Data-Mining-Methode durchsucht die Eingabedaten nach Merkmalen, die häufig zusammen auftreten und segmentiert die Daten anhand dieser Merkmale. Dabei sollen Daten, die demselben Cluster zugeordnet werden, sich möglichst ähnlich (homogen), Objekte unterschiedlicher Gruppen jedoch möglichst unähnlich sein und sich stark voneinander unterscheiden. Ähnlichkeiten bzw. Verschiedenheiten werden durch Ähnlichkeits- bzw. Verschiedenheitsmaße ausgedrückt und aus den Abständen (Distanzen) zwischen den Datensätzen und zwischen den Gruppen von Datensätzen ermittelt [Küs01, MeWi00]. Dabei lassen kleine Distanzen auf ähnliche Objekte und große Distanzen auf unähnliche Gebilde schlussfolgern.

Die Clustering-Verfahren können, wie in Abbildung 2.15 auf Grundlage von [GrRu02] dargestellt, in partitionierende, hierarchische oder probabilistische Cluster-Verfahren unterteilt werden.

Das vom Clustering-Verfahren erstellte Modell enthält alle erkannten Cluster, deren Merkmale und die Verteilung der Merkmale innerhalb der Cluster. Das Clustering dient in vielen Fällen als Teilziel lediglich der Vorverarbeitung, um Daten-Mengen handhabbar oder homogener zu halten und somit die Analyse zu vereinfachen. Eine typische Anwendung ist z.B. die Segmentierung von Kunden im Marketing.

Die entstehende Segmentierung kann auch Grundlage für eine weiterführende Abweichungs-Entdeckung sein. Unter Ausreißern versteht man in dem Sinne einzelne Objekte, die zu keinem der gefundenen Cluster gehören bzw. mit vergleichsweise wenigen anderen Objekten in einem Cluster liegen. Eine derartige Untersuchung der zugrundeliegenden Daten nach Ausprägungen von Merkmalen, die sich besonders stark von den übrigen Ausprägungen dieser Merkmale unterscheiden, findet beispielsweise Anwendung bei der Kennzahlenanalyse im Controlling.

Die gängigen Algorithmen werden maßgeblich durch die Wahl geeigneter Eingabeparameter beeinflusst. Eine gute initiale Zerlegung der Datenmenge kann die Qualität des Endergebnisses des Algorithmus und die Anzahl an Iterationen bis zur Terminierung begünstigend beeinflussen. Eine geeignete Wahl der Anzahl der zu findenden Cluster spielt ebenso für die Qualität der zu findenden Ergebnisse eine entscheidende Rolle. Eine suboptimal festgelegte Clusteranzahl kann dazu führen, dass die Resultate stark von der Realität abweichen und nicht weiterverwendet werden können. Die Anwendung des Verfahrens setzt zudem die Auswahl eines im Hinblick auf die Merkmale der Datensätze geeigneten Abstandsmaßes voraus. Ein weiteres häufiges Problem beim Einsatz von Clusterverfahren ist letztlich die Interpretation der ermittelten Cluster.

Regression

Ziel des statistischen Verfahrens der Regressionsanalyse ist die Ermittlung (Schätzung) bzw. Erklärung von Ursache-Wirkungs-Zusammenhängen in Form funktionaler Beziehungen zwischen mehreren Merkmalen der Datenbasis. Dies kann sehr hilfreich sein, um auf den Ergebnissen basierend geeignete Folge-Analysen durchzuführen, aber auch um entsprechende Prognosen über bestimmte Größen zu treffen. Die typischen Beispiele der Regressionsmodellierung im Data-Mining-Umfeld stimmen mit den Anwendungsfeldern in der Statistik weitgehend überein. Klassische Anwendungsbeispiele für die Regressionsanalyse findet man insbesondere im Kreditwesen zur Erklärung von Entwicklungen von Aktienkursen oder beispielsweise im Bereich der Wirtschaftswissenschaften zur Ermittlung einer Preis-Absatz-Funktion, mit deren Hilfe der optimale Verkaufspreis berechnet werden kann.

Regressionsmodelle werden im Data Mining oft nicht nur für analytische, sondern auch für prognostische Zwecke benutzt. Dabei verwendet man vor allem Zeitreihen zur Prognose der zukünftigen Entwicklung. Ein Beispiel ist etwa die Prognose der Absätze eines Artikels auf der Grundlage der bisherigen Entwicklung.

Die Regression ist der Klassifikation sehr ähnlich. Bei der Klassifikation wird eine Klassenbezeichnung, bei der Regression hingegen ein numerischer Wert des Zielattributs anhand bekannter Erläuterungsattribute vorhergesagt. Die Regression ermöglicht für die Vorhersage der Zielfeldwerte eine Ermittlung der Eingabefelder mit der höchsten Relevanz. Die Verfahren gliedern sich anhand der durchgeführten Transformation, wie in Abbildung 2.15 ersichtlich, in lineare und nichtlineare Regressionstechniken.

Wird zwischen den „erklärenden Variablen und der „abhängigen Variable“ ein linearer Zusammenhang vermutet, so kann eine (multiple) *lineare Regression* durchgeführt werden. Die so generierten Modelle stellen Gleichungen dar, die das Zielfeld approximieren.

Da die initiale Wahl der Variablen einen erheblichen Einfluss auf die Qualität des Modells hat, sind zur Auswahl der Erläuterungsattribute spezifische Kenntnisse nötig. Mittels einer autonomen Variablenwahl können Mining-Tools oft selbstständig entscheiden, welche Erläuterungsparameter relevant sind.

Oft liegt für eine gegebene Datenmenge anstelle einer linearen Abhängigkeit eine nichtlineare Beziehung vor und es ist eine nichtlineare Regressionsfunktion zu bestimmen. Die Technik der *Polynomial-Regression* beispielsweise nimmt zwischen den Erläuterungsattributen und dem Zielattribut die Form einer Polynomial-Beziehung an.

Eine Beschränkung auf lineare Zusammenhänge ist in der Praxis aufgrund guter Ergebnisse durchaus vertretbar. Darüber hinaus sind lineare Funktionen wesentlich leichter zu handhaben und zu interpretieren als nichtlineare Abbildungen. Außerdem gibt es Möglichkeiten, mit denen nichtlineare Funktionen linearisiert werden können.

Assoziations- und Sequenzanalyse

Das Ziel der Assoziationsanalyse ist die Bestimmung von aussagekräftigen (logischen) Beziehungs-Zusammenhängen zwischen unterschiedlichen Attributen bzw. Ausprägungen von Merkmalen des zugrunde liegenden Datenbestandes. Diese daraus folgenden *Assoziationsregeln* beschreiben Korrelationen zwischen (häufig) gemeinsam auftretenden Dingen und können in der Form „*IF Rumpfbedingung THEN Kopf*“ (Rumpf → Kopf bzw. „Wenn Voraussetzung, dann Konsequenz“ bzw. „ Wenn Bedingungen, Dann Ergebnis“) ausgedrückt werden. Einführende Literatur zu diesem Thema findet man in [Bol96]. Regeln in dieser Form sind aussagekräftig, wenn ihr *Support* (Häufigkeit des gemeinsamen Auftretens der Rumpfbedingung und des Kopfes im gesamten Datensatz) und *Confidence* (Häufigkeit des Auftretens vom Kopf, unter der Bedingung, dass die Rumpfbedingung erfüllt ist) einen benutzerdefinierten Wert überschreiten.

Ein typisches Beispiel für die Assoziationsanalyse ist die sogenannte *Warenkorbanalyse* im Handel [Lar05]. Bei einer gegebenen Menge von Einkäufen soll hierbei untersucht werden, ob es bestimmte Zusammenhänge zwischen den einzelnen Einkäufen gibt. So soll basierend auf einem bestimmten gekauften Artikel oder eine Menge von Artikeln ermittelt werden, ob mit einer gewissen Häufigkeit auch andere Artikel (durch denselben Kunden) gekauft wurden. Es gibt diverse Gründe, um solche konditionalen Regeln zu suchen. Der Lagerbestand kann optimiert werden, wenn man voraussehen kann, was sich mehr verkaufen lassen wird. Das Finden von Assoziationsregeln zwischen Produkten kann auch zur Reorganisation der Verkaufsfläche führen.

Assoziationen können Zusammenhänge zwischen Werten verschiedener Attribute (horizontal) und zwischen Werten eines Attributes (vertikal) herstellen. **Abbildung 2.16** veranschaulicht diese Unterscheidung anhand einer einfachen Warenkorb-Analyse.

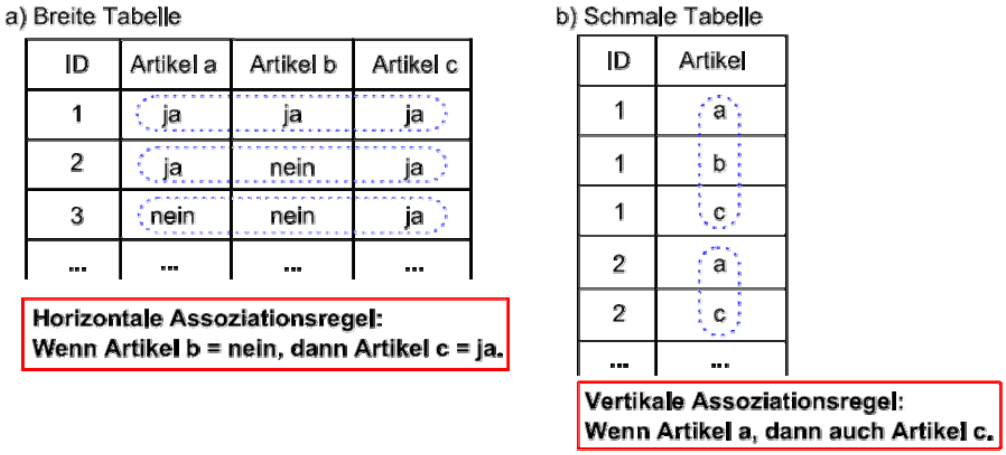


Abbildung 2.16: Horizontale vs. vertikale Assoziationen [Kap08]

Es ist zu beachten, dass gefundene Regeln nichts über einen kausalen Zusammenhang aussagen. Sie stellen lediglich empirisch gefundene Abhängigkeiten dar. Man stelle sich beispielsweise vor, dass in einem kleinen Kaufhaus innerhalb einer bestimmten Zeitspanne von rund 200 Leuten Windeln gekauft wurden. Etwa 50 Leute kauften neben Windeln auch Bier. Insgesamt seien in der Zeitspanne 1000 Personen im Kaufhaus einkaufen gewesen. Aufgrund dieser Zahlen könnte nun folgende Regel aufgestellt werden: „Wenn jemand Windeln kauft, dann kauft er auch Bier“. Diese Regel hat einen Support-Wert von 200/1000 = 20% und ihr Confidence-Wert liegt bei 50/200 = 25%. Beide Werte sind relativ hoch und sprechen für eine gute Regel. Insbesondere bei sehr vielen Variablen können u.U. sinnfreie Korrelationen und Schmutzeffekte aufgrund transitiver Abhängigkeiten entstehen.

Eine *Sequenzregel* ist ähnlich aufgebaut wie eine Assoziationsregel und betrachtet zusätzlich den Faktor Zeit. Sie besteht aus einer vorhergehenden Sequenz im Regelrumpf, die zu einer nachfolgenden Elementmenge im Regelkopf führt, die nach Ablauf eines bestimmten Zeitraums auftritt.

Fazit

Bei der Auswahl des geeigneten Verfahrens sind die mit dem Data Mining verbundene Zielsetzung, die Eigenschaften (Struktur und Qualität) der zu analysierenden Daten und die Darstellungsform der zu ermittelnden Beziehungsmuster zu berücksichtigen [BaGu01]. Insbesondere die Darstellungsform (z.B. Entscheidungsregeln, Funktionen) hat Auswirkungen auf die Möglichkeit der Erklärungsfähigkeit und Interpretation der abgebildeten Zusammenhänge [FPSS96a]. Wir werden schließlich in den Abschnitten 7.3, 7.4 und 7.5 einige Anwendungen von Data-Mining-Techniken im Umfeld des autonomen Performance-Tuning vorstellen und auf die dargelegten Grundlagen zurückgreifen.

Kapitel 3

Autonomes Tuning von Datenbanksystemen

Das vorangegangene Kapitel hat sich intensiv den Grundlagen des Tuning und wesentlichen Aspekten des Autonomic Computing gewidmet. Im Folgenden stehen die Verschmelzung der beiden Gebiete und die Übertragung der Problemstellung auf die (Selbst-)Optimierung, sprich die Überwachung und Steuerung des Leistungsverhaltens von Datenbanken, im Mittelpunkt der Betrachtungen. Das Ziel ist hierbei die Unterstützung und Entlastung der Datenbank-Administratoren bei der durch die Vielzahl interner Parameter komplizierten Systemverwaltung und bei Routine-(Tuning-)Tätigkeiten, wie der Datensammlung, der Problem-Erkennung und -Diagnose sowie der letzlichen Problem-Auflösung.

Der derzeitige Stand der Entwicklungen im Bereich des autonomen Datenbank-Tuning zeigt sich in sehr differenzierter Form. In diesem Kapitel werden daher zunächst in **Abschnitt 3.1** ausgewählte autonome Funktionalitäten, Komponenten und Tools von IBM DB2, Oracle und Microsoft SQL Server als bedeutende heutige Vertreter der relationalen Datenbankwelt gegenübergestellt und auf ihre Tauglichkeit in Bezug auf die Automatisierung von Optimierungs- und Administrationsaufgaben untersucht. Im Anschluss werden in **Abschnitt 3.2** einige grundlegende Herangehensweisen unterschieden, die eine Selbst-Verwaltung und insbesondere die Selbst-Optimierung von und in Datenbanksystemen ermöglichen. Danach werden die seit einigen Jahren sowohl in Form vereinzelter Konzepte als auch in Form von Prototypen aus der Forschung konkret existierenden Ansätze kurz vorgestellt und den Herangehensweisen zugeordnet. Das Kapitel schließt im **Abschnitt 3.3** mit einem kurzen Fazit und der Motivation unserer, im Rahmen der vorliegenden Arbeit angestrebten Vorgehensweise zum autonomen Tunen von Datenbanksystemen.

3.1 Autonomie in heutigen Datenbanksystemen

Führende Hersteller von Datenbank-Management-Systemen (DBMS) haben die Notwendigkeit erkannt, dem System selbst die Administrations- und Optimierungsaufgaben zu übertragen und arbeiten seit einiger Zeit unter dem Schlagwort der „autonomen Datenbanksysteme“ daran, Funktionen, die bislang von Spezialisten übernommen wurden, als Automatismen in die Systeme zu integrieren [EPBM03].

Das Spektrum der Aufgaben reicht dabei von reaktiven, in Echtzeit zu erfolgenden bis hin zu proaktiven, langfristigen Maßnahmen, die sich in die folgenden (nicht disjunkten) Kategorien gliedern:

- **(Initiale) Konfiguration**

  Initiale bzw. seltene/unregelmäßige Tätigkeiten zum System-Setup, zur Initialisierung und zur Konfiguration. Darunter fallen u.a. die Kapazitätsplanung, Installation oder Upgrade des DBMS, Konfiguration der wesentlichen Parameter, Definition der zu verwendenden Speichermedien, die Aufteilung der Daten auf diese und ggf. deren Formatierung sowie Daten-Migration.

- **Physischer Entwurf**

  Im Anschluss an Tätigkeiten der Schemadefinition im Rahmen des logischen Datenbankentwurfs befasst sich der Datenbank-Administrator üblicherweise mit der Optimierung des physischen Datenbank-Layouts. Zu dem physischen Datenbank-Entwurf zählen Tätigkeiten, wie die Strukturierung, Verteilung und Auslagerung der Daten auf den Speichermedien, das Einrichten und Löschen von Hilfsstrukturen wie Zugriffspfade (Indexe) oder materialisierte Sichten, das Partitionieren von Tabellen und Replizieren von Daten etc. Für das Anlegen, Verwalten (und Konfigurieren) dieser die Performance maßgeblich beeinflussenden Strukturen verwendet man in der Regel SQL-DDL-Anweisungen.

- **Systemverwaltung**

  - **Ressourcen-Management**

    Anlegen, Konfigurieren, Verwalten und Überwachen der benötigten Systemressourcen, insbesondere von Primär- und Externspeicher (Storage Management und Memory Management). Im Unterschied zu den Datenbank-Objekten des physischen Designs gibt es für diese Systemressourcen keine SQL-Konstrukte. Sie werden mittels DBMS-spezifischen (System-)Befehlen oder Konfigurationsparametern verwaltet.

  - **Pflege und Wartung (Maintenance)**

    Datenbank-Management-Systeme können nicht garantieren, dass die durch den logischen oder physischen Datenbankentwurf festgelegten Eigenschaften im Laufe dieses Betriebs erhalten bleiben. Vordefinierte Bereiche können überlaufen, Cluster-Eigenschaften verloren gehen, Daten fragmentieren, Statistiken veralten usw. Die Pflege und Wartung umfasst daher alle Maßnahmen zur Stabilisierung des Systemverhaltens und Aufrechterhaltung eines akzeptablen Grundzustands. Dazu zählen die Defragmentierung, die Reorganisation von (Index-)Daten, das Erstellen und Verwalten von Sicherungs-(Backup-) und Wiederherstellungs-(Recovery-) und Archivierungs-Verfahren, die Pflege und Aktualisierung der Statistiken sowie das Exportieren und Laden von Daten (Massendatentransfer).

- **Performance-Überwachung und -Tuning (Performance Management)**

  Der DBA ist verantwortlich für zufriedenstellende Antwortzeiten und einen ausreichenden Durchsatz des Datenbanksystems. Um diese und andere im Rahmen von Richtlinien definierten Ziele zu erreichen, bedient er sich der Beobachtung und Protokollierung des Systemverhaltens zur Identifikation von Anomalien, Bestimmung von Ursachen und zum Einleiten von lindernden bzw. behebenden Maßnahmen.

- **System- und Ausfallsicherheit**

  Hierunter fallen das Erstellen, Implementieren und Überwachen von Datenbank- und DBMS-Sicherheitskonzepten zur Authentifizierung, Autorisierung und Zugangskontrolle. Dies umfasst u.a. das Einrichten und Löschen von Benutzerkennungen und das Verwalten von Last-abhängigen Richtlinien bzw. Berechtigungen der Benutzer auf die Datenbank-Objekte sowie für die verschiedenen Programme, die auf die Datenbank zugreifen. Um in bestimmten Anwendungs-Szenarien nachvollziehen zu können, wer welche Änderungen verursacht hat, stehen dem DBA Auditing-Werkzeuge zur Verfügung. Kritischer werdende Applikationen und eine globaler(e) Nutzung verlangen in erheblichem Maß eine dauerhafte Verfügbarkeit und Ausfallsicherheit (Availability). Die „Disaster Recovery" hilft dem System durch den Aufbau und die Wartung einer Recovery-Umgebung sowie der Verwaltung von Archiven von Logs und Backups nach einem Desaster in einen stabilen Zustand zurückzukehren und sich zu erholen.

In den nachfolgenden Unterabschnitten werden DB2 UDB von IBM in der Version 9[9], Oracle 11g[10] und Microsoft SQL Server 2008[11] basierend auf allgemein zugänglichen Materialien, wie größtenteils den (Online-)Systemdokumentationen [IBM10, MS10, ORA10], verschiedenen Forschungspapieren, wie bspw. [EPBM03] sowie entsprechend genannter White Paper hinsichtlich ihrer autonomen Fähigkeiten in den obigen Kategorien verglichen, um abschließend beurteilen zu können, inwieweit sie sich dem autonomen Ideal nach heutigem Stand genähert haben und welche zu lösenden Probleme weiterhin bestehen.

Einen Datenbanksysteme-Vergleich zu erstellen ist im Allgemeinen keine leichte Aufgabe. Die Funktionalität und Komplexität der DBMS ist sehr hoch und der tatsächliche Wert einiger Aspekte kann nur durch diverse praktische Erfahrungen und Benchmarks beurteilt werden. Eine objektive Beurteilung wird jedoch deutlich erschwert durch große Unterschiede in den Implementierungsdetails und der Produktphilosophie. Daher handelt es sich im Folgenden mehr um eine beispielhafte, wertungsfreie Veranschaulichung, als um eine vollständige Gewinner-Verlierer-Liste.

3.1.1 (Initiale) Konfiguration

Neben initialen Tätigkeiten zur Installation und dem System-Setup steht vor allem die (Erst-) Konfiguration des DBMS und seiner Datenbanken im Mittelpunkt des performanten Betriebs. Initiale Installationskonfigurationen können naturgemäß weder für die Vielzahl an Nutzungsvarianten, noch unter allen denkbaren Umständen akzeptable Performance liefern. Klassischerweise muss die Konfiguration daher auf die zugrunde liegende Hardware und die zugreifenden Applikationen zugeschnitten sein.

Zur Konfiguration eines DBMS können weitgefasst Stellschrauben, wie System- und Umgebungsparameter, obere bzw. untere Schranken für den Ressourcenverbrauch und auch die (Nicht-)Existenz bzw. Parametrisierung von Hilfs-Strukturen, wie Indexe oder materialisierte Sichten, gezählt werden. Zur feineren Abgrenzung sollen hier jedoch lediglich die Konfigurationsparameter zur Steuerung des DBMS-Verhaltens betrachtet werden.

---

[9] http://www-01.ibm.com/software/data/db2/linux-unix-windows/
[10] http://www.oracle.com/technology/products/database/oracle11g/index.html
[11] http://www.microsoft.com/germany/sql/2008/default.mspx

In Anbetracht der Fülle von Konfigurationsparametern moderner, leistungsfähiger Datenbanksysteme und der zahlreichen Abhängigkeiten zwischen diesen, ist die optimale Systemkonfiguration eine Arbeit für erfahrene Spezialisten. Daher sind Konfigurations-unterstützende Werkzeuge eine große Erleichterung für jeden Administrator. Ein autonomes DBMS sollte seine Nutzer nicht nur mit einer zufriedenstellenden „out-of-the-box"-Konfiguration versorgen, sondern sich auch im Sinne der Selbst-Konfiguration dynamisch und möglichst ohne große Störungen des operativen Betriebs den geänderten Anforderungen bzw. Umgebungsbedingungen anpassen. Das Erfordernis von zur Laufzeit (online) (re)konfigurierbaren Parametern ist erst seit kurzem Thema bei den DBMS-Herstellern. Für einen Teil der Tuning-Parameter müssen beispielsweise noch zunächst alle (Applikations-)Verbindungen zur Datenbank getrennt oder gar das System neu gestartet werden, bevor ein Effekt eintritt.

Der **DB2 Configuration Advisor** macht Vorschläge, um rund 35 Konfigurations-parameter der DB2-Instanz und der aktuellen (verbundenen) Datenbank zu optimieren. Der DBA gibt dafür im Rahmen einer einfachen Fragenliste wesentliche System-Charakteristika, Speicherbegrenzungen und Parameter zum Lastprofil vor. Dabei kann er wählen, ob der Configuration Advisor ihm die Vorschläge anzeigen soll oder die Parameter-Änderungen gleich „selbst" durchführen darf. Bis zur Version 8 musste der DB2 Configuration Advisor manuell aufgerufen werden, um sich Vorschläge zur Verbesserung der Datenbank und des Datenbank-Manager errechnen zu lassen. In der aktuellen DB2-Version wird der Advisor automatisch (per AUTOCONFIGURE) gestartet und übernimmt einige grundlegende Empfehlungen. So werden zum Beispiel die Größe des Default-Bufferpools sowie die Anzahl der Page Cleaner und I/O-Server konfiguriert. Somit bleibt manch unbedarftem Nutzer eine unliebsame Überraschung erspart, sofern dieser nicht wusste, dass die Startkonfiguration noch mit oft zu kleinen Standardwerten aus den Anfängen von DB2 bestückt wird.

Der **Oracle Database Configuration Assistant (DBCA)** ist ein grafisches Tool, welches ermöglicht, eine neue Datenbank anzulegen, die Optionen einer existierenden Datenbank anzupassen, eine Datenbank zu löschen und Vorlagen zur Konfiguration von Datenbanken zu verwalten. Außerdem übernimmt dieses Tool das initiale Speichermanagement (Setzen der Größen von Speicher-Bereichen, wie z.B. DB_CACHE_SIZE, SHARED_POOL_SIZE). Des Weiteren ist der DBCA für die initiale Einrichtung von Tablespaces verantwortlich.

Oracle bietet zudem eine schnelle und leichte Installation mittels **Oracle Universal Installer**. Im Anschluss können Datenbanken einfachst mittels des **Database Creation Assistant** erstellt und konfiguriert werden. Der **Database Upgrade Assistant** erlaubt automatische, flexible und fehlertolerante Upgrades, im Sinne des Einspielens neuer Fixpacks oder bei einem DBMS-Releasewechsel.

Der Microsoft **SQL Server Configuration Manager** ist ein Tool zur Verwaltung der Dienste des SQL Servers und Konfiguration der Netzwerkprotokolle. Damit vereinigt der MS SQL Server Configuration Manager bisherige Tools wie den Service Manager, das Client Network Utility sowie das Server Network Utility. Das **SQL Server Management Studio**, ein neuer Bestandteil von Microsoft SQL Server 2005, bietet darüber hinaus eine integrierte Umgebung und eine umfassende Sammlung an grafischen Tools zum Zugreifen,

Konfigurieren, Verwalten und Entwickeln aller Komponenten des SQL Server. SQL Server Management Studio kombiniert die Fähigkeiten des Enterprise Manager, Query Analyzer und Analysis Manager aus den vorherigen Versionen von SQL Server in einer einzigen Umgebung und stellt für Administratoren ein einziges Dienstprogramm mit einfachen grafischen Tools und umfangreichen Skripterstellungsfunktionen zum Ausführen ihrer Aufgaben bereit.

Der MS SQL Server entlastet den DBA durch eine vereinfachte Installation mittels Microsoft **Windows-Installer-Paket** und bietet in Problemfällen zur Konfigurations- bzw. Laufzeit **Reporting Services** mit dem Ziel der Weiterleitung der richtigen Informationen an die zuständigen Personen. Das **Declarative Management Framework** (DMF) ermöglicht ferner das Erstellen und Ausführen von Konfigurationsrichtlinien für einen oder mehrere Datenbankserver, um sicherzustellen, dass für alle Zielserver und Zieldatenbanken Standard-Konfigurationseinstellungen angewendet und verwaltet werden.

Sämtliche Tools sind, trotz der Möglichkeit, die vorgeschlagenen Parameter-Änderungen auch durchzuführen, letztlich eher statischer Natur und müssen zu opportunen Zeitpunkten vom Nutzer angestoßen und mit Input versorgt werden. Dynamisch hingegen werden einige der wesentlichen Systemressourcen (Primär- und Sekundärspeicher) anhand weniger Parameter automatisch konfiguriert und ggf. kontinuierlich angepasst. Das DBMS belegt bei Bedarf selbst Speicher und bleibt dabei entweder in von den Nutzern festegelegten oder in den systembedingten Grenzbereichen. Näheres dazu findet sich in Abschnitt 3.1.3.

3.1.2 Physischer Entwurf

Die bedächtige physische Organisation der Daten ist durch ansteigende Datenvolumina unverzichtbar geworden. Sie beschäftigt sich mit der Speicherung und Verwaltung der Relationen einer Datenbank sowie der darauf aufsetzenden Hilfsstrukturen, um eine möglichst große Effizienz und Performance im Kontext der Applikationen zu ermöglichen. Zu den wichtigsten und herausforderndsten Tuning-Aufgaben im Rahmen des physischen Entwurfs zählt die Bestimmung der Indexe und materialisierten Sichten, die eine gegebene Menge von Anfragen am besten unterstützen. Für eine entsprechende Auswahl sollte naturgemäß der Gewinn durch die Zusatzstrukturen die zusätzlichen Kosten für den Speicherplatz und die Verwaltung (Anlegen, Aktualisieren) überwiegen [Skat06].

Entsprechend stellen führende Datenbankhersteller diverse (Ratgeber-)Tools bereit, die sich dieser und anderen Aufgaben des physischen Datenbankentwurfs stellen, indem sie die Workload analysieren und darauf aufbauend Empfehlungen für eine optimale Menge an vom Optimizer zu berücksichtigenden Indexen, materialisierten Sichten und ggf. Partitionen abgeben.

Die korrekte, reale Arbeitslast, unter welcher die Datenbank sich befindet, spielt eine entscheidende Rolle bei dem physischen Datenbank-Entwurf. Aus diesem Grund verlangen viele Tools User-Input über die vorherrschende Workload auf der Datenbank. So kann der DBA die repräsentativen Statements (ggf. mit Gewichtung) in Form eines Last-Profils an die Tools übergeben. Alternativ können die Workload-Statements aber auch automatisch protokolliert und an die Tools übergeben bzw. von ihnen ausgelesen werden. Weiteren Input können Speicher- bzw. Zeitbeschränkungen für das Finden eines „optimalen“ physischen Layout bilden.

Beim Microsoft SQL Server ist der **Database Tuning Advisor** (DTA) als Nachfolger des Index Wizard für die Vorschläge zur Optimierung des physischen Datenbank-Designs zuständig. Zu seinen Funktionalitäten gehören Vorschläge für das Anlegen von Indexen, materialisierten Sichten und die Durchführung der horizontalen Partitionierung für eine vorgegebene Workload. Dabei hat der DBA die Möglichkeit, einen Teil der Zielkonfiguration vorzugeben und vom System die restlichen Konfigurationseinstellungen bestimmen zu lassen. Darüber hinaus kann der DBA die zu betrachtenden Tabellen und den maximal zur Verfügung stehenden Speicherplatz für die Datenbank(en) inklusive aller physischen Datenstrukturen (PDS) beschränken. Die Workload-Statements können manuell eingegeben, aus einer Text- oder XML-Datei oder in Form von Trace-Files bzw. Trace-Tables aus dem SQL Server Profiler entnommen und mit Prioritäten versehen werden. Der Profiler bietet Mechanismen zum Tracen spezieller Events (z.B. Commit, Deadlock etc.). Durch die korrekte Nutzung solcher Traces (vor allem im richtigen Zeitraum) ist es möglich, eine repräsentative Workload anhand der tatsächlich angefallenen SQL-Statements zu erstellen. Vorteilhaft ist bei dieser Methode weiterhin, dass die Ausführungsdauer der Statements aufgezeichnet wird und der DTA dadurch eine Gewichtung der Prioritäten vornehmen kann. Konkret bedeutet dies, dass solche PDS bevorzugt werden, welche vor allem langlaufende Queries beschleunigen könnten. Leider sind Nachbearbeitungen von Traces, wie beispielsweise die Änderung der Priorität durch Manipulation der Dauer, nur manuell und eingeschränkt möglich.

Ausgehend von dieser Workload werden zunächst Index-Kandidaten ausgewählt. Anschließend werden durch Nutzung sogenannter „What-If-Indexe" verschiedene Index-konfigurationen untersucht und nach dem Greedy-Verfahren die voraussichtlich beste ermittelt. Die Existenz der Indexe wird dabei dem Anfrage-Optimierer vorgetäuscht, der unter diesen Umständen für die Kostenabschätzung genutzt werden kann. Dabei verfolgt der DTA einen integrierten Tuning-Ansatz und optimiert gleichzeitig *mehrere* Datenbanken, falls sich Statements in der übergebenen Workload auf verschiedene Datenbanken beziehen. Sind alle Analysen abgeschlossen, bekommt der DBA eine, unter den gegebenen Bedingungen, als optimal errechnete Konfiguration vorgeschlagen. Er hat die Wahl, die Vorschläge direkt zu übernehmen oder in einem Skript zu speichern. Dieser Output wird von einer Reihe zusätzlicher Berichte ergänzt, welche die Auswirkungen auf die Workload abschätzen, falls das empfohlene physische Datenbank-Design übernommen wird. Diese Berichte geben z.B. darüber Auskunft, wie sich die Ausführungszeiten einzelner Statements unter Zuhilfenahme welcher PDS verändern.

Der Microsoft **Physical Design Alerter** kann erkennen, wann der ressourcenhungrige DTA aufgerufen werden sollte und bestimmt im Falle der Ausführung die untere und obere Schranke für die erwartete Verbesserung.

Bereits in DB2 Version 8.1 wurden verschiedene autonome Funktionen bereitgestellt, welche die Administration, das Tuning und die Pflege der Datenbanken erleichtern sollen. Dabei wurde u.a. der **DB2 Design Advisor**, als Nachfolger des ehemaligen Index Advisor, um Empfehlungen für eine Vielzahl physischer Strukturen erweitert. Der DB2 Design Advisor dient nunmehr der Verbesserung der bestehenden Datenstrukturen *einer* Datenbank in Bezug auf eine definierte Workload. Die repräsentativen Workload-Statements können einzeln in die Befehlszeile eingegeben werden, in Dateien oder Tabellen stehen, aus dem Cache des dynamischen SQL oder dem Query Patroller, einem Tool zur Workload-Überwachung, übernommen werden. Im Gegensatz zu Oracle mit seinem Automatic Workload Repository (AWR) besitzt DB2 keine zentrale Ablage, in der

regelmäßig Informationen über die Workload gesammelt werden. Das Ergebnis des Advisor-Laufs sind Vorschläge für (1) Indexe, (2) materialisierte Sichten (MQTs, materialized query tables), (3) mehrdimensionale Cluster-Tabellen (MDCs, multidimensional clustering) und (4) horizontale Tabellen-Partitionierung. Des Weiteren werden dabei auch nicht genutzte Strukturen aufgelistet, die der Administrator bei Bedarf löschen kann.

Im Rahmen der Erstellung einer Data-Warehouse-Lösung ist auch der **Cube Views Optimization Advisor** als Teil des separaten Aufsatzes DB2 Cube Views erwähnenswert, der besonders bei der Erstellung von MQTs für multidimensionale Analysen hilft.

Auch bei DB2 werden durch Heuristiken als potentiell nützlich deklarierte Indexe zunächst virtuell angelegt. Der DB2 Design Advisor bedient sich intern des erweiterten Optimizer, der die Last für verschiedene Konfigurationsvarianten durchrechnet, unter Verwendung der virtuellen Indexe den besten Plan bestimmt und die genutzten Indexe als Empfehlung ausgibt. Zusätzlich gibt es einen Modus, in dem zufällig Indexe der initialen Lösung des sogenannten Rucksackproblems [MaTo90] bis zum Überschreiten einer vorgegebenen Zeitschranke ausgetauscht werden.

Oracle Database stellt eine Vielzahl von Ratgebern bereit, die für die verschiedensten Aufgaben gedacht sind. Damit diese konsistent und gleichartig funktionieren, existiert das **Advisor Framework**. Dieses stellt sicher, dass die Ratgeber auf die gleiche Weise aufgerufen werden können und die Ergebnisse in gleicher Form - in der Regel über den Oracle Enterprise Manager - präsentiert werden. Hauptsächlich werden die Ratgeber vom Datenbank-Management-System selbst aufgerufen und genutzt. Über den Enterprise Manager oder die dokumentierten APIs (Application Programming Interface) ist es allerdings auch möglich, sie explizit zu starten. Der zentrale Ratgeber ist der Automatic Database Diagnostic Monitor (ADDM), der bei Bedarf auf alle anderen Ratgeber zugreift, deren Ausführung empfiehlt und deren Ergebnisse dann in aufbereiteter Form an den Nutzer weiter gibt. Nähere Details finden sich in Abschnitt 3.1.4.

Der **Oracle SQL Access Advisor** beispielsweise analysiert die Datenstrukturen für eine durch den DBA vorgegebene Workload und empfiehlt das Anlegen, Beibehalten und Löschen von Indexen und materialisierten Sichten. Die Workload-Informationen können hierbei automatisch aus dem Inhalt des SQL Cache, in Form von SQL-Tuning-Sets aus dem Workload Repository und aus dem Summary Advisor gewonnen werden. Zudem hat der Datenbank-Administrator, ähnlich wie in DB2, die Möglichkeit eine Tabelle mit repräsentativen SQL-Anweisungen zu befüllen und diese als Grundlage für die Arbeitslast zu nehmen. Der Administrator kann daneben die zur Arbeitslast gehörenden einzelnen SQL-Anweisungen über die Kommandozeile hinzuzufügen.

Der Aufruf des Advisors kann sowohl aus einer Empfehlung des ADDM als auch manuell über den Enterprise Manager oder die Kommandozeilen-API erfolgen. Der Ratgeber schließt seine Analysen mit einer Empfehlung ab, die nach einer Bestätigung des Administrators automatisch umgesetzt wird. Wie auch die anderen Ratgeber, berücksichtigt er dabei die Auswirkungen der vorgeschlagenen Objekte auf die Gesamtlast auch hinsichtlich der Änderungsoperationen. Dieser Ratgeber stellt insbesondere eine Ergänzung zu dem SQL Tuning Advisor (siehe Abschnitt 3.1.4) dar und bildet mit ihm zusammen einen wichtigen Bestandteil der automatischen Optimierungsfähigkeiten.

Bedingt durch ihre Arbeits- und Wirkungsweise eignen sich die genannten Ratgeber-Werkzeuge vornehmlich für statische bzw. vorhersehbare Workloads. Die Entscheidung,

ob und wann neue Strukturen erzeugt, bestehende angepasst oder gelöscht werden, bleibt weiterhin dem DBA vorbehalten. In der durchaus realistischen Praxis mit vielen dynamischen Ad-hoc-Anfragen sind passende Indexe etwa jedoch nur schwer vorherzusagen und oft nur kurzzeitig von bedeutendem Mehrwert. Als Beispiel wären hier temporäre Tabellen im OLAP-Umfeld zu nennen. Neben Änderungen der Workload bieten Änderungen des (relativen) Datenvolumens bzw. der Datenverteilung in Tabellen, Änderungen am Datenbankschema oder an der (Hardware-)Infrastruktur sowie etwaige Konflikte mit anderen Tuning-Maßnahmen Anlass zur Anpassung der physischen Organisation einer Datenbank.

3.1.3 Systemverwaltung (Ressourcen-Management, Pflege und Wartung)

Die Verwaltung (der Ressourcen) eines Datenbanksystems stellt eine Menge von Herausforderungen an die Administratoren und adressiert diverse Bereiche. Dazu zählen im Konkreten unter anderem das Ressourcen-Management und insbesondere die Bereiche des Memory Management, Storage Management, Maintenance, Availability uvm. Zur Entlastung des DBA und der Maskierung der systeminhärenten Komplexität ist es daher förderlich, autonome Tools oder interne Mechanismen für jeden der Anwendungsbereiche zur Verfügung zu stellen.

3.1.3.1 Dynamische (Re-)Konfiguration

Eine der schwerwiegenden Aufgaben im Leben eines DBA ist die andauernde Last- bzw. Problem-verursachte Konfiguration von DBMS-Ressourcen (im laufenden Betrieb). Heutige DBMS bieten hierfür unzählige Stellschrauben auf Instanz- oder auch Datenbankebene. Deren Werte, gegenseitige Abhängigkeiten und Einflüsse auf das Systemverhalten (insbesondere im Hinblick auf die Performance) gilt es ständig im Auge zu behalten, Änderungen an der System-Umgebung (Workload-Variationen, Hardware etc.) selbstständig zu erkennen und daraufhin die Parameterwerte individuell und dynamisch, schnell reagierend anzupassen. Idealerweise soll dies ohne den Administrator und ohne Störung des laufenden Betriebs erfolgen. Das ist jedoch bei keinem der untersuchten DBMS uneingeschränkt möglich, da in den meisten Fällen alle aktiven Datenbank-Verbindungen zu trennen sind bzw. ein Neustart der Instanz notwendig ist, um die Änderungen wirksam werden zu lassen.

Die DBMS-Hersteller gehen mehr und mehr dazu über, dem DBA immer weniger „händisch“ zu adjustierende Konfigurationsparameter zu überlassen, um so die Komplexität zu verbergen und Fehlentscheidungen zu vermeiden. DB2 hat trotz diverser automatischer Parameter noch immer über 100 rein manuell änderbare Stellschrauben auf Instanz- und Datenbankebene, die es einerseits recht flexibel und individuell anpassbar an spezielle Workloads und System-Umgebungen machen und das „Herausquetschen“ auch des letzten Quäntchen Performance erlauben, andererseits den unerfahrenen DBA jedoch vor einen schier unüberschaubaren „Konfigurationsurwald“ stellen. In Oracle wird durch eine Unterteilung der Initialisierungsparameter in „Basic“ und „Advanced“ versucht, die Menge derer überschaubar zu halten. So werden für die tägliche Arbeit nur 28 Basic-Parameter vom DBA benötigt. Microsoft SQL Server bietet auch noch 66 Konfigurationsparameter[12]. Davon erfordern 14 einen Neustart (entweder der Instanz oder des gesamten Systems), sechs sind selbst-konfigurierend. Von letzteren gibt es aber noch immer zwei, die einen Neustart benötigen.

[12] http://msdn2.microsoft.com/de-de/library/ms189631.aspx

3.1.3.2 Memory Management

In **DB2** ist es möglich, für die zu nutzenden Hauptspeicherbereiche (Buffer Pools, Package Cache, Sort Heap, Locklist, ...) eine Obergrenze auf Datenbankebene (DATABASE_MEMORY) oder auch Instanzebene (INSTANCE_MEMORY) vorzugeben. Der **Self Tuning Memory Manager** (STMM) von DB2 überwacht die zur Verfügung stehenden Speicherressourcen und nimmt dann durch automatisches, Workload-abhängiges Anpassen verschiedener (im Vorhinein durch den DBA auf den Wert „Automatic" gesetzter) Konfigurationsparameter die Aufteilung des ausgewiesenen Hauptspeichers auf die einzelnen Bereiche, sogar Datenbank-übergreifend, vor. Demnach können die Speicher-Konsumenten dynamisch in ihrer Größe schrumpfen und anwachsen, je nach Bedarf und je nach verändertem Workload-Aufkommen. Diese neue Möglichkeit entlastet bzw. befreit von der mühseligen Aufgabe der DB2-Konfiguration durch wiederholtes Anpassen der relevanten Konfigurationsparameter und dem Anpassen der Bufferpools, indem es sowohl die jeweiligen Parameter, als auch die eigene Tuning-Frequenz gemäß den Anforderungen an die Workload und die ermittelten Umstände anpasst.

Auf Windows- und AIX-Plattformen kann STMM den gesamten Speicherbedarf überwachen und entsprechend die Shared-Memory-Bereiche von DB2 optimieren. Dabei nimmt sich DB2 je nach Last mehr Hauptspeicher oder gibt ihn wieder frei. Wenn DB2 lastabhängig seine Speicherbereiche anpassen kann, entlastet es nicht nur den DBA bei der Konfiguration, sondern optimiert auch dynamisch seine Performance. Die Vorteile von STMM sind insbesondere groß, wenn das Lastprofil hinsichtlich der Speicheranforderungen unbekannt ist, das Lastprofil stark schwankende Speicheranforderungen über die Zeit aufweist, der Server viel freien Speicherplatz zur Verfügung hat, die DB mehrere Bufferpools mit unterschiedlichen Seiten-Größen besitzt oder wenn der DBA recht unerfahren ist. Weniger sinnvoll ist der Einsatz von STMM, wenn das Lastprofil mit seinen Speicheranforderungen wohl bekannt ist, diese Anforderungen eher statischer Natur sind oder wenn der DBA sehr erfahren in DB2 ist.

Beim **Automatic Shared Memory Management** (ASMM) von **Oracle** gibt der DBA eine Obergrenze (SGA_MAX_SIZE bzw. SGA_TARGET) für den verfügbaren Hauptspeicherbereich[13] (SGA, System Global Area) vor und Oracle verteilt den Speicher in Abhängigkeit von der Workload auf seine Bereiche (Buffer Cache, Shared Pool, Large Pool, Java Pool). Wechselnde Speicheranforderungen zwischen diesen Bereichen werden im laufenden Betrieb automatisch ausgeglichen. Im Gegensatz zu DB2 und Microsoft SQL Server kann dies jedoch nur im Rahmen des zum Start fest allokierten Speichervolumens geschehen. Es kann somit weder zusätzlicher Speicher vom Betriebssystem angefordert, noch an jenes zurückgegeben werden. Als nachteilig sei noch erwähnt, dass im Gegensatz zu DB2 eine dynamische Anpassung der Sortier-Bereiche nicht möglich ist, da diese bei Oracle in der PGA (Program Global Area) liegen und damit kein Bestandteil des Shared Memory sind. Weiterhin ist das dynamische Speichermanagement derzeit auf den Default Buffer Cache beschränkt. Bei den anderen DBMS abgedeckte Aspekte wie die Verwaltung von Katalog-, Sortier- oder Locking-Speicher sind also bspw. ausgenommen.

Ähnlich zum **DB2 Memory Visualizer** bietet **Oracle** einen **Memory Advisor** an, der jedoch nur beim deaktivierten Automatic Memory Management benutzt werden kann. Er ergänzt das umfangreiche Oracle-Advisor-Framework um drei Ratgeber: den Shared Pool

[13] Bei der System Global Area handelt es sich um einen Speicher, der von allen Oracle-Prozessen geteilt wird. Die Program Global Area dagegen ist ein Speicher, der für jeden Prozess einzeln und unabhängig existiert.

Advisor (SGA), den Buffer Cache Advisor (SGA) und den PGA Advisor. Mit seiner Hilfe können nicht nur die Größen der einzelnen Speicherbereiche, sondern auch der Einfluss der Speichergrößen-Veränderungen auf die Datenbank-Performance bestimmt werden.

Das Memory Management von **Microsoft** SQL Server überzeugt dagegen mit wesentlich ausgereifteren Mechanismen. So wird der Arbeitsspeicher für diverse Caches dynamisch und völlig autonom zugewiesen. Der SQL Server versucht, so viel Speicher zu allokieren, wie nötig ist, ohne dabei Speicherknappheit auf dem System zu verursachen. Es kann damit auf externen, vom Betriebssystem verursachten und auch internen, von SQL-Server-Komponenten hervorgerufenen Speichermangel reagieren und bei etwaigen Engpässen eine Umverteilung vornehmen. Der SQL Server bestimmt dabei automatisch, wieviel Hauptspeicher zum Start zu allokieren ist, basierend darauf, wieviel Speicher das Betriebssystem und andere Applikationen momentan nutzen. Wenn sich die Last auf dem Rechner und damit auch auf SQL Server ändert, so ändert sich auch der allokierte Speicher. Eingriffe der DBAs sind konfigurativer Natur und somit meist nur selten nötig. Es werden lediglich einige wenige Konfigurationsparameter (MIN/MAX SERVER MEMORY, MAX WORKER THREADS, ...) angepasst, so dass bei Bedarf die fortlaufende Speicherallokation bis zum Erreichen von „MAX SERVER MEMORY“ bzw. die Speicherfreigabe bis zum Erreichen von „MIN SERVER MEMORY“ erfolgt.

Der Memory Broker überwacht den aktuellen Speicherverbrauch einzelner Komponenten (Bufferpool, Query Executor, Query Optimizer und weitere Caches) des Memory Pools und vermag, basierend auf der Workload, Trends im Speicherverbrauch festzustellen. Deutet sich an, dass ein zukünftiger Verbrauch nicht mehr durch den zugesicherten physischen Speicher gedeckt werden kann, werden, basierend auf den Ergebnissen der Analyse, individuelle Zielgrößen für den möglichst optimalen Speicherverbrauch der Komponenten ermittelt. Diese Pläne werden an die Komponenten gesendet, welche daraufhin den Speicherverbrauch entsprechend anpassen. Je nach Zielgröße werden Komponenten veranlasst Speicher freizugeben bzw. keinen weiteren mehr zu beanspruchen.

Die Berechnung der anzustrebenden Speicherverbräuche und der Austausch erfolgt mehrmals pro Sekunde. Trotzdem kann es zu Out-Of-Memory-Situationen kommen, falls mehrere Komponenten versuchen, gleichzeitig zusätzlichen Speicher zu belegen [BCC+07]. Es kommt letztlich auch darauf an, dass die vom Memory Broker koordinierten Komponenten intelligent agieren und zusätzlichen Speicher nur anfordern, wenn auch entsprechende Performance-Verbesserungen zu erwarten sind. Zudem muss nicht mehr benötigter Speicher zeitnah freigegeben werden, getreu dem Motto „so viel wie nötig, so kurz wie möglich“.

3.1.3.3 Storage Management

Die Dateien der Datenbank liegen in der Regel auf Platten oder Platten-Subsystemen, die über (Logical) Volume Manager verwaltet werden. Teilweise werden auch Raw Devices eingesetzt. Die Konfiguration des Plattenspeichers hat großen Einfluss auf die Performance von Datenbanken. Das Konfigurieren von Storage-Systemen, das Einrichten von Laufwerken und Datenbank-Dateien ist ein aufwendiger Vorgang. Die Optimierung einer solchen Konfiguration und das Umlegen von Datenbank-Dateien sind meist nur manuell und bei Stillstand der Datenbank und ihrer Anwendungen möglich. Neben den bereits in Abschnitt 3.1.1. angesprochenen Advisor-Komponenten für die Initial-DB-Konfiguration, erlauben Mechanismen, wie das automatische Storage Management das

(automatische) Anwachsen der Größe der Datenbank über Platten und Dateisysteme ohne Eingriff des DBA.

Besteht eine Datenbank aus Tablespaces und zugeordneten Containern, war in früheren Versionen von DB2 noch eine manuelle Anpassung notwendig, wenn sich das gespeicherte Datenvolumen vergrößert hat. Ab **DB2** Version 8.2 können Datenbanken mit der Option **AUTOMATIC STORAGE** definiert und ihre Tablespaces mit MANAGED BY AUTOMATIC STORAGE angelegt werden. Damit ist das automatische Wachsen der DB über die Platten und Dateisysteme hinweg, ohne jegliche Verwaltung der Storage Container, gewährleistet. Container- und Speichermanagement-Merkmale werden dabei vollständig durch den DB2-Datenbankmanager bestimmt. Seit Version 9 ist diese Option vorab aktiviert und auch bei partitionierten Datenbanken möglich. Darüber hinaus bietet DB2 ein *Storage Management Tool*, mit dem der DBA wertvolle Statistiken sammeln und Schwellwerte für automatische Alarme definieren kann. Das *Space Estimation Tool* wiederum hilft bei der Berechnung des benötigten Datenbank-Platzes. Beide Tools sind integraler Bestandteil von DB2.

Mit dem **Automatic Storage Management** (ASM) nimmt **Oracle** dem DBA die Aufgaben der Platten(platz)verwaltung ab und optimiert die zur Verfügung gestellten Speicherbereiche (sog. Disk Groups) selbst. Durch einfache Installation und Verwaltung des Storage-Systems (Raw-Devices, Cluster-File-Systeme, Mirroring, Striping etc.) sollen die anfallenden Administrationsaufgaben vereinfacht und vom Systemadministrator hin zum Datenbank-Administrator verlagert werden. Auf diesen sogenannten ASM-Disks können alle Dateien einer Datenbank (Control Files, Log Files, Data Files, Archive Files) abgelegt werden. Lediglich TraceFiles brauchen ein herkömmliches File-System.

Das **Storage Management** bei **MS SQL Server** weist diesbezüglich sehr einfache Verfahren auf, um die Kapazität einzelner primärer und sekundärer Datendateien automatisch zu erhöhen (Auto Grow) bzw. (optional) zu verringern (Auto Shrink). Trotz dieser Mechanismen sind DBAs jedoch hochgradig in das Storage Management involviert. Sie tragen die alleinige Verantwortung für das Hinzufügen von Platten, die Konfiguration von Dateien bzw. Filegroups und deren Verteilung auf Festplatten.

Eine durchaus wichtige Tätigkeit in der heutigen „Volumengesellschaft“ ist die physische Vergrößerung des Plattenplatzes, d.h. insbesondere das Hinzufügen und Umkonfigurieren von Platten im laufenden Datenbankbetrieb. Um den neuen Speicherplatz dann tatsächlich auch der Datenbank verfügbar zu machen und um die Daten entsprechend zu verteilen, gibt es eine Vielzahl an Möglichkeiten. Das Automated Storage Management von Oracle erlaubt die automatische Rebalancierung einer Disk Group im Falle des Hinzufügens einer neuen Disk, kann das System aber für eine ganze Weile ausbremsen. Das AUTOEXTEND-Feature von Microsoft SQL Server ermöglicht die Vergrößerung von Filegroups ohne die Notwendigkeit einer Rebalancierung. Die DB2 System Managed Tablespaces arbeiten auf ähnliche Weise und werden je nach Bedarf solange automatisch vergrößert, bis es keinen Freiplatz mehr auf der zugrundeliegenden Platte gibt. Die alternativen Database Managed Tablespaces operieren performanter und arbeiten mit „Balanced Files“, d.h. das Hinzufügen einer Datei sorgt für eine Umverteilung der Daten in dem Tablespace.

3.1.3.4 Systemwartung (Maintenance)

Eine der Hauptaufgaben eines DBA liegt in der Wartung des Systems im laufenden Betrieb. Zur ungeliebten Routine des Administrators gehören dabei die regelmäßigen Sicherungen (Backup und Recovery Management), Reorganisationen (Fragmentierungs-Überwachung) sowie damit und mit dem operativen Betrieb verbundenen Arbeiten, wie das Sammeln aktueller Statistiken (Statistics Management) oder das „REBIND" betroffener Packages. Entlastung verschaffen hier schon seit längerem Werkzeuge verschiedener DBMS-Anbieter, die diese Maßnahmen abhängig von diversen Parametern automatisch ausführen.

Unter dem Schlagwort **Autonomic Object Maintenance** fassen die Hersteller nun Funktionen zur automatisierten Administration und Wartung einer Datenbank zusammen und versprechen automatisches Überwachen und Adaptieren zur Laufzeit. Dazu gehören u.a. jene oben genannten Aktivitäten, wie die Sicherung der Datenbank, die Reorganisation von Tabellen und das Sammeln von Statistiken. Der Administrator kann individuelle Rahmenbedingen, wie die Zeit eines Wartungsfensters oder die Notwendigkeit zu reinen Online-Operationen, vorgeben.

Unter **Backup** versteht man die Sicherung der Daten als Schutz vor „Zerstörung", etwa durch Datenträgerausfall, Fehler in Hard- oder Software oder Benutzerfehler. Die Daten werden normalerweise in regelmäßigen Abständen, durch den DBA gesteuert, gesichert (auf Magnetbänder oder ähnliche Speichermedien) und an einem sicheren Ort gelagert (im Tresor, oft an einem geographisch verschiedenen Ort). Sollte ein Datenverlust auftreten, dann können eine dieser Sicherungskopien und die Änderungen, die seitdem im Datenbanksystem durchgeführt wurden, anhand des System-Logs wieder eingespielt werden. Die Crash-Recovery, d.h. der Wiederanlauf nach einem Systemfehler, sollte indes im Normalfall ohne Mitwirkung des DBA erfolgen und muss von diesem auch nicht besonders vorbereitet werden. Während die Datensicherung eine Routineaufgabe sein muss, falls sie erfolgreich sein soll, ist die **Datenwiederherstellung** (Recovery) eine zeitkritische Ausnahmetätigkeit. An die Datensicherung kann es verschiedene Anforderungen bezüglich ihres Verhaltens geben (z.B. niedrige Dauer, geringe Belastung, inkrementelle Sicherung, partielle Sicherung etc.). Die Wahl des Sicherungsverfahrens hat unmittelbaren Einfluss auch auf die Dauer der Datenwiederherstellung [Stö01].

Alle drei DBMS erlauben ein vollständiges bzw. inkrementelles Backup im Online-Modus sowie ein Restore auf Datenbank- bzw. Tablespace-Ebene. Oracle bietet diesbezüglich sogar, im Falle eines Media Failures optimale, Block-Granularität, bei der man selektiv „korrupte" Blöcke auswählen und wiederherstellen kann. Oracle bietet zudem einen nutzerfreundlichen, mitgelieferten *Recovery Manager*. Oracles *Log Miner* ermöglicht hingegen fortgeschrittenes Analysieren archivierter Logs. Das konfigurierbare *Throttling-Feature* von DB2, zur Last-abhängigen Reduktion des Backup-Einflusses auf das System, hat kein Pendant in Oracle oder SQL Server. Um jedoch die Zeit für das Backup zu reduzieren, erlauben alle 3 DBMS die parallele Ausführung durch mehrere Prozesse. In DB2 ist es für das Backup und das Recovery zudem möglich, die Anzahl und Größe der verwendeten Puffer sowie den Wert für die CPU- bzw. IO-Parallelität automatisch von den Kommandos BACKUP DATABASE und RESTORE DATABASE ermitteln zu lassen. Ferner bietet DB2 einen zusätzlichen Ratgeber, den *Recovery Expert***.** Das Tool stellt leistungsfähige Funktionen zur Diagnose, Berichterstellung und automatisierten Wiederherstellung (recovery) bereit, welche die Erhaltung korrekter Daten und einer hohen Verfügbarkeit unterstützen. Analog zu anderen Advisor- und Wizard-Komponenten gilt

auch hier die Maxime, dass der DBA mit wenigem Input definiert, WAS er tun will und das Tool automatisch das WIE bestimmt.

Die Datensicherung dient lediglich dazu, den aktuellen Datenbestand der Datenbank vor dem Verlust zu bewahren und erfolgt, z.B., in regelmäßigen Abständen. Im Gegensatz hierzu soll **Datenarchivierung** [Sch99] den Datenbestand zu einer bestimmten Zeit festhalten, beispielsweise zu Revisionszwecken oder aus rechtlichen Gründen. Schneller Zugriff auf die Archivdaten ist u.U. nicht erforderlich (es kann Tertiärspeicher eingesetzt werden). Die Archivierung steht nicht im Fokus unserer Betrachtungen und sei daher an dieser Stelle außen vor.

Um der durch häufige Änderungen allmählich entstehenden Fragmentierung der Daten entgegenzuwirken und die Daten erneut zu verdichten, sollte man regelmäßig Tabellen und Indexe **reorganisieren** [Dor06]. Datenbanksysteme sollten die Möglichkeit bereitstellen, alle gespeicherten Daten dynamisch zu reorganisieren und zu restrukturieren. Ziel dieser Reorganisation ist eine Verbesserung der Performance für Anfragen, die an eine Datenbank gestellt werden. Oracle und DB2 bieten Reorg Utilities für alle diese administrativen Aufgaben, mit vollem Zugang zu den Daten, an. Auch hier greifen wieder die Throttling-Mechanismen diverser DB2 Tools. In DB2 gibt es zudem das In-Place Reorg Utility, das die Reorganisation besonders Ressourcen-schonend ohne Zusatzstrukturen durchführt. Des Weiteren kann DB2 als einziges DBMS MDC-Tabellen aufweisen, die keiner Reorganisation bedürfen um die Daten-Clusterung aufrecht zu erhalten. Microsoft SQL Server hat kein explizites Reorg Utility.

Auch das **Laden** großer Daten-Mengen durch ein ausgereiftes Load Utility, welches insbesondere auch Verfügbarkeits-Aspekte abdeckt, ist eine wichtige DBA-Routine-Tätigkeit. Das *DB2 Load Utility* erlaubt den Lesezugriff auf Tabellen und MQTs (Materialized Query Tables) und bietet darüber hinaus eine Massendaten-fähige Bulk-Loading-Option. Diese dient dem Laden großer Daten-Mengen in eine Tabelle und bietet gegenüber dem herkömmlichen IMPORT durch das Schreiben der Daten auf Seiten-Ebene und geringem Logging-Umfang vor allem einen Geschwindigkeitsvorteil. Die autonomen Mechanismen von DB2 greifen nun auch hier. Durch Miteinbeziehung von Tabellen-Charakteristika, Freispeicher und die Anzahl der Tablespace-Container sowie Anzahl an CPUs im System und weiterer Umgebungsparameter können bisweilen händisch zu übermittelnde Utility-Parameter nun automatisch bestimmt werden. So werden automatisch der Speicherverbrauch (Bufferpools und Sorting), der Grad der IO-Parallelität (Anzahl an IO-Subagents) und der CPU-Parallelität (Anzahl an Subagents) ermittelt. Indexe verwaltet das Tool ebenfalls, in Form eines kompletten Neuaufbau oder durch inkrementelles Aufbauen mit jedem eingefügtem Tupel.

Oracle Load kann in zwei Modi laufen. Im höchst performanten (1) Direct Path Load, welches jedoch die Tabelle sperrt, werden vor dem eigentlichen Lade-Vorgang alle Constraints deaktiviert und die Indexe gelöscht, um sie danach wieder neu zu erzeugen und die Constraints zu re-aktivieren. Analog dem DB2 Load Utility steht in Oracle das (2) *Data Pump* für ein schnelles, wiederanlauffähiges Umherbewegen von Massendaten zwischen den Plattformen zur Verfügung.

Das Microsoft SQL Server *Bulk Copy Utility* läuft zudem im transaktionellen Rahmen und erlaubt dadurch den Zugriff auf den Rest der Tabelle. In der Praxis wird bei einem Massen-Insert jedoch die gesamte Tabelle auf Grund der häufig eintretenden Lock-Eskalation gesperrt.

Um den Datenbank-Administrator von periodischen Wartungsarbeiten zu befreien, gibt es den **Configure automatic maintenance wizard** in der DB2 Automatic-Maintenance-

Infrastruktur. In diesem Tool können online und offline Wartungsfenster (geringer oder keiner Aktivität) für die durch das DBMS zu übernehmenden Verwaltungsaktivitäten definiert werden, in denen es möglich ist, Wartungsarbeiten, wie Sicherungen (Backup), Defragmentierungen (Reorg) und das Aktualisieren der Statistiken (Runstats) automatisch bzw. nach gewissen in **Maintenance-Policies** definierten Vorgaben durchzuführen, ohne den laufenden Betrieb zu stören. Wenn DB2 anhand diverser Evaluations-Kriterien die Notwendigkeit einer Wartung erkennt, werden die entsprechenden Schritte im nächsten verfügbaren Zeitfenster „gedrosselt" ausgeführt. Dieses *utility throttling* macht den Hauptunterschied zu den anderen DBMS aus. Des Weiteren ist es möglich, Details zu Kontaktpersonen zu hinterlegen, die im Falle einer nicht erfolgreichen Wartung zu benachrichtigen sind.

Die **Automated Maintenance Task Infrastructure** von Oracle bietet ähnliche Möglichkeiten wie DB2 und ermöglicht die automatische Ausführung von Routineaufgaben innerhalb eines vordefinierten Wartungsfensters. Diese Funktionalität wird über den Scheduler realisiert und ist gleichermaßen über den Enterprise Manager als auch über die PL/SQL-Schnittstelle steuerbar. Anhand der vom Automatic Workload Repository erstellten (Performance-)Daten wird ein Profil erzeugt, das das Benutzerverhalten auf der Datenbank widerspiegelt. So kann die Oracle-Datenbank selbstständig bestimmen, zu welchem Zeitpunkt welche Routineprozesse laufen sollen, wie z.B. die Erzeugung oder Aktualisierung von Indexen etc. Diese Prozesse werden über den Unified Scheduler gesteuert und in einem Wartungs-Zeitfenster ausgeführt. In der Voreinstellung läuft dieses Zeitfenster nachts von 22 Uhr bis 6 Uhr. Die Grenzen sowie der Umfang des Zeitfensters und das Intervall sind vom Administrator frei konfigurierbar.

Das SQL Server **Management Studio** als zentrale Anlaufstelle für den DBA dient der einfachen Entwicklung, Wartung und Verwaltung von Datenbanken, indem es ein einheitliches und übersichtliches User Interface zur Vereinfachung der Aufgaben anbietet. Eine der Sub-Komponenten ist **SQL Management Objects** zur Erstellung von neuen Datenbanken und dem Erledigen administrativer Aufgaben. Mittels *SQL Server Maintenance Plans* und dem Database **Maintenance Plan Wizard** können menügeführt Wartungspläne definiert und damit Reorganisationen von Daten- und Index-Seiten, interne Konsistenzprüfungen, Komprimierungen sowie Backups angestoßen werden.

3.1.3.5 Verwaltung von Statistiken

Bei der Übersetzung von SQL-Befehlen wählt der Optimizer aufgrund der Katalogstatistiken die Zugriffspfade aus, die er als optimal errechnet. Dennoch können die ausgeführten Zugriffe weit hinter den aufgrund von Vorab-Zugriffsplänen erwarteten Ergebnissen zurückbleiben. Ursachen können in nicht vorhandenen bzw. einer mangelnden Aktualität der Katalogstatistiken ebenso begründet sein wie in Modellierungs-Unzulänglichkeiten, etwa von Abhängigkeiten zwischen Attributen, welche durch fehlende Statistiken nicht berücksichtigt werden.

Alle DBMS bieten aus diesem Grund eine automatisierte Statistikerstellung (**Automatic Statistics Collection**), die regelmäßig bzw. anhand vorher definierter Zeitfenster relevante Statistiken aktualisiert. Dieser Aspekt versorgt den kostenbasierten Optimizer mit aktuelleren Informationen und kann somit ein besseres Laufzeitverhalten implizieren. Die Statistik-Generierung erfolgt online und mit minimalem Einfluss auf andere Operationen, in DB2 sogar mittels Throttling.

Für sehr große Tabellen erlauben die Datenbanksysteme sogar das **Sampling**, bei dem nicht mehr die gesamte Tabelle, sondern lediglich ein in der Größe erheblich reduzierter

Ausschnitt (Sample) analysiert werden muss. Oracle bietet zudem das Dynamic Sampling. Hierbei werden Statistiken gesammelt, während die Query optimiert wird. Mit dem DB2 Release 9.5 unterstützt DB2 ebenfalls die Erstellung von *Echtzeit-Statistiken,* die, sobald für die Ausführung einer Abfrage benötigt, automatisch erfasst werden.

Darüber hinaus überwacht der **Learning Optimizer** von DB2 (kurz LEO) die Ausführung der Zugriffe und die Statistiken. Hierzu vergleicht er die geschätzten Annahmen, die er bei der Übersetzung eines SQL-Befehls und der Auswahl der passenden Zugriffspfade getroffen hat, mit den tatsächlichen Ergebnissen der Ausführung des Befehls. Der Overhead für dieses zusätzliche Monitoring soll laut IBM weniger als 5% betragen. Bei größeren Abweichungen sind eine Überprüfung der Annahmen und ihre Korrektur notwendig. LEO kann automatisch eine Menge möglicher Ursachen für derartige Abweichungen erkennen und versucht diese durch genauere Statistiken zu beheben (Progressive Optimization, **POP**). Hierzu können dem Dienstprogramm RUNSTATS entsprechende Parameter mitgegeben und der SQL-Befehl anschließend erneut, mit verbesserten Statistikdaten, übersetzt werden.

Anfrage-Optimierungen können zeitaufwendig sein und nicht immer in dem Maße benötigt werden. DB2 erlaubt daher dem Nutzer mit dem **Optimization Level** das Feinjustieren des Arbeits- und Ressourcenumfangs, den DB2 für die Optimierung des Zugriffsplans aufwendet. Oracle und Microsoft scheinen diesbezüglich fortgeschrittener, da sie pro Query selbst den nötigen Optimierungsgrad bestimmen. Zusätzlich kann bei allen DBMS zur Laufzeit automatisch ermittelt werden, ob und in welchem Grad eine Parallelisierung der Query-Abarbeitung durch Mehrkern-CPUs möglich und sinnvoll ist.

Zu den Routineaufgaben innerhalb der **Oracle** Automated Maintenance Task Infrastructure zählen ebenfalls die Sammlung bzw. Aktualisierung der Optimizer-Statistiken (**Automatic Statistics Collection Job**). Dabei wird überprüft, für welche Objekte Statistiken fehlen oder veraltet sind. Diese werden entsprechend erstellt bzw. aktualisiert. Das Oracle Table Monitoring ermittelt dabei, wie viele DML-Befehle auf welcher Tabelle durchgeführt wurden. Wenn mehr als 10 Prozent einer Tabelle verändert worden sind, wird im Hintergrund eine Aktualisierung der Statistiken angestoßen.

Bei den meisten Anfragen generiert der MS SQL Server Anfrage-Optimierer automatisch die notwendigen Statistiken für einen angestrebt hochwertigen Anfrageplan (**AUTO_CREATE_STATISTICS**-Option). In einigen Fällen muss der DBA jedoch weitere Statistiken erstellen oder den Anfrage-Entwurf ändern, um optimale Ergebnisse zu erzielen. Wenn die **AUTO_UPDATE_STATISTICS**-Option zur automatischen Aktualisierung von Statistiken aktiviert ist, stellt der Anfrage-Optimierer fest, wann Statistiken veraltet sein könnten und aktualisiert diese Statistiken, sobald sie von einer Anfrage verwendet werden. Statistiken gelten auch hier als veraltet, wenn die Datenverteilung in der Tabelle durch Änderungs-Vorgänge grundlegend geändert wurde. Der Optimizer ermittelt hierzu die Anzahl der Daten-Änderungen seit der letzten Statistikaktualisierung und vergleicht sie mit einem auf der Anzahl von Zeilen basierenden Schwellenwert.

In einigen Fällen können in MS SQL Server Anfragepläne und damit die Abfrageleistung verbessert werden, indem der DBA selbst zusätzliche Statistiken mit der CREATE-STATISTICS-Anweisung erstellt bzw. mit der UPDATE-STATISTICS-Anweisung oder der gespeicherten Prozedur SP_UPDATESTATS aktualisiert. Damit lassen sich u.a. durch die Anwendung bekannte, statistische Korrelationen aufzeichnen, die vom Anfrage-Optimierer beim Erstellen von Statistiken für Indexe oder einzelne Spalten nicht berücksichtigt werden.

Eine manuelle Aktualisierung von Statistiken ist trotz fortgeschrittener Techniken der Produkte gelegentlich weiterhin erforderlich. Bei langsamen Anfragen bzw. ungenügenden oder nicht vorhersagbaren Antwortzeiten, sollte vor allen nachfolgenden Maßnahmen zunächst sichergestellt werden, dass der Optimizer auf aktuelle Statistiken zurückgreift. Die Erneuerung der Statistiken ist ebenfalls nach dem Durchführen von Wartungsvorgängen, welche die Verteilung der Daten ändern, zu empfehlen. Dazu gehören u.a. Masseneinfügungen für einen großen Prozentsatz von Zeilen bzw. die Reorganisation von Tabellen, oder auch Archivierungsläufe. Durch ein präventives manuelles Vorabeingreifen in die Statistikverwaltung lassen sich zukünftige, durch Warten auf automatische Statistikaktualisierungen entstehende Verzögerungen bei der Anfrageverarbeitung vermeiden.

Letztlich entscheidet das Anwendungs-Szenario, wie oft Statistiken zu aktualisieren sind. Dazu sind die Vorteile optimierter Anfragepläne gegen den Zeit- und Rechenaufwand sowohl für die Sammlung bzw. Aktualisierung von Statistiken als auch für die Neukompilierung von Anfragen abzuwägen.

3.1.4 Performance Management (Überwachung und Tuning)

Der DBA ist verantwortlich für ein zufriedenstellendes Antwortzeitverhalten und einen ausreichenden Durchsatz des Datenbanksystems. Voraussetzung für solche Maßnahmen ist, dass er Engpässe und Überlastungen im Datenbanksystem erkennen kann. Die kontinuierliche Überwachung der Datenbanksysteme und ihrer Umgebungen zur Laufzeit im Rahmen des Performance Management dient dieser Forderung. Fehlersituationen bzw. Performance-Degradationen können frühzeitig erkannt, wenn möglich sogar vermieden und im Problemfalle mit den Mitteln des DBMS ein Tuning und damit die Optimierung der Performance vorgenommen werden. Letzteres kann durch Aktualisierung von Statistiken, das Einplanen von Dienstprogrammen (Utilities), die Änderung der Datenverteilung auf die Speichermedien oder durch Änderung von Laufzeitparametern, wie Puffergrößen u.ä. geschehen. Natürlich kann auch eine Änderung des physischen Datenbankentwurfs Ergebnis einer solchen Engpass-Analyse sein.

Da ein DBA heute oft mehrere Datenbanksysteme betreut, ist eine Unterstützung beim Performance Management durch entsprechende Werkzeuge dringend erforderlich. Derartige Tools werden vom DBMS oder Drittherstellern zur Verfügung gestellt und ermöglichen, gezielt eingesetzt, die Umsetzung einer iterativen rückgekoppelten (Tuning-)Kontrollschleife (siehe Abschnitt 2.1 und 2.2). Die Resultate der kontinuierlichen Sammlung, Speicherung, nachfolgenden Analyse und Aufbereitung von Informationen der systemeigenen Komponenten, der Workload und anderer performance-relevanten Metriken vor, während und nach Problem-Situationen sollten dem DBA in entsprechender Form präsentiert werden und den anderen (autonomen) Komponenten und Operationen als Input dienen.

Zusätzlich zum internen, eher ungeeigneten DB2 System Monitor (siehe Abschnitt 5.3.2) zur bedarfsgesteuerten Sammlung von Snapshop- bzw. Event-Daten dient der **DB2 Health Monitor** einer etwas intelligenteren Überwachung des laufenden DB2-Betriebs. Er basiert auf regelmäßig gesammelten und kurzzeitig gespeicherten Snapshots des DB2 System Monitor. Bei Überschreitung voreingestellter Schwellwerte werden Meldungen an das **DB2 Health Center** (die grafische Anlaufstelle für den DBA, dt. Diagnosezentrale) ausgegeben, die je nach Voreinstellung gezielt als E-Mail an einen Benutzer oder eine Gruppe gesendet werden können. Zu den Alarm-Meldungen und für alle Health-Indikatoren sind Erläuterungen und Empfehlungen (mittels des Health Center internen *Recommendation Advisor*) zur Problemlösung abrufbar. Das DB2 Health Center ermöglicht

zudem die Reaktion auf Events (z.B. zu hohe Anzahl an Deadlocks) durch die Verknüpfung mit Tasks (z.B. Konfigurationsparameter adjustieren, Skripte aufrufen), die im Task Center (Notifications, Conditions) definiert wurden. Als Nachteil sind die beschränkte Anzahl an fest vorgegeben Metriken und die in der Mächtigkeit stark eingeschränkte Event Algebra zur Korrelation der Events zu nennen.

Für die Diagnose und Analyse gibt es auch das Problem Determination Tool **db2pd**. Damit verfügen DBAs über die Möglichkeit, Informationen über Sperren, Bufferpools, Tablespaces, Container, dynamische SQL-Kommandos, Anwendungen, Speicherbereiche, Transaktionen und Log-Dateien aus den internen DB2-Verwaltungsstrukturen abzurufen, ohne dabei Snapshots verwenden zu müssen.

Der **DB2 Activity Monitor** ist ein Administrations-Tool zur Identifikation, Diagnose und Auflösung von Datenbank- bzw. Anwendungs-Problemen. Es erlaubt einen Einblick in die momentanen Performance-relevanten Aktivitäten durch eine Menge vordefinierter Reports. Die Information wird über Snapshots aus dem Speicher geholt und nicht auf Platte abgelegt. Das DB2 Control Center enthält einen Set Up Activity Monitor Wizard zur Unterstützung der Administratoren bei der Konfiguration des Activity Monitors. Für jeden vordefinierten Report kann der Activity Monitor entsprechende Aktionen zur Auflösung von Problemen in der Ressourcen-Ausnutzung, zur Performance-Optimierung oder zum Aufruf eines anderen Tools zur weitergehenden Untersuchung vorschlagen.

Der zusätzlich erwerbbare **DB2 Performance Expert** ist ein Performance-Monitoring- und Analyse-Tool zur Unterstützung des DBA. Das Produkt bietet einen konsistenten Überblick über alle DB2 Instanzen und Datenbanken über viele Plattformen hinweg. Es sammelt, speichert und aggregiert die Performance- und Workload-Daten in zentraler Stelle in einem Repository, dem sog. *Performance Warehouse*. Das Tool bietet professionelle Analysemöglichkeiten, Online-Überwachung in Echtzeit und eine Vielzahl von Berichten zum Analysieren und Optimieren von DB2-Anwendungen und SQL-Anweisungen. Darüber hinaus kann mit dem integrierten Exception Processing eine weitergehende automatisierte Problemdiagnose und -behebung angestoßen werden.

Analog zu dem DB2 Health Center wird bei Oracle die „Gesundheit“ des Systems durch Metriken gemessen, die mit warnenden und alarmierenden Schwellwerten assoziiert sind. Das Oracle Automatic Workload Repository (**AWR**) sammelt zur Laufzeit automatisch umfangreiche Daten und erstellt Statistiken, die die Grundlage für alle weiteren autonomen (Self-Management-)Funktionen von Oracle bilden sowie zur Erkennung von Problemen dienen. Im Gegensatz zu Oracle besitzt DB2 von Haus aus keine zentrale Ablage, in der regelmäßig Informationen über die Workload gesammelt werden.

Das AWR erstellt regelmäßig sogenannte Snapshots, also Abbilder des aktuellen Zustands der Datenbank, und speichert diese ab. Die gespeicherten Daten werden regelmäßig aufgearbeitet, analysiert und nach einiger Zeit (die Voreinstellung beträgt 7 Tage) gelöscht. Das Intervall, die Vorhaltedauer und im gewissen Grad die Granularität können angepasst werden. Die erfassten Statistiken umfassen u.a. Objektstatistiken, Zeit- sowie Ressourcenverbrauch von DB-Aktivitäten. Zur Reduzierung der hinterlegten Datenmenge werden zwischen zwei Snapshots, soweit möglich, anstelle konkreter Daten deren Deltas gespeichert.

Der Automatic DB Diagnostic Monitor (**ADDM**), der zentrale Ratgeber von Oracle, nutzt den AWR-Output zur Messung und Analyse der Leistung des Datenbanksystems, zur Identifikation von Symptomen und der Analyse von Ursachen von Problemen, zur

Anfertigung von Berichten, zum Aufruf weiterer Ratgeber bei Bedarf, zur Identifikation von Flaschenhälsen und Verbesserungspotential und auf Basis dieser Ergebnisse zur Erstellung einer Empfehlung. Aufgrund der Informationen im AWR ist der ADDM in der Lage, die Performance zu überwachen und für erkannte Probleme Lösungen zu bestimmen. Der ADDM gibt demnach nicht nur Hinweise, sondern macht zusätzlich Verbesserungsvorschläge. Nicht automatisch durchführbare Optimierungsvorschläge werden, wie beim Self-Management üblich, mit einem Interaktionsvorschlag dem Datenbank-Administrator mitgeteilt.

Der ADDM identifiziert unter anderem die Top-SQL-Statements mit einem hohen Ressourcenverbrauch (etwa CPU-Zeit, Pufferzugriffe, Festplattenlesevorgänge usw.) und speichert sie im AWR. Diese TOP-Statements werden vom **Automatic Tuning Optimizer (ATO)** aufgegriffen und an den **SQL Tuning Advisor** weitergegeben, der das sich anschließende SQL-Tuning weitestgehend automatisiert. Der Ratgeber analysiert eine oder mehrere gegebene bzw. identifizierte SQL-Anweisungen, erstellt verbesserte Ausführungspläne oder gibt Tuning-Empfehlungen, wie bspw. die Umformulierung bzw. Restrukturierung einer SQL-Anweisung, die Ergänzung und Aktualisierung der Katalogstatistiken, die Sammlung von zusätzlichen Informationen zur Erstellung eines SQL-Profils oder auch die Suche nach einem geeigneten neuen Index. Die Verbesserungsvorschläge müssen i.d.R. nur noch vom DBA bestätigt werden.

Mit Hilfe des AWR überwacht Oracle sich selbstständig und alarmiert den DBA mittels **Server Generated Alerts** in Fällen, die externe Eingriffe notwendig machen. Ein servergenerierter Alarm kann aus zwei Gründen ausgelöst werden. Entweder durch Eintreten eines definierten Ereignisses, wie bspw. einer bestimmten Fehlermeldung, oder durch Erreichen oder Überschreiten eines Schwellenwerts. Die automatisch erzeugten Benachrichtigungen beinhalten neben der Warnung auch einen Lösungsvorschlag, der dem Admistrator bei der Lösungsfindung behilflich sein soll. Dieser enthält zusätzlich auch Vorschläge für vorbeugende Maßnahmen, die den Fehler zukünftig vermeiden lassen können. Die Alarmmeldungen können über den Enterprise Manager eingesehen oder aber auch dem DBA per E-Mail zugesendet werden. Darüber hinaus ist es möglich, mit einem Alarm ein Skript oder Programm zu verknüpfen, welches gestartet wird, sobald der Alarm ausgelöst wurde.

Zentrale Anlaufstelle Microsofts für den DBA ist das **SQL Server Management Studio**, welches ein einheitliches und übersichtliches User Interface zur Vereinfachung der Tasks anbietet.

Die Monitoring-Tools des SQL Server decken sowohl das Polling von Performance-Indikatoren als auch das Tracing von Events ab. Mögliche Quellen für Performance-Metriken sind die Event Logs, Error Logs, der Windows System Monitor, Activity Monitor, die Dynamic Management Views und der Trace&SQL Profiler. Ähnlich dem Oracle ADDM ermöglicht der **SQL Profiler** ein spezifisches Workload-orientiertes Performance-Monitoring des SQL-Servers. Anhand dieser Daten findet er z.B. langandauernde SQL-Anfragen („Langläufer“). Für die Entwicklungsphase stellt er eine Hilfsumgebung zur Verfügung, mit der man z.B. schrittweise durch die SQL-Statements gehen kann, um sie zu kontrollieren und ggf. Fehler zu suchen.

Zu bemängeln war bisher, dass die Monitoring-Funktionalitäten über mehrere Tools verteilt wurden. So mussten die Tools separat verwendet werden, was eine integrierte Auswertung von Informationen erschwerte. Im Hinblick auf den SQL Server 2008 ist

daher eine wichtige Neuerung zu verzeichnen: das **SQL Server Performance Studio,** welches als integrierte Monitoring-Lösung eingeführt wurde. Letzteres trägt dazu bei, die Sammlung, Analyse und Fehlersuche anhand von SQL-Server-Diagnose-Informationen an zentraler Stelle zu integrieren. Über den Data Collector können Performance-Daten feingranular gesammelt und in einem sog. **Management Data Warehouse** langfristig gespeichert werden. Durch den zentralen Datenspeicher wird es leichter, Daten mittels SQL-Anweisungen und vordefinierter Reporting-Services-Berichte anzuzeigen und zu analysieren.

Für die weitergehende Problem-Erkennung kann der DBA im Windows-eigenen **System Monitor** (bzw. Performance Monitor in Windows-NT-Systemen) basierend auf Objekt-Informationen **Alerts** festlegen, auf die automatisch reagiert werden kann. Zu den Objekt-Informationen, die Alerts auslösen können, gehören Meldungen des SQL Server, Einträge im Windows Application Log und auch bestimmte Performance-Indikator-Werte.

Mittels der SQL Server **Extended-Events**-Infrastruktur wird zudem die Korrelation von Daten aus dem SQL Server sowie unter bestimmten Umständen die Korrelation von Daten aus dem Betriebssystem und aus Datenbank-Anwendungen unterstützt.

Hierbei bedient man sich typischerweise des **Server Agent**, der zum einen selbst Monitoring-Funktionalität besitzt und zum anderen die bekannten Monitoring Tools im Rahmen von Jobs aufrufen kann. Der SQL Server Agent ist der Zeitplandienst/Scheduler innerhalb der Microsoft SQL Server, mit dem mittels Jobs nicht nur die unbeaufsichtigte Ausführung von (einmaligen) Aufgaben, sondern auch regelmäßige Wartungsaufgaben im SQL Server und beliebige wiederkehrende Aufgaben in Windows ermöglicht werden. Dabei können Jobs manuell, durch einen Schedule und insbesondere durch einen Alert angestoßen werden und erlauben damit eine automatische Reaktion auf Fehlerzustände. Diese Jobs ermöglichen es dem Server Agent nunmehr, den Data Collector zu starten, Monitoring-Ergebnisse an Analyse-Tools weiterzuleiten bzw. direkt Gegenmaßnahmen zu identifizierten Problemen einzuleiten. Der integrierte Benachrichtigungsdienst (**Server Notification Services**) bietet darüber hinaus Event-basierte und Zeit-gesteuerte Benachrichtigungen per Mail, Mobiltelefon, PDA oder auch den Windows Messenger.

In [Rei07] und [Aut07] wurde konzeptuell und anhand eines praktischen Versuchs ausführlich gezeigt, wie mit Hilfe der oben genannten, vom SQL Server und Oracle bereitgestellten Komponenten jeweils eine MAPE-Loop konstruiert werden kann, die eine typische DBA-Aufgabe von der Problem-Erkennung bis zur -Auflösung automatisiert. Ziel war es, die durch das DBMS zur Verfügung gestellten Tools geschickt zu verknüpfen und die Möglichkeiten bzw. Grenzen der Tools in Verbindung mit anderen Mechanismen aufzuzeigen. Als eine konkrete, vereinfachte Aufgabenstellung wurde die Ereignis-basierte, automatische Vergrößerung von Datendateien gewählt. Es galt ein Event, welches einen hohen Füllgrad einer Datendatei signalisiert, abzufangen und mit einer Aktion zu koppeln, die wiederum die betroffene Datendatei vergrößert, ohne dass der DBA selbst eingreifen musste. Wie die in DB2 vorhandenen Monitoring- und Analyse-Tools zur Problem-Erkennung und -Auflösung verknüpft werden können, zeigen die nachfolgenden Hauptkapitel.

3.1.5 System- und Ausfallsicherheit

Eine der wichtigsten Anforderungen an Datenbanksysteme ist die Verfügbarkeit. Im Zuge der Globalisierung und internationalen Vernetzung rückt diese Forderung immer mehr in den Vordergrund der Endnutzer, so dass ein heutiges DBMS den Ansprüchen einer

ständigen Hochverfügbarkeit 365 Tage im Jahr und 24 Stunden am Tag genügen muss. Nicht nur das DBMS und die in den verwalteten Datenbanken enthaltenen Daten, sondern auch die Hardware, Netzwerkressourcen, vor allem aber die Applikationen müssen beinahe dauerhaft betriebsbereit und für die Nutzer verfügbar sein. Konkret bedeutet das, auf geplante und ungeplante Ausfälle vorbereitet zu sein.

Zu den geplanten Ausfällen gehören Maintenance-Aktivitäten, wie Recovery (Backup/Restore), Tabellen-Reorganisationen, Schema-Evolution oder auch Änderung von Konfigurationsparametern, die die Verfügbarkeit des Systems beeinträchtigen können. Das Ziel dabei ist die Ausführung dieser Aufgaben im laufenden Betrieb mit minimaler Ressourcen-Beeinträchtigung und minimalem Sperr-Aufkommen. Ungeplante Ausfälle sind meist in Naturkatastrophen, Sabotage-Akten, Hardware-Versagen (Festplatte, CPU), oder in anderen Infrastruktur-bedingten- bzw. User- oder Applikations-Fehlern begründet. Das erklärte Ziel ist, wie zu erwarten, die Reduzierung der „Downtime“ im Falle eines solchen ungeplanten Ausfalls.

Neben der Ausfallsicherheit spielt auch die Systemsicherheit eine zentrale Rolle für die Aufrechterhaltung der Verfügbarkeit eines DBMS. Ziel des Selbst-Schutzes ist das Abschirmen des DBMS vor feindlichen bzw. fehlgeleiteten Anfragen, welche die Performance negativ beeinflussen oder gar das ganze System zum Absturz bringen können. Darunter fallen Maßnahmen zum Erstellen, Implementieren und Überwachen von Datenbank- und DBMS-Sicherheitskonzepten, inklusive Verwalten von Berechtigungen der Programme sowie der Benutzer für die verschiedenen SQL-Befehle (wie Select, Insert, Update) auf die Datenbank-Objekte.

Die Ausgestaltung der Schutzmechanismen sieht bei den Datenbank-Management-Systemen folgendermaßen aus: Über ein geeignetes Authentifikations-System wird sichergestellt, dass ausschließlich berechtigte Nutzer Zugang zu dem System mit ihren Daten erhalten. Eine feingranulare Rechtevergabe beschränkt jeden Nutzer auf die für ihn freigegebenen Daten und die Operationen, die er darauf ausführen kann (Abfrage-, Einfüge-, Aktualisierungs- und Löschoperationen). Dieser Datenschutz ist notwendig, da die Nutzer teilweise aus unternehmerischen Interessen nicht auf alles zugreifen dürfen sollen, aber dies zum Teil aus gesetzlichen Gründen auch gar nicht dürfen. Daneben dient dies auch dem Schutz des Systems. Die Nutzer, die ausschließlich lesend zugreifen, können auch keine ungewollten Schäden anrichten. Der Schutz vor unbefugtem Zugriff auf Daten kann typischerweise auf Tabellen-, Zeilen- oder Spaltenebene oder auch wertabhängig für bestimmte Datensätze eingerichtet werden. Oracle erlaubt hierbei als einziges DBMS eine feingranulare, Tupel-basierte Autorisierung (row-level security).

Alle DBMS bieten zusätzlich Möglichkeiten der Zugangskontrolle (Admission& Application Control), zum „Schutz“ vor Anfragen, Nutzern oder Applikationen, die sich negativ auf Performance auswirken oder unerwünscht System-Ressourcen verbrauchen und damit die Performance beeinträchtigen.

Darüber hinaus erlauben manche DBMS auch die Definition zusätzlicher Sicherheitsmaßnahmen gegen unbefugten Zugriff, wie bspw. die Erzwingung physischen Löschens bei logischem Löschen oder das Verschlüsseln (Data Encryption) der abgespeicherten Daten (Primärdaten, Protokolle etc.) auf dem Speichermedium bzw. zum Schutz der Daten bei ihrer Übertragung zwischen verschiedenen Systemen.

In einigen Anwendungs-Szenarien ist es zudem wichtig, nachvollziehen zu können, wer eine bestimmte Änderung durchgeführt hat oder welche Änderungen ein spezifischer Benutzer verursacht hat. Dazu stehen dem DBA Auditing-Werkzeuge (insbesondere

diverse Logs) zur Verfügung, die die DB-Aktivitäten verfolgen. Das DBMS kann diese Informationen schließlich nutzen, um Trends zu erkennen, potentielle Gefahren zu analysieren, zukünftige Sicherheitsmaßnahmen zu planen und die Effektivität eingeleiteter Gegenmaßnahmen zu erkennen.

3.1.6 Abschließender Vergleich

Heterogenität von Hard- und Software in heutigen Systemen und die Verteilung verkomplizieren fraglos die Datenbank-Administration. Nimmt man noch die ständig wachsenden Verantwortungsbereiche der DBA hinzu, ist der Bedarf an unterstützenden Werkzeugen zur Durchführung der Datenbankadministration offensichtlich.

In Abschnitt 2.2 wurde bereits auf die Ähnlichkeit von Tuning-Kreislauf und MAPE-Loop hingewiesen. Eine Adaption bekannter Tuning-Verfahren auf Mechanismen des Autonomic Computing ist dementsprechend theoretisch durchführbar und wird auch in den Produkten praktisch umgesetzt. Grundsätzlich sollte unterschieden werden, ob der Administrator vorerst nur unterstützt wird oder ob Aufgaben völlig autonom ablaufen sollen. Prinzipiell ist es ratsam, solche Tuning-Aufgaben zu automatisieren, bei denen, entweder (1) regelmäßig oder in Reaktion auf ein bestimmtes Ereignis, immer die gleichen Maßnahmen erfolgen. Sollte z.B. ein Systemfehler auftreten und einen DBMS-Neustart erforderlich machen, wird prinzipiell eine Recovery durchgeführt. In allen aktuellen DBMS wird diese Aufgabe deswegen auch autonom ausgeführt. Ähnlich verhält es sich mit (2) Aufgaben, welche eine kontinuierliche Online-Optimierung erfordern. Für DBAs wäre es nicht im Entferntesten möglich, alle an das DBMS gestellten Anfragen zu optimieren oder die Größen der diversen Caches eines DBMS kontinuierlich an die Workload anzupassen. Abgesehen von Ausnahmefällen, in denen immer dieselbe Workload bearbeitet wird, kann diese Optimierung nur autonom oder sehr eingeschränkt erfolgen. Eine Unterstützung des DBA, im Sinne einer ausgesprochenen Handlungsempfehlung, kommt wegen der Anzahl der erforderlichen Anpassungen und dem Bedarf einer zeitnahen Bearbeitung nicht in Frage. Des Weiteren eignen sich (3) Aufgabenstellungen, welche vollständig numerisch gelöst werden können. Die vom DBA und vom autonomen Verfahren ermittelten Ergebnisse sind im Idealfall identisch, jedoch liegen die autonom berechneten schneller vor und der DBA wird entlastet.

Obwohl Anbieter, wie IBM, Oracle oder Microsoft in dieser Hinsicht dieselben Ziele verfolgen und fast gleiche Wege gehen, klassifizieren sie ihre Werkzeuge unterschiedlich. Die implementierten neuen Funktionen verbergen sich hinter einer Reihe von Schlagworten, die oftmals Marketing-geprägt sind. Es ist daher verständlich, dass bei Anwendern eher Begriffsverwirrung als Klarheit über den realen Funktionsumfang entsteht. Außerdem legen die Hersteller den Begriff „autonomic“ großzügig aus und fassen auch Werkzeuge darunter, die durchaus noch manuelle Vor- oder Nacharbeiten benötigen.

Funktion\DBMS	IBM DB2 v9	Oracle 11g	MS SQL Server 2008
(initiale) Konfiguration	▪ Configuration Advisor	▪ Database Configuration Assistant	▪ Configuration Manager
Physischer Entwurf	▪ Design Advisor (Indexe, MQTs, Partitionierung, MDCs)	▪ SQL Tuning & Access Advisor (Indexe, mat. Sichten)	▪ Database Tuning Advisor (Indexe, mat. Sichten, Partitionierung) ▪ Physical Design Alerter

Tabelle 3.1: Statische Optimierung des physischen DB-Designs und der Konfiguration

Das Problem der **Selbst-Konfiguration** kann in zwei Teilaspekte heruntergebrochen werden. Das System sollte zum einen nach der Installation, ohne weiteres Eingreifen des Nutzers, sinnvoll konfiguriert sein und somit eine akzeptable Performance bieten. Zur Laufzeit soll sich das System zum anderen autonom einer verändernden Umwelt, Last etc. anpassen, um möglichst optimale Leistung zu erbringen. Das generelle Ziel ist dabei nicht die automatische Konfiguration eines einzelnen Systems, sondern vielmehr die einer komplexen Infrastruktur mit einer Vielzahl von Teilsystemen. Die wesentlichen Aspekte einer DBMS-Konfiguration beinhalten Parameter, Schwellenwerte für den Verbrauch von Ressourcen sowie Indexe und andere physische Datenstrukturen. **Tabelle 3.1** fasst die wesentlichen Ergebnisse der statischen Initialkonfiguration sowie die Möglichkeiten zum Automatisieren des physischen Datenbank-Entwurfs der drei betrachteten Hersteller kategorisiert zusammen. Die ersten beiden Zeilen in Tabelle 3.2 zeigen ferner die Techniken für die dynamische Verwaltung und online-Rekonfiguration von Festplatten- und Hauptspeicher.

Funktion\DBMS	IBM DB2 v9	Oracle 11g	MS SQL Server 2008
Memory Management	▪ Self Tuning Memory	▪ Automatic Shared Memory Management	▪ dynamische Verteilung zwischen Min / Max Server Memory
Storage Management	▪ Automatic Storage	▪ Automatic Storage Management	▪ Auto Grow & Auto Shrink
Workload Management	▪ Workload Manager (Governor & Query Patroller)	▪ Database Resource Manager	▪ Resource Governor
Maintenance (Backup, Reorg, Load)	▪ Autonomic Object Maintenance ▪ Utility Throttling ▪ Online Index Reorganization ▪ Self-tuning Backup/Restore	▪ Autonomic Maintenance Task Infrastructure ▪ Online Index Reorganization	▪ Maintenance Plans ▪ Online Index Reorganization
Statistiken	▪ Automatic Statistics Collection (ASC) ▪ LEO & POP ▪ Real-Time Statisitcs ▪ Sampling	▪ Automatic Optimizer Statistics Collection ▪ Dynamic Sampling	▪ Automatic Statistics Management ▪ Sampling

Tabelle 3.2: Dynamische DBMS-interne Systemverwaltung

Bei heutigen Datenbank-Management-Systemen liegt der Fokus der Autonomie-Bestrebungen auf der **Selbst-Optimierung**. Sie erlaubt dem DBMS, jede Aufgabe und jedes Utility auf möglichst effiziente Weise auszuführen, abhängig von der gegenwärtigen Workload, den verfügbaren Ressourcen und den Umgebungs-Einstellungen. Übergeordnetes Ziel ist es dabei, die Gesamtleistung des Systems zu optimieren. **Tabelle 3.2** gibt einen Überblick über die wichtigsten eingesetzten Methodiken zur (internen) Optimierung des Systems und seiner Ressourcen.

Insbesondere der Teilbereich des SQL-Tuning bzw. der Optimierung der Query-Ausführung wird im Allgemeinen als derjenige mit großem Verbesserungspotential angesehen [EPBM03]. Ansätze für die Optimierungen finden sich dabei bspw. in der Suche

nach verbesserten Ausführungsplänen, effizienteren, umformulierten (aber semantisch äquivalenten) SQL-Abfragen und weiteren dynamischen Laufzeit-Optimierungen. Zur Durchführung dieser Aufgaben ist es vor allem notwendig, dass aktuelle und zuverlässige Statistiken vorliegen, da diese zusammen mit der vorliegenden Hardware (z.B. Anzahl und Geschwindigkeit an CPUs, Parameter der Speichermedien etc.) und der aktuellen Konfiguration (Größen der Speicher-Konsumenten wie Bufferpool, Sort Heaps, Caches etc.) die Basis für jegliche Optimierung bilden. Die Umgebungsbedingungen eines DBMS ändern sich fortwährend, so dass es immer Potential für Optimierungen gibt. Das System muss sich an veränderte und nicht vorhersehbare Umweltzustände, wie z.B. eine steigende Arbeitslast, schnellstmöglich anpassen und sollte ständig nach Wegen suchen, die Gesamt-Performance zu verbessern. Sobald die Notwendigkeit erkannt wird, sollten der aktuelle Status und die Umgebung evaluiert und darauf basierend notwendige Aktionen durchführt werden.

Eine elementare Anforderung an Datenbank-Management-Systeme ist die Erfüllung der ACID-Eigenschaften (Atomicity, Consistency, Isolation, Durability, siehe [HäRa01]). Diese Eigenschaften sollen sicherstellen, dass die Datenbankintegrität jederzeit gewährleistet ist. Insbesondere im und nach dem Fehlerfall, bspw. Strom- oder Festspeicherausfall, muss die Garantie der Eigenschaften gelten. Im Zusammenhang mit der **Selbst-Heilung** geht es dabei vor allem um den Aspekt der Konsistenz [EPBM03]. Dies bedeutet, dass sich die Datenbank stets in einem konsistenten Zustand befindet oder zu jeder Zeit wieder in einen (rück-)überführt werden kann. Dazu ist es notwendig, geeignete Protokollierungs- (Logging), Sicherungs- (Backup) und Wiederherstellungsstrategien (Recovery) auszuwählen und im Fehlerfall anzuwenden. Die in den DBMS eingesetzten Strategien zur autonomen bzw. automatisierten Wartung finden sich ebenfalls in Tabelle 3.2.

Ziel ist es, dass ein autonomes System einen Fehler möglichst schnell bemerkt, die betroffene(n) Komponente(n) identifiziert und das Problem soweit möglich von sich aus und ohne äußeren Eingriff löst. So soll ein autonomes DBMS (ADBMS) u.a. selbst erkennen, wann ein vollständiges oder inkrementelles Backup notwendig ist, um im Anschluss die notwendigen Operationen mit geringer Betriebsstörung auszuführen. Dies verlangt jedoch, dass in den SLAs bspw. Recovery-Zeiten vorliegen. Im Falle von Hardware-Versagen oder anderen Systemfehlern muss ein ADBMS das letzte Backup hernehmen und in einen konsistenten Zustand möglichst nahe vor dem Fehlerfall zurückkehren, um mit den pausierten Operationen nach der Fehlerbehandlung fortzufahren. Alle betrachteten DBMS unterstützen Logging-, Backup- und Recovery-Mechanismen sowie eine automatisch initiierte, mittels der Transaktions-Log-Dateien durchführbare Crash Recovery für Fehler, die nicht auf Katastrophen basieren. Die auftretenden Probleme können allerdings auch anderer Natur sein. Kommt es durch ein Konfigurationsproblem etwa zu einer Speicherknappheit, sollte auch hier das System korrigierend eingreifen, sodass ein reibungsloser und vor allem performanter Ablauf der Aufgabe gewährleistet ist. Hier besteht auch durchaus ein Abgrenzungsproblem zu Selbstkonfiguration und -optimierung.

Funktion\DBMS	IBM DB2 v9	Oracle 11g	MS SQL Server 2008
Datensammlung	▪ System Monitor (Snapshots, Events)	▪ Automatic Workload Repository (AWR)	▪ Data Collector ▪ Monitoring Tools (Event Logs, System Monitor, Dynamic Management Views, Trace & SQL Profiler, Server Agent)

Problem-Erkennung	▪ Health Center	▪ Automatic DB Diagnostic Monitor (ADDM) ▪ Server Generated Alerts	▪ System Monitor ▪ Server Agent
Problem-Auflösung	▪ Recommendation Advisor ▪ Task Center (Definition von Tasks und Schedules) ▪ Health Center (Verknüpfung von Scripts und Tasks) ▪ DB2 Scheduler System	▪ ADDM ▪ Enterprise Manager (Definition und Verknüpfung von Jobs) ▪ Oracle Scheduler ▪ Advisor Framework	▪ SQL Server Management Studio -> Server Agent (Definition und Verknüpfung von Jobs) ▪ Server Agent Schedules

Tabelle 3.3: DBMS-eigene Tools zum Performance Management

Eine der wichtigsten Grundvoraussetzungen für die selbstständige Optimierung, aber maßgeblich für das Performance Management, ist das Prinzip der **Selbst-Inspektion**. Dabei handelt es sich um das im Abschnitt 2.2.2 eingeführte Charakteristikum der Selbstkenntnis des Systems. Als Basis der anderen Eigenschaften ist es notwendig, dass das Datenbank-Management-System sich seiner selbst bewusst ist. Hierbei sollte das System relevante Informationen über seine Komponenten, deren Zustand und Performance sowie über die Workload sammeln, speichern und auswerten. Nur dann sind intelligente Entscheidungen der autonomen Mechanismen möglich und das DBMS kann sich effektiv selbst kontrollieren. Diese Informationen können bspw. verwendet werden, um die Performance zu optimieren, mögliche Probleme zu identifizieren, Statistiken der gespeicherten Daten zu aktualisieren und deren Integrität zu überprüfen, Wartungen zu planen und eventuelle Trends bezüglich der Workload zu erkennen. Die Erkenntnisse daraus können sowohl dem Administrator präsentiert, als auch als Input für andere Komponenten oder Operationen verwendet werden.

Tabelle 3.3 umfasst die gängigsten Tools, zum größten Teil bereits in den vorangegangenen Abschnitten skizzierten, zur Datensammlung und darauf basierender Problem-Erkennung und -Auflösung. Die gesammelten und zumeist kurzfristig gespeicherten Daten werden auf kritische Zustände untersucht und es können automatisch korrigierende Auflösungs-Aktionen vorgeschlagen bzw. ausgeführt werden.

Zusammenfassend lässt sich feststellen, dass besonders solche Tuning-Aufgaben für eine autonome Umsetzung geeignet sind, welche hinreichend formalisiert werden können. Ist dies nicht gegeben und die vollständige Semantik bleibt dem System vorenthalten, ist stark abzuwägen, ob der Einsatz von Automatismen bzw. autonomen Mechanismen gerechtfertigt ist. Immerhin besteht die Gefahr einer Fehlentscheidung. Beispielhaft sei die Optimierung des physischen Datenbank-Entwurfs genannt. So könnten physische Design-Strukturen seit geraumer Zeit nicht mehr verwendet worden sein und dementsprechend automatisch gelöscht werden. Dem DBA ist aber bekannt, dass diese Strukturen in Kürze wieder verwendet werden. Er hätte sich gegen eine Löschung ausgesprochen, zumal ein „Ad-hoc-Wiederanlegen“ solcher Strukturen sehr aufwendig sein kann (z.B. Indexierung großer Tabellen). In solchen Fällen ist vorerst die Unterstützung der DBAs (in Form von Vorschlägen) vorzuziehen.

3.1.7 Resümee und Ausblick

Die drei Marktführer im Bereich Datenbanken haben erste große Schritte zur Selbst-Verwaltung ihrer Produkte getan [Aut07, EPBM03, Rei07, RMH+09, Sat07]. Das Spek-

trum der Selbst-Verwaltung ist bestimmt durch das Zeitfenster für die Entscheidung und den Grad der Integration in das DBMS. Funktionalitäten reichen von tief in den Kern integrierten, selbst-optimierenden Algorithmen zum Prefetching und zur Seitenersetzung, über die regelmäßige Verwaltung von Statistiken, das bedarfsgesteuerte und vom System angestoßene Memory Management, bis hin zu längerfristigen, weitaus weniger tief eingebetteten Aktivitäten, wie dem physischen Entwurf sowie der Installation und Konfiguration [Sat07].

Der ansteigende Grad an Selbst-Verwaltung ermöglicht ein frühzeitiges (automatisches) Aktivwerden im Falle von bzw. idealerweise sogar das Vermeiden von Problemen, ohne dass man zunächst auf Beschwerden seitens der Anwender warten muss. Viele der Werkzeuge und Mechanismen sind bezüglich des Autonomic Computing Maturity Model (siehe Abschnitt 2.2.3) jedoch erst in etwa auf dem Predictive Level einzuordnen, da sie nicht vollautomatisch arbeiten und noch durchaus eine Menge manueller Vor- oder Nacharbeiten sowie Interaktionen mit dem DBA benötigen [RMH+09].

Um eine breite Nutzbarkeit zu gewährleisten, sollten diese Tools über eine einheitliche, zentrale und integrierte Schnittstelle zugänglich sein, dem DBA bei der Entscheidungsfindung unterstützen und dabei eine möglichst geringe Menge an Input verlangen. Sofern dem Nutzer die Komplexität nicht verborgen wird, ist eine komplexe Umgebung diffiziler zu nutzen als eine simple. Alle drei DBMS bieten dem DBA daher eine integrierte GUI zum Zugriff auf alle verfügbaren Tools, einen zentralen Zugriffspunkt für Metadaten sowie viele Wizards. Vornehmlich sind dabei der Oracle Enterprise Manager, das MS SQL Server Management Studio und das DB2 Control Center zu nennen.

Von adaptiven oder autonomen Systemen kann man somit zu diesem Zeitpunkt noch nicht sprechen. Dennoch kann die Arbeitserleichterung für DBAs teilweise erheblich sein, insbesondere durch Tools und Techniken, die sich den Aufgaben der automatischen Speicherverwaltung, des automatisierten physischen Datenbankentwurfs sowie der Sammlung/Aktualisierung und Verwaltung von Statistiken stellen und damit primär die Bereiche der Selbst-Optimierung und Selbst-Konfiguration abdecken.

In [Rei07] wurde anhand umfangreicher Untersuchungen die folgende Bewertung getroffen: „Obwohl das Memory Management alles andere als trivial ist, gelingt es dem SQL Server 2005 den DBA weitreichend von dessen Betreuung zu entbinden“. In der Arbeit von [Aut07] wird demonstriert, wieviele autonome Ansätze bereits im Rahmen der Oracle Selbstverwaltungs-Infrastruktur (Intelligent Self-Management-Framework) realisiert und tauglich sind um dem Ziel einer selbstverwalteten Datenbank näher zu kommen. Auch IBM hat in DB2 bereits frühzeitig ausgereifte autonome Mechanismen zur Speicherverwaltung integriert [Ahu06] [SaPr07].

Obwohl bis jetzt bereits große Fortschritte in oben genannten Bereichen erzielt wurden, stehen spezielle Aspekte des Storage Management, der Optimierung des physischen Designs und insbesondere des automatischen SQL-Tunings derzeit eher am Beginn größerer Entwicklungen.

Die dynamische Änderung von System-Konfigurationsparametern bspw. kann Ausfallzeiten erzeugen, da einige Änderungen erst durch Stoppen und Neustarten der Instanz bzw. Datenbank effektiv werden. Die Hersteller der im Blickfeld befindlichen DBMS sind jedoch bestrebt, immer mehr Änderungen im laufenden DB-Betrieb möglich werden zu lassen (Stichwort „Online Everything“).

Die Ratgeber für das physische Design funktionieren sehr gut für Indexe. Die Unterstützung materialisierter Sichten und der Partitionierung ist jedoch oft unzureichend.

Zudem werden, den DBMS eigene, spezielle Lösungen (Bitmap-Index, mehrdimensionale Indexe, ...), wenn überhaupt, nur eingeschränkt behandelt. Zudem sind die Ratgeber vornehmlich statischer Natur und augenscheinlich nützlich nur zu einem Zeitpunkt. Sie passen sich nicht den ständig ändernden Bedingungen an. Der DBA muss immer noch selbst das System überwachen und feststellen, wann eine Anpassung der Konfiguration bzw. des physischen Datenbankdesigns sinnvoll und nötig ist. Das ideale autonome DBMS sammelt ständig Performance-Metriken (mit geringem Overhead) und bestimmt, wann welche Ressourcen adjustiert werden müssen, um die Performance beizubehalten bzw. zu verbessern.

In Oracle Database existiert bereits eine Vielzahl an Ratgebern, die sich, ähnlich wie die Ratgeber der anderen Hersteller, bisher allerdings darauf beschränken, Empfehlungen zu geben, ohne sie automatisch umzusetzen. An dieser Stelle wäre es durchaus überlegenswert, den nächsten Schritt zu gehen und die Empfehlungen der Ratgeber automatisch zu implementieren.

Grundsätzlich sind die derzeitigen Tools noch sehr auf die Eingaben, Entscheidungen und damit auch auf die Intelligenz des DBA angewiesen. Menschliche Eingaben sind jedoch fehleranfällig und nicht immer zuverlässig und zeitnah möglich, gerade im Hinblick auf die sich ständig ändernde System-Umgebung und die dynamische Workload. Von daher wünschte man sich intelligentere Wartungs-Tools, die in Abhängigkeit von der Workload und dem Systemzustand vorhersagen, wann die beste Zeit zum Ausführen ist, wie lange die Ausführung dauern wird und sich schließlich selbst aufrufen. Das Utility-Throttling von IBM ist ein erster vielversprechender Schritt in diese Richtung.

Die Schnittstellen für die Aufgaben der Systemadministration sind von DBMS zu DBMS extrem verschieden. Ein DBMS-übergreifendes Systemadministrations- und Monitoring-werkzeug sowie standardisierte Schnittstellen sind daher gerade in Umgebungen, in denen mehrere verschiedenen DBMS zu administrieren sind (oder gar bei föderierten Datenbank-systemen), wünschenswert. Für Administrationswerkzeuge sind allerdings noch weitere Anforderungen relevant, die zu den vorgestellten Aspekten von Datenbankadministration orthogonal sind. So erscheint z.B. die Administration einer verteilten Umgebung und eines gesamten Software-Stack von einer einzelnen Stelle aus (Stichworte wie Remote Administration bzw. Single Point of Control) wichtig.

Auffällig ist insbesondere bei Microsoft die lose bis nicht vorhandene Kopplung verschiedener Tools, vor allem im Bereich des Performance Management. Autonome Datenbank-Management-Systeme müssen die Daten, die sie sammeln, geeignet analysieren können. Hier erscheint es erstrebenswert, die Monitoring-Vielfalt und -Funktionalität ohne Beeinflussung der Leistung des Systems auszubauen, die verschiedenen Ergebnisse stärker untereinander auszutauschen, die Tools zusammen wirken zu lassen bzw. unter einer integrierten Einheitslösung zu vereinen und damit komplexere, übergreifende Aufgaben sowie Analysen zu erlauben. Hierfür ist eine adäquate Modellierung des Systems an sich notwendig. Nur so ist sichergestellt, dass sich das System selbst, seine interagierenden Komponenten und die Umgebung überhaupt angemessen kennen und den eigenen Zustand, auch im Kontext der Gesamtumwelt, richtig einschätzen sowie über akkuratere analytische Vorhersagemodelle verfügen kann.

Letztlich erscheinen auch intelligentere Mechanismen zur Workload-Erkennung und -Vorhersage für ein dynamischeres, Workload-orientiertes Tuning mit einer erweiterten Problem-Vorhersage und -Vermeidung unter Erkennung und Nutzung spezifischer Workload-Charakteristika (Intensität, Trend, Eigenschaften etc.) sinnvoll.

Eine idealisierte DBA-freie Verwaltung verfügt über intelligente, selbst-entscheidende Tools, welche die Komplexität nach außen reduzieren und maskieren. Getreu dem Motto des Autonomic Computing kann an diesem Punkt eine Verlagerung der menschlichen Aufgaben hin zur strategischen Langzeitplanung unter Vorgabe der Tuning-Ziele und einzuhaltender Richtlinien erfolgen. DBAs werden demnach nicht ersetzt, sie können sich durch den Wegfall wiederkehrender Routineaufgaben mit mehr Zeit anspruchsvolleren Aufgaben oder auch der Überprüfung der Tuning-Vorschläge des Systems widmen. Vor allem bei „schwerem Geschütz", also Änderungen, die nicht oder nur mit sehr großem Aufwand rückgängig gemacht werden oder gar schwerwiegende Folgen besitzen können (z.B. Index-Erzeugungen oder Reorganisations-Operationen), sollte das System die Entscheidung den Administratoren überlassen.

Auf dem langen, evolutionären Weg hin zur vollständigen Autonomie erscheint somit die Symbiose von Tools, autonomen Mechanismen und dem DBA am sinnvollsten. Dennoch wäre es naiv zu glauben, dass Performance-Optimierung durch ein autonomes DBMS überflüssig wird. Ein ungünstiges Datenmodell, fehlende Indexe, extrem umständliche Abfragen und natürlich auch eine falsche Hardware-Konfiguration kann keine Software automatisch korrigieren. Zusammenfassend können Datenbank-Management-Systeme derzeit zu den am weitesten entwickelten autonomen Systemen gezählt werden.

3.2 Etablierte Methodiken zum autonomen Tuning

Es gibt eine Vielzahl an Möglichkeiten, ein Software-System sich selbst verwalten zu lassen. Grundsätzlich kann man nach [Sul03] zwischen den Techniken der (1) empirischen Vergleiche, der (2) analytischen Modelle und der (3) Feedback-Schleifen unterscheiden. Die sehr unterschiedlichen Schwerpunkte und Zielsetzungen der einzelnen Ansätze lassen eine hohe Breite und Vielfalt der Thematik erahnen und sollen durch konkrete Prototypen, Forschungs- und Industriearbeiten mit dem Fokus auf dem autonomen Tuning von Datenbank- bzw. Betriebssystemen im Überblick veranschaulicht werden.

3.2.1 Empirische Vergleiche

Eine der Möglichkeiten, die optimale Konfiguration (der Ressourcen des Datenbanksystems) für eine gegebene Workload zu ermitteln, ist der empirische Vergleich der Performance einiger oder aller möglichen Konfigurationen. Es ist also prinzipiell möglich, in endlicher Zeit alle Konfigurationen durchzuprobieren, die jeweilige Performance zu messen und im gemeinsamen Vergleich die optimale Belegung für eine bestimmte Workload zu ermitteln. Verkürzt, aber auch in der Genauigkeit verringert werden kann dieses „Trial-and-Error"-Verfahren durch den Einsatz von Simulationen. Im Falle einer großen Anzahl an Stellschrauben und Kombinationsmöglichkeiten ist dieser erschöpfende Ansatz alles andere als wirtschaftlich und wird, unabhängig von der verwendeten Suchmethodik, eine unzumutbare Zeitdauer beanspruchen.

Wenn jedoch die auf dem System erwarteten Workloads sehr leicht charakterisierbar und im Voraus bekannt sind, ist es möglich, die empirischen Vergleiche einmalig vorab durchzuführen und für jede unterscheidbare Workload die optimale Ressourcenkonfiguration zu bestimmen. Da in der gängigen Praxis jedoch die künftige Workload eher unbekannt ist und sich dynamisch ändert, ist ein neuer aufwendiger Vergleichsdurchlauf zur Laufzeit unabdingbar. Es kann sogar notwendig sein, periodisch neue Vergleiche für bereits untersuchte Workloads durchzuführen um auf andere veränderte Umgebungsbedingungen zu reagieren.

Daher ist der wesentliche limitierende Faktor von auf empirischen Vergleichen basierenden Tuning-Ansätzen die Schwierigkeit, eine angemessene Konfiguration in adäquater zu Zeit ermitteln, insbesondere bei häufigen Schwankungen in der Workload. Zudem können die Ansätze nicht aus der Erfahrung generalisieren, da die Ergebnisse eines vergleichenden Durchlaufs lediglich auf eine bestimmte Workload anwendbar sind.

Reiner und Pinkerton [Rei81] präsentieren einen auf empirischen Vergleichen beruhenden Ansatz zur dynamischen Adaption der Stellschrauben in einem Betriebssystem. Sie führen die Vergleiche während einer speziellen Experimentierphase durch. Wann immer sich der Zustand des Systems signifikant ändert, wird eine der Stellschrauben-Kombinationen zufällig als Konfiguration für das System ausgewählt und die resultierende Performance gemessen. Hält ein Systemzustand lange genug an, werden mehrere Konfigurationen probiert. Mit der Zeit sammeln sie dadurch genug Daten, um die optimalen Konfigurationen für jeden möglichen Systemzustand zu ermitteln und können die Experimentierphase beenden. Darauf folgend können die Stellschrauben entsprechend der ermittelten optimalen Werte für jeden Systemzustand angepasst werden. Gelegentliche ergänzende Experimente erlauben die Anpassung an veränderte Umgebungsbedingungen. Die Autoren haben gezeigt, dass das System bis zu einem gewissen Grad performant ist, ließen allerdings offen, welche zusätzliche Last die Experimentierphase auf dem System erzeugt.

Seltzer und Small [Sel97] schlagen einen Ansatz zur Konstruktion eines seine Performance überwachenden und sich an die veränderte Workload anpassenden Betriebssystem-Kernels vor. Ihre Vergleiche zielen darauf ab, die optimale Cache-Seitenersetzungsstrategie für eine bestimmte Workload zu ermitteln. Sie machen sich jedoch eine spezielle Eigenschaft des erweiterbaren Betriebssystems zu Nutze, die es erlaubt, bestimmte Module im Kernel zur Laufzeit auszutauschen. Auf diese Art und Weise ist es möglich, dass der Kernel mit seinen Standard-Strategien weiteroperiert, aber spezielle austauschbare Simulationsmodule mögliche Alternativen mit der realen Workload erforschen. Dadurch werden die potentiellen Seiten-Effekte auf das Laufzeitverhalten des Systems reduziert. Jedoch bieten die meisten Betriebssysteme nicht die notwendige Erweiterbarkeit und Modularität für diese spezifische Art der Simulation.

Auch **Feitelson** und Naaman [FeNa99] diskutieren eine empirische Methodik zur Konstruktion sich selbst optimierender Betriebssysteme. Anstatt exhaustiv alle möglichen Konfigurationen miteinander zu vergleichen, nutzen sie genetische Algorithmen für die Suche nach guten Kandidaten, während sich das Sytem im Ruhezustand befindet. In jeder Vergleichsrunde werden diejenigen Konfigurationen ausgewählt, die einem bestimmten Gütekriterium am besten entsprechen. Deren Eigenschaften werden dann leicht verändert und miteinander kombiniert, um eine neue Population von Lösungskandidaten zu erzeugen. Dieser Vorgang wird viele Male bis zu einer Konvergenz wiederholt. Konfigurationen, die eine gute Performance in einer Simulationsrunde ergeben, erreichen diese erwartungsgemäß in ggf. „mutierter“ Form auch in der nächsten Runde. Die Autoren zeigen die Effizienz ihres Ansatzes lediglich anhand des Tunings eines parametrisierbaren Scheduling-Algorithmus, der in Isolation von dem Rest des Betriebssystems offline evaluiert werden kann. Sie gestehen ein, dass es schwierig wird, gute Simulationen anderer Aspekte des Betriebssystems unter Berücksichtigung eines geringen Overheads durchzuführen.

Wie bereits angedeutet, können auf empirischen Vergleichen beruhende Tuning-Ansätze ihre Erfahrungen nicht zur Verallgemeinerung nutzen. Das Auftreten neuer Workload macht daher ein zusätzliches Training erforderlich, welches weder bei den Feedback- noch

bei den Modell-basierten Mechanismen (siehe nachfolgende Erläuterungen) notwendig ist. Seltzer und Small eliminieren die Notwendigkeit dieses zusätzlichen Aufwands durch spezielle, jedoch auf den meisten Systemen nicht verfügbare Simulations-Module. Als Konsequenz führen die empirischen Ansätze bei dynamischen Änderungen in der Workload in der Regel zu einer Performance-Degradation, solange sie das zusätzliche Training nicht offline oder zu lastarmen Zeiten durchführen. In jedem Fall führt der Zusatzaufwand jedoch zur Unfähigkeit, zeitnahe Tuning-Maßnahmen für bisher unbekannte Workloads durchzuführen.

3.2.2 Analytische Modelle

Mithilfe mathematischer Modelle kann die System-Performance für mögliche Kombinationen von Workload-Charakteristika und Stellschrauben-Einstellungen vorhergesagt werden. Insbesondere kann ein solches Modell dazu verwendet werden, bei einer bestimmten Workload die Konfiguration zu bestimmen, die die erwartete Performance unter dieser Workload maximiert. Analog zu empirischen Verfahren wird hier die Performance verschiedener Konfigurations-Szenarien bei gegebener Workload verglichen, jedoch basiert diese auf den Vorhersagen des Modells und nicht auf tatsächlichen bzw. simulierten Performance-Metriken. Aber auch hier können bei einem ausreichend großen Raum an möglichen Konfigurationen spezielle Such-Techniken angebrachter sein als die langwierige exhaustive Suche.

Es gibt eine Vielzahl an gängigen Typen von Modellen, wie beispielsweise statistische Modelle, die auf Regression basieren oder auch graphische, probabilistische Modelle. Jeder Typ hat eine Menge assoziierter Parameter, die typischerweise aus einer Menge an Trainingsdaten erlernt werden. Die Trainingsdaten umfassen Statistiken über die Workload-Charakteristika, Konfigurations-Szenarien und die Performance des Systems aus einem bestimmten Zeitintervall. Zur Laufzeit werden diese Parameter angepasst und erlauben eine kontinuierliche Adaption an veränderte Umgebungsbedingungen sowie genauere Vorhersagen.

Vorausgesetzt, das analytische Model repräsentiert die aktuelle Umgebung, so kann die optimale Konfiguration für eine gegebene Workload nach einer geringen Zahl an Berechnungen ermittelt werden und bedarf keiner iterativen Phase mehrfacher Änderungen, wie es bei dem Feedback-Ansatz häufig der Fall ist. Ein Modell-basierter Ansatz kann auch aus den Erfahrungen generalisieren und mittels seiner Modelle die Performance bisher unbekannter Workloads und damit auch die optimale Konfiguration für jene vorhersagen. Eine auf empirischen Vergleichen beruhende Umsetzung muss im Gegensatz dazu eine Reihe von Konfigurations-Szenarien durchtesten bzw. simulieren, sofern mit einer unbekannten Workload konfrontiert. Daher haben die Modell-basierten Ansätze gängigerweise geringe Laufzeiten als die anderen beiden vorgestellten Methodiken.

Jedoch bedürfen sie oft einer initialen Zeitaufwendung zum Training. Die Vorhersagen des Modells sind eben solange als unzuverlässig anzusehen, bis ausreichend Trainingsdaten gesammelt und ausgewertet wurden. Die Sammlung von umfangreichen Trainingsdaten mit verschiedensten Konfigurations-Szenarien in einem operativen System wird unweigerlich zu diversen sub-optimalen Konfigurationen führen, die letztlich schlechtere Performance als die Default-Einstellungen liefern. Das Hauptmanko liegt aber vielmehr in der Komplexität moderner Softwaresysteme begründet, die es erschwert und beinahe unmöglich macht, ein akkurates Performance-Modell aufzustellen, insbesondere bei einer Vielzahl an Konfigurationsparametern und einer Vielzahl an möglichen Werten pro Parameter.

Brewer [Bre94, Bre95] verwendet Regressions-Modelle, um Unterprogrammbibliotheken zu optimieren. Dabei setzt er auf lineare Regression, bei der die unabhängigen Variablen nicht-linear sein dürfen (d.h. eine unabhängige Variable kann das Ergebnis von zwei oder mehr Parametern eines zu optimierenden Unterprogramms repräsentieren). Obgleich der vorgestellte Ansatz gut für Unterprogramme funktioniert, ist es unklar, ob es damit möglich ist, akkurate regressionsbasierte Performance-Modelle für größere Software-Systeme zu erzeugen, insbesondere mit dem Hinblick auf mögliche Interaktionen und Abhängigkeiten zwischen den relevanten Variabeln.

Matthews et al. [MRC+97] machen sich einen Modell-basierten Ansatz zu Nutze, um eine modifizierte Version des von Unix bekannten Log-structured File System (LFS) zu optimieren. So ermöglichen sie dem LFS bspw., die bessere von zwei Methoden der Garbage Collection auf dem vom System auf Platte gehaltenen Log auszuwählen. Ihre Modelle beruhen auf einfachen Formeln zu Abschätzung der Kosten bzw. des Kosten-Nutzen-Verhältnis verschiedener möglicher Konfigurations-Szenarien, mit dem Wissen um die vom System ausgeführten Operationen und die damit verbundenen Kosten. Auch wenn die Autoren die Effektivität ihres spezialisierten Ansatzes zeigen, so ist doch unklar, wie gut die Modelle sich bei realen und vielmehr bei komplexeren Systemen zur Vorhersage der Performance eignen.

Im Rahmen des Microsoft **AutoAdmin**-Projekts wurden Modell-basierte Techniken zur Automatisierung der Selektion zu erzeugender Indexe und materialisierter Sichten [ChNa97, ACN00] entwickelt, mit dem Ziel häufig wiederkehrende Datenbankanfragen zu beschleunigen. Mit Hilfe der Kosten-Abschätzungen des Anfrage-Optimierers wird in effektiver Weise der Suchraum der möglichen Kombinationen von Indexen und materialisierten Sichten ergründet und es werden geeignete Kandidaten-Mengen bestimmt. Gerade weil die Ergebnisse später Einzug in den SQL Server 2000 Index Wizard fanden und viele der Details, insbesondere der internen Modelle des Anfrage-Optimierers, nicht aufgegriffen wurden, ist es schwierig, den Ansatz auf ein beliebiges Software-System zu übertragen.

Menascé et al. [MBD01] verwenden einen auf Warteschlangenmodellen (Queueing Network Models) beruhenden Ansatz [Laz84] zur Optimierung der Quality of Service (QoS) einer E-Commerce-Seite. Sobald ihre Implementierung eine Abweichung von den Anforderungen und damit eine QoS-Verletzung feststellt, kommt eine Suche nach dem Bergsteigeralgorithmus (Hill Climbing) zum Einsatz, die von einer gegebenen Startlösung aus solange zum besten Punkt - aus der Nachbarschaft der aktuellen Lösung - geht, bis keine Verbesserung des Zielfunktionswertes, sprich der QoS mehr möglich ist. Somit werden die Prognosen der Modelle genutzt, um die Konfiguration mit der lokal maximalen QoS zu finden. Die Autoren präsentieren die Ergebnisse ihres Ansatzes, der unter Berücksichtigung von vier Stellschrauben adäquate QoS-Werte unter ansteigender Last aufrechterhalten kann. Es ist jedoch noch unklar, inwieweit derartige Warteschlangen-Modelle in anderen Domänen (in denen Stellschrauben bspw. keine direkte Verbindung zu den Warteschlangen haben) die Basis einer generischen Software-Tuning-Lösung darstellen können.

Hervorzuheben sei hier vor allem der Ansatz von **Sullivan** [Sul03]. In seiner Dissertation wird eine Methodik zum autonomen Software-Tuning nach einem sog. Abhängigkeits-diagramm-Formalismus (Influence Diagram) und verwandten Lern- und Inferenz-Algorithmen vorgestellt und anhand der Optimierung von vier Stellschrauben der Berkeley DB evaluiert. Durch die eingeführten Abhängigkeitsdiagramme kann effektiv von Trainings-daten abstrahiert sowie verallgemeinert und somit eine bedeutende Performance-Verbess-

erung im Vergleich zu den alternativen auf linearer Regression beruhenden Ansätzen erzielt werden. Zudem unterbreitet der Autor gezielte Vorschläge, wie diese Methodik auf ein beliebiges Software-System angewendet werden und wie ein entsprechender Workload-Generator zur Erzeugung der notwendigen Trainingsdaten aussehen kann.

Wir werden die Thematik eines Modell-basierten Optimierungsansatz erneut in Abschnitt 7.5 aufgreifen und anhand eigener Konzipierungen und prototypischer Evaluierungen darlegen, welcher tatsächliche Nutzen sich bei dem Einsatz im autonomen Tuning von Datenbank-basierten Software-Umgebungen erzielen lässt, aber auch welche Probleme dabei auftreten.

3.2.3 Feedback-Mechanismen

Bei einem, bereits aus Abschnitt 2.2.4. hinlänglich bekannten, Feedback-basierten Tuning-Ansatz wird die Konfiguration in Abhängigkeit von einer oder mehreren Performance-Metriken und deren Verhältnis zu einem kritischen Schwellwert bzw. in Relation zu einem durch eine Richtlinie vorgegebenen Wunschzustand bestimmt. Der resultierende Zustand wird analysiert und sorgt ggf. für einen weiteren Durchlauf und eine erneute Anpassung der entsprechenden Stellschrauben. Die Bestimmung konkreter Parameterbelegungen kann dabei auf Heuristiken, Schätzungen und weiteren Techniken der klassischen Kontrolltheorie beruhen [FPE01].

Die autonome Umsetzung des Datenbank-Tunings nach dem Feedback-Prinzip orientiert sich dabei an der Vorgehensweise eines DBA. So muss dieser das DBMS ständig überwachen, indem er wichtige Kenngrößen sammelt und protokolliert. Stellt er nach Analyse der Daten einen Performance-Engpass fest, so sucht er die Ursache und führt geeignete Gegenmaßnahmen durch. Durch weitergehendes Überwachen des Systems können auch die Auswirkungen der durchgeführten Änderungen überprüft werden.

Tuning-Verfahren nach dem empirischen Ansatz vergleichen die Performance verschiedener Konfigurationen einer bestimmten Workload und nutzen das Ergebnis zur Ermittlung der adäquatesten Einstellung für diese Workload. Im Gegensatz dazu werden beim Feedback-Ansatz die aktuellen Werte der untersuchten Performance-Metriken solange iterativ auf eine Menge von Konfigurationsparameter-Einstellungen „abgebildet“, bis keine weiteren Anpassungen mehr notwendig sind. Hierbei wird normalerweise weder explizit die Performance verschiedener Konfigurations-Szenarien verglichen, noch die Workload auf dem System berücksichtigt.

Vorteil des Ansatzes ist die grundsätzlich schnelle Anpassbarkeit an veränderte System- und Umgebungsbedingungen bei für Feedback empfänglichen Tuning-Problemen. Derartige Methodiken sind leistungsfähig ohne jegliches Training und können daher mit bisher unbekannten Workload-Szenarien umgehen. Natürlich steigern Heuristiken und experimentelle Versuchsdurchführungen die Qualität und erhöhen die Wahrscheinlichkeit, die für die Performance-Metriken kritischen Werte zu bestimmen und die Anpassungen gezielter durchzuführen.

Einige der Nachteile dieses Ansatzes sollen nicht verschwiegen werden. Es ist oft schwer, mehrere Stellschrauben simultan zu optimieren. Im Gegensatz zu einem Parameter, den man lediglich vergrößern oder verkleinern kann, verlangt die Adjustierung mehrerer Parameter das Verständnis über deren Abhängigkeiten und den gemeinsamen Einfluss auf die Performance des Systems. Zudem kann es sein, dass eine hohe Anzahl an Schleifendurchläufen für eine optimale Konfiguration und damit unangemessen hohe Laufzeitkosten benötigt werden. Es gibt darüber hinaus Stellschrauben, wie das Rekonfigurieren des physischen Datenbank-Layouts, die sehr kostspielig in punkto

Ressourcen und Zeit sind, deren Anpassungen man zu reduzieren versucht und dabei auch eine weniger günstige Lösung in Kauf nimmt.

Das **Architectural Blueprint** for Autonomic Computing [IBM03a] beschreibt die Basiskomponenten einer allgemeingültigen, bereits in Abschnitt 2.2.4 ausführlich vorgestellten, Architektur zur Konstruktion autonomer Systeme basierend auf der vierstufigen MAPE-Feedback-Schleife und ist zur Grundlage vieler Forschungsansätze geworden.

So schlagen **Powley et al.** [PoMa06], darauf aufbauend, ein generisches Framework zur Entwicklung von Autonomic Managers unter ausschließlicher Verwendung von Konzepten und Techniken relationaler Datenbank-Management-Systeme (i.W. Trigger und Stored Procedures) vor.

Oracle spricht in Zusammenhang mit seinen Selbstverwaltungs-Funktionen von einer Self-Managing-Loop, die aus drei Phasen besteht [DaDi06]. Es wird unterschieden zwischen der Observierungs-, der Diagnose- und der Auflösungsphase. Die (1) Observierungsphase (Observe Phase) ist gleichzusetzen mit der Überwachungsphase der MAPE-Schleife. Es werden Daten gesammelt und Statistiken erstellt, um so die Grundlage für die autonomen Mechanismen zu legen. Diese Daten bzw. Statistiken werden im Autonomic Workload Repository chronologisch und über einen langen Zeitraum gespeichert. In der (2) Diagnosephase (Diagnose Phase) werden die gesammelten Daten analysiert. Dazu werden geeignete Ratgeber bzw. Advisor aufgerufen. Die Advisor erstellen im Anschluss an die Analyse Empfehlungen, um die Performance zu steigern. Diese Phase entspricht der Analyse- und Planungsphase der MAPE-Schleife und wird maßgeblich durch den ADDM (siehe Abschnitt 3.1.4) bestimmt. Die (3) Auflösungsphase (Resolve Phase) schließlich ist vergleichbar mit der Ausführungsphase der MAPE-Schleife. In ihr werden die Empfehlungen aus dem zweiten Schritt umgesetzt.

Aber auch schon vor der Autonomic-Computing-Initiative von IBM gab es Ansätze zum autonomen, Feedback-orientierten Tunen von (Teilen von) Software-Systemen. Weikum et al. verfolgten mit ihrem **COMFORT**-Projekt das Ziel, Tuning-Prozesse zu automatisieren oder gar autonom in die Verantwortung des Systems zu geben [WHMZ94]. Ziel war es, ein Datenbanksystem zu entwickeln, das auf gewisse Situationen reagieren und, sofern nötig, Parameter selbstständig ändern kann, um eine gute Performance sicherstellen zu können. Eine wesentliche Rolle spielen dabei die Fragen, auf welche Art und Weise ein Handlungsbedarf auf Grund eines Performance-Problems erkannt werden kann und ob eventuelle Änderungen nachhaltig genug sind und nicht kurz darauf widerrufen werden müssen. Sie verwendeten zur Umsetzung des Autonomic Database Tuning bereits eine dreistufige Feedback Control Loop, die sich in die Phasen Überwachung (Observation), Vorhersage (Prediction) und Reaktion (Reaction) gliedert. Auf der anderen Seite beschäftigte sich das Projekt mit konkreten Tuning-Problemen, beispielsweise der Transaktions- oder der Speicherverwaltung. Betrachtet wurde maßgeblich der Parallelitätsgrad (MPL, Multiprogramming Level) mit wesentlichem Einfluss auf das Leistungsverhalten (Durchsatz und Antwortzeit). Es wird demonstriert, wie man die Anzahl gleichartiger Objekte (z.B. Prozesse im Betriebssystem oder Transaktionen innerhalb der Datenbank), die sich die Systemressourcen (Speicher, Sperren etc.) im DBMS teilen bzw. um sie konkurrieren, dynamisch begrenzt.

Zunächst wurde ein Prototyp entwickelt, der in der Lage ist, sich selbst zu tunen. Weiterhin wurde für verschiedene Problemstellungen in Datenbanksystemen untersucht, auf welche Art und Weise diese automatisch gelöst werden können. Mit nur wenigen Ausnahmen

konnten dafür stabile Verfahren gefunden werden, die mindestens eine ähnlich gute Performance sicherstellen wie Methoden, bei denen (zumindest teilweise) manueller Handlungsbedarf von Nöten ist. Dabei erwies sich der dreistufige Überwachungskreislauf bestehend aus Beobachtung, Vorhersage und Reaktion als sehr nützlich. Weiterhin zeigte sich, dass eine Überwachung einfacher Metriken, die den aktuellen Systemzustand widerspiegeln, anstelle von schwierigen mathematischen Metriken zum automatischen Tunen sehr von Vorteil ist.

Nachteil des Vorhersage-Schritts ist allerdings, dass dabei stets davon ausgegangen wird, dass sich die aktuelle Situation in der nahen Zukunft weiter fortsetzen wird. Wünschenswerter wäre es, wenn dabei vorhergesagt werden kann, welche genauen Auswirkungen die Änderungen der Parameter auf das Datenbanksystem haben werden. Problematisch stellt sich auch die Übertragung der automatischen Routinen auf andere in der Praxis verwendete Datenbanksysteme dar. Wünschenswert wäre es natürlich, die entwickelten Problemlösungen auf realen Datenbanksystemen zu testen, um beispielsweise unerwünschte Nebeneffekte besser zu erkennen und unter Kontrolle zu haben. Weiterhin ergibt sich für das COMFORT-Tuning-Projekt eine beschränkte Tuning-Mächtigkeit, da sich die Aktionen zur Performance-Verbesserung auf das Ändern eines einzelnen Parameters beschränken. In der Regel werden dafür in der Praxis verschiedene Parameter zur Verfügung gestellt, deren Veränderung unter Berücksichtigung der aktuellen Betriebssituation unter Umständen unterschiedliche Auswirkungen haben können. In [WMHZ02] werden unter anderem Erkenntnisse und Nutzen des COMFORT-Projekts im Hinblick auf heutige DBMS und deren Anforderungen diskutiert und es wird eine Neuorientierung für die Entwicklung neuer Datenbankprodukte angeraten.

Eine Reihe weiterer Forschungsarbeiten befasste sich in den 1990ern mit dem automatischen, selbst-optimierenden Ansatz zur Performance-Kontrolle von Transaktions- und Datenbanksystemen, allerdings meist nur bezüglich eines relativ kleinen Teilbereiches (z.B. Anpassung des Multiprogramming Levels zur Begrenzung von Sperrkonflikten). Zur Vereinfachung der System-Administration wird dabei oft ein zielorientierter Kontrollansatz verfolgt (**goal-oriented resource allocation**) [BCL96, BMC+94, FNGD93, NFC92]. Dabei werden durch die Administratoren lediglich die Performance-Ziele beschrieben, nicht jedoch mehr, wie diese Ziele erreicht werden sollen. Die in Form von externen Leistungsanforderungen für einzelne Lastgruppen (z.B. Transaktionstypen) vorgegebenen Performance-Ziele werden automatisch in geeignete Einstellungen für interne Kontrollparameter abgebildet.

Kurt **Brown** et al. [BCL96, BMC+94, Bro95] haben ähnlich wie Weikum et al. die Transaktions- und die Speicherverwaltung als Szenarien gewählt. Ziel des Tunings ist hierbei das Erreichen von Workload-Klassen-spezifischen Antwortzeiten. Die Stellschrauben werden solange angepasst, bis die Ziele entweder erreicht sind, oder bis erkannt wird, dass dies unmöglich ist. Der Einsatz von Heuristiken dient hierbei der Adressierung von Abhängigkeiten zwischen den verschiedenen unabhängig von einander optimierten Workload-Klassen. Dabei werden jedoch nur entweder eine oder zwei Stellschrauben manipuliert und Schätzungen bzw. Heuristiken dienen als Leitfaden. Die Autoren gestehen ein, dass es u.U. recht lange dauern kann, bis Antwortzeitvorgaben für bestimmte Workload-Typen zufriedenstellend erfüllt sind. Beinahe alle vorgestellten Ansätze optimieren individuelle Tuning-Stellschrauben in Isolation. Die Arbeit von Brown et al. bildet die Ausnahme, da hier für jede Workload-Klasse zwei Parameter simultan optimiert werden.

Bereits seit 1999 kommt im Microsoft SQL Server ein Feedback-Mechanismus zur Anpassung der Größe des Caches für Datenbankseiten zum Einsatz [CCG+99]. Wenn die

Zahl der freien Speicherseiten im System unter einen gewissen Schwellwert fällt, wird die Datenbank-Cache-Größe verringert. Wenn die Zahl einen zweiten Schwellwert überschreitet, kann der Cache vergrößert werden. Die Autoren der Veröffentlichung schweigen sich jedoch darüber aus, wie die Schwellwerte ausgewählt und welche Abhängigkeiten zu anderen Speicherkonsumenten berücksichtigt werden müssen.

Feedback-Mechanismen finden seit geraumer Zeit bereits große Verbreitung bei Ressourcen-Verwaltungs-Problemen in **Betriebssystemen**, wie bspw. CPU-Scheduling [CMD62, Mas90, SGG+99] sowie Aspekte des Kontrollfluss im Netzwerkverkehr [Kes91, Jac88].

Die Forschungsgruppe um **Hellerstein** stellt eine Verbindung zwischen dem Autonomic Computing und der Kontrolltheorie her [DHK+05]. Basierend auf der Ähnlichkeit der beiden Ansätze werden für das automatische Verwalten von Systemen die Konzepte der Kontrolltheorie, wie Stabilität, Einpendelzeit und Steuerung der Genauigkeit des Kontrollsystems, angewendet. Die Herausforderungen liegen dabei vor allem in der Entwicklung eines zugrundeliegenden Ressourcen-Modells unter Berücksichtigung von Sensorverzögerungen und Ausführungszeiten.

Benoit beschreibt ein Verfahren zur automatischen Problemdiagnose von Performance-Engpässen in IBM DB2 Universal Database für Linux, UNIX und Windows unter OLTP-Workload [Ben05]. Ausgehend von einem Ressourcen-Modell wird ein systemspezifischer, auf fest verdrahteten Heuristiken beruhender Diagnosebaum entwickelt. Der Entscheidungsbaum wird zur Laufzeit traversiert und genutzt, um durch schrittweise Anpassung problembehafteter Ressourcen die Gesamt-Performance des Systems zu verbessern.

Bigus et al. beschreiben AutoTune, einen mittels der Agent Building and Learning Environment erstellten generischen Agenten zum autonomen Tuning [BHS00]. Zentrales Element der Architektur ist dabei eine Komponente, die das Verhalten des Zielsystems lernt, ein entsprechendes Modell erzeugt und davon wiederum eine Kontrollschleife ableitet.

In der Theorie benötigen Feedback-Mechanismen in der Regel weder Training noch Modell-Bildung. Zur Konstruktion eines effektiven Ansatzes ist jedoch in der Praxis die Durchführung von Experimenten zur Gewinnung von Heuristiken oft notwendig. Brown selbst hebt explizit die Schwierigkeit hervor, einen feedbackbasierten Ansatz zu implementieren, der mehr als eine Stellschraube berücksichtigt und betont notwendiges Vorabarbeiten, um eine akzeptable Performance zu erreichen. Auch Ansätze, die dies nicht explizit erwähnen, sind auf die Bestimmung der Schwellwerte angewiesen, die den Tuning-Prozess lenken.

3.3 Abgrenzung der bestehenden Ansätze

Momentan existiert eine große Anzahl an Konzepten, Prototypen und kommerziellen Produkten, welche primär einzelne Teile der Selbst-Verwaltung und des autonomen Datenbank-Tuning umsetzen. Ein umfassendes System, das diese vielen teilweise bereits sehr guten Einzellösungen zu einem Ganzen zusammenfasst, gibt es jedoch noch nicht wirklich. Vor allem ein möglichst integriertes und ganzheitliches Performance Management auf dem Datenbanksystem wäre äußerst wünschenswert und ist eines der wesentlichen Aspekte der vorliegenden Arbeit.

Die Wahl einer der drei vorgestellten Techniken zur Umsetzung dieser gewünschten autonomen Überwachung und Steuerung hängt stark von der Natur des zu optimierenden Systems und dem Ziel des Tuning-Prozesses ab. Wenn das System einer geringen Anzahl, regelmäßig wiederkehrender Workloads ausgesetzt ist, dann macht wohl ein empirischer Ansatz am meisten Sinn. Wenn die Ziele des Tunings aber durch Richtlinien vorgegeben und durch häufig wechselnde, verschiedenartige und nicht unmittelbar vorhersehbare Workloads beeinflusst sind, haben Modell- und Feedback-basierte Ansätze ihre Vorteile. Obwohl mithilfe eines analytischen Modells die optimale Konfiguration ohne eine Reihe iterativer Anpassungen gefunden werden kann, so überwiegen bei diesem Ansatz für uns doch die Nachteile. Zum einen muss ein entsprechendes Modell trainiert und u.U. zur Laufzeit mit einhergehenden Leistungseinbußen angepasst werden. Zum anderen ist das Konstruieren eines möglichst akkuraten Modells gerade bei komplexen System-Umgebungen eine große Herausforderung.

Dabei soll die Verarbeitung systemseitig ständig überwacht und analysiert werden, so dass Problem-Situationen unmittelbar erkannt werden können. Kontrollparameter des Systems sind automatisch einzustellen und in Abhängigkeit des aktuellen Systemzustands anzupassen. Daneben muss gewährleistet sein, dass der automatische Kontrollansatz auch in Überlastsituationen stabil arbeitet und nur vergleichsweise geringen Overhead verursacht.

Aus diesen Gründen erscheint ein **Feedback-orientierter Ansatz** mit einer auf dem DBMS aufsetzenden, **externen MAPE-Schleife** am vielversprechendsten für ein fortgeschritteneres Performance Management. Wir zeigen indes im weiteren Verlauf der Arbeit (siehe Abschnitt 7.5), dass der von uns gewählte Ansatz die Nutzung mathematisch analytischer Modelle bei weitem nicht ausschließt und durchaus davon profitieren kann.

Auf den ersten Blick nachteilig an dem iterativen Vorgehen erscheint die zusätzliche, obligatorische Bestimmung von mit Problem-Situationen verbundenen kritischen Schwellwerten sowie von Heuristiken zur Problemlösung. Dieser vermeintliche Mehraufwand stellt jedoch kein sonderlich großes Problem dar, da vielfältige Informationen über bewährte Datenbank-Tuningpraktiken in den Köpfen der erfahrenen Administratoren und zahlreich in elektronischer Form zur Verfügung stehen, vorausgesetzt, die Informationen in bspw. News-Groups, IT-Blogs, Online-Magazinen, „IBM Redbooks"-Veröffentlichungen oder Produkthandbüchern lassen sich vereinheitlichen und formalisieren.

Keine der im Abschnitt 3.2.3 aufgeführten Techniken lässt hingegen die **Formalisierung** des Tuning-Wissens durch die DBA zu. Sie basieren primär auf komplexen, starren (mathematischen) Modellen ohne Berücksichtigung oder Integration von bewährten Datenbank-Tuning-Praktiken. Zu einer effizienten Integration von Datenbank-Tuning-Praktiken in eine Autonomic-Computing-Architektur ist eine Formalisierung dieser jedoch unabdingbar. Die Idee der Formalisierung von Arbeitsabläufen ist nicht neu. Schon vor der IT-Industrie war das Gesundheitswesen in den 1990ern daran interessiert, die Qualität von Dienstleistungen zu verbessern und die Behandlungskosten zu reduzieren. Zu diesem Zweck wurden Standardarbeitsanweisungen (Standard Operating Procedures - SOPs) zur Diagnose und Behandlung von Patienten erstellt[14]. Die Medizinische Informatik (MI) entwickelte in diesem Zusammenhang formale Repräsentationen für SOPs sowie Workflow-Systeme für deren semi-automatische Abarbeitung [PTB+03, SRR+06]. Der Fokus der MI war hierbei jedoch nicht die vollständige Automatisierung der Behandlung, sondern deren Qualitätsunterstützung bzw. -sicherung. Eine ausführliche Analyse

[14]Auf der Webseite der internationalen, gemeinnützigen Organisation *OpenClinical* (http://www.openclinical.org), die sich der Verbreitung und Nutzung von Entscheidungs-unterstützenden Technologien im Klinik-Umfeld verschrieben hat, findet sich eine Zusammenstellung von Ansätzen zur Formalisierung und Modellierung von SOPs.

bestehender verwandter Formalisierungs-Ansätze und die Abgrenzung unseres Vorgehens können in [Rab11] nachgelesen werden.

Darüber hinaus sind die angesprochenen klassischen Techniken nur begrenzt erweiterbar und das automatische Tuning ist lediglich auf spezifische Bereiche sowie eine Auswahl an Konfigurationsparametern zugeschnitten. Der im Rahmen unseres Kooperationsprojektes und im Verlauf der Arbeit vorgestellte Ansatz soll jedoch universell sein. Die Überwachung und Administration des zu tunenden (Datenbank-Management-) Systems soll dabei ausschließlich über definierte Schnittstellen erfolgen. Wissen über System-Interna wird nicht benötigt, kann allerdings den Tuning-Prozess begünstigen. Dadurch ist der Ansatz nicht spezialisiert auf das Tuning von Datenbanksystemen und kann beispielsweise zur Administration beliebiger IT-Systeme verwendet werden.

Die Fähigkeit zur adaptiven bzw. proaktiven Reaktion auf (kommende) Problem-Situationen erfordert zudem Wissen über zu kontrollierende Ressourcen und deren dynamische Umgebung. Gerade weil universelle DBMS wie IBM DB2 zunehmend mit unterschiedlichen, schwankenden Workloads genutzt werden, ist das Wissen über die Art der Workload für das autonome Tuning entscheidend. Ein weiteres Problem liegt in der steigenden Komplexität und Heterogenität der Lastprofile, da neben einfacheren OLTP-Anwendungen zunehmend Daten- und Berechnungs-intensive Anfragen für den Decision Support auf denselben Daten „gleichzeitig“ zu bearbeiten sind.

Die Arbeit von Elnaffar [Eln04] beschreibt eine System-unabhängige Workload-Erkennung für DB2 mit Hilfe von Data-Mining-Techniken. Jedoch wird dabei der Einfluss des (autonomen) Tunings auf die Workload-Erkennung selbst nicht berücksichtigt. Wir greifen im späteren Verlauf in Abschnitt 7.4 diesen Ansatz auf, erweitern ihn jedoch, um sowohl eine System- als auch Tuning-unabhängige Workload-Erkennung zu ermöglichen. Die Integration von Data-Mining-Techniken ermöglicht es, Informationen über die aktuell anliegende Datenbank-Workload allen Tuning-Komponenten zur Verfügung zu stellen. Dadurch wird gewährleistet, dass sowohl bei der Problem-Erkennung als auch bei der Problem-Lösung die aktuelle Workload adäquat berücksichtigt wird.

Unser erklärtes Ziel ist die Entlastung bzw. Unterstützung der (Datenbank-)Administratoren bei Routine-(Tuning-)Tätigkeiten, insbesondere bei der Problem-Erkennung und -Diagnose sowie der Problem-Auflösung. Dies macht eine Erweiterung eines beliebigen Datenbank-basierten Softwaresystems um autonome Komponenten erforderlich. Wir haben in diesem Kapitel bereits die verschiedenen Techniken zum autonomen Tuning (und den derzeitigen Autonomiegrad in DBMS) kennen gelernt. Von dem architekturellen Standpunkt aus bedarf ein derartiges Vorhaben zumindest einer zusätzlichen (nicht notwendigerweise in das System integrierten) Komponente: dem **Autonomic Tuner**. Dieser hat, unabhängig von der verwendeten Technik im Wesentlichen zwei Hauptaufgaben, die sich wiederum in zwei separaten (Sub-)Komponenten widerspiegeln können: das Monitoring und das eigentlichen Tuning.

Die einzige Voraussetzung für unsere Methodik ist das Vorhandensein bzw. die Bereitstellung von Sensor- und Effektor-Schnittstellen seitens der zu überwachenden und zu optimierenden System-Ressourcen. Im operativen Betrieb des Systems werden idealerweise periodisch Daten und Statistiken über den Zustand des Systems gesammelt (*über eine Sensor-Schnittstelle*). Basierend auf dem aktuellen Zustand adjustiert der Tuner in adäquater Weise die Stellschrauben (der Ressourcen) des Systems im Sinne einer vorgegebenen optimalen Ziel-Konfiguration. Sobald die Monitor-Komponente eine signifikante System-Zustands-Änderung erkennt, bestimmt der Tuner, ob eine Anpassung

der Konfiguration notwendig ist und führt sie ggf. nach Anforderung zusätzlicher Statistikdaten durch den Monitor durch (*über eine Effektor-Schnittstelle*).

Offensichtlich hängt die Qualität der Entscheidungsfindung bzw. des Tuning im hohen Maße von der Qualität, dem Umfang und der Konsistenz der gesammelten Daten ab. Das Hauptaugenmerk auf der Entwicklung eines derartigen Monitors liegt damit sowohl auf der Bestimmung der relevanten Daten-Quellen, eines probaten Sammelintervalls, als auch auf der Fähigkeit, signifikante Zustands-Änderungen zu erkennen und von erwarteten, sprich natürlichen, statistischen Schwankungen zu unterscheiden. Insbesondere die nachfolgenden Kapitel 4 und 5 befassen sich mit der für das autonome Monitoring und Tuning notwendigen Datenabgrenzung und -gewinnung.

Das eigentliche autonome Tuning, vielmehr die Genauigkeit der Entscheidungen, wird zudem nicht nur von den Daten des Monitors, sondern darüber hinaus durch die Umgebung bestimmt. Darunter fallen insbesondere Faktoren des Software-Systems, die außerhalb des Einflussbereichs des Autonomic Tuner liegen bzw. Eigenschaften der Hardware, auf der das System läuft. Die auf dem System lastende Workload sei als ein entscheidender nicht beeinflussbarer Faktor genannt und gesondert hervorgehoben. Typischerweise ist ein Tuner darauf abgestimmt, in einer bestimmten Umgebung zu arbeiten und muss u.U. (in seiner internen Logik) angepasst werden, wenn sich (Teil-) Aspekte der Umgebung mit der Zeit ändern.

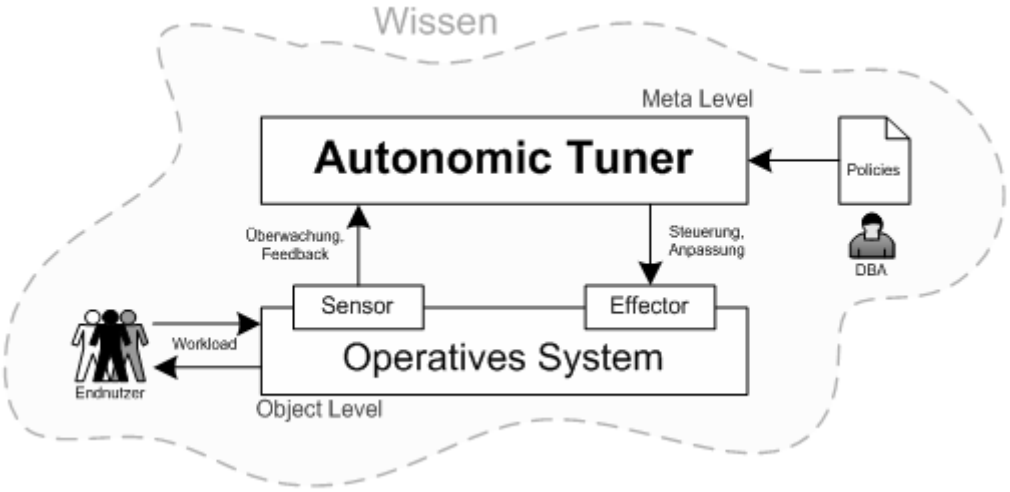


Abbildung 3.1: Autonomic Tuner - Übersicht

Zentrales Element eines Autonomic Tuner nach dem von IBM propagierten Feedback-orientierten Ansatz ist der Autonomic Manager, der verantwortlich ist für die optimierte Nutzung einer einzelnen, dedizierten Ressource. Jeder Autonomic Manager implementiert seinerseits eine rückgekoppelte MAPE-Kontrollschleife, die kontinuierlich die vier aus Abschnitt 2.2.4 bekannten Phasen durchläuft.

Abbildung 3.1 veranschaulicht vereinfacht die Integration eines derartigen Autonomic Tuner in ein bestehendes operatives System. Die Endnutzer erzeugen durch ihre Anfragen eine Workload auf dem System (Object Level), dessen Zustand durch den Autonomic Tuner (Meta Level) über die Sensoren ermittelt und über die Effektoren geändert werden kann. Ganz von selbst kann das System nicht autonom agieren. Die Administratoren sind weiterhin dafür verantwortlich, Richtlinien (Policies) zu definieren und dem Tuning-System damit den gewünschten Zielzustand und die Grenzen, in denen es autonom operiert, vorzugeben.

Die gesamte Infrastruktur lebt von und mit **Wissen**, das den Schlüssel zur Autonomie darstellt. Die Komponenten des Autonomic Tuner tauschen von außen bereitgestelltes oder ausgelesenes bzw. generiertes oder erlerntes Wissen in Form von Daten und Metadaten aus, um die Funktionalität des MAPE-Regelkreises und eine koordinierte Abarbeitung zu gewährleisten. Dabei handelt es sich beispielsweise um Wissen über Systemanforderungen, den Zustand einzelner Elemente, Topologieinformationen sowie über die Abhängigkeiten zwischen einzelnen Komponenten [WiRa09].

Im Rahmen unseres Kooperationsprojektes mit der IBM sind zahlreiche Konzepte sowie ein Basis-Framework, namens **Autonomic Tuning Expert** (ATE, siehe Abschnitt 8.3), zur Überwachung und automatisierten Abarbeitung formalisierter bewährter Standard-Tuning-Praktiken im DB2-Umfeld entstanden [WiRa07, WRRA09]. Der leitende Grundgedanke für das Architekturdesign ist dabei die MAPE-Schleife. Unsere im Rahmen der vorliegenden Arbeit beschriebene Architektur kombiniert u.a. zentrale Konzepte des Autonomic Computing mit dem an SOPs angelehnten Konzept der Formalisierung des Tuning-Wissens. Dies ermöglicht die Unterstützung der Administratoren durch eine nutzerfreundliche Definition von maschinenlesbaren Tuning-Abläufen, eine semantische Auswertung dieser zur Laufzeit und automatische Workload-abhängige Anwendung dieser bewährten Datenbank-Tuning-Praktiken zur Laufzeit.

Auf Grundlage dieser Umgebung wird es möglich, sowohl das Verhalten eines zu optimierenden Systems als auch das des selbst-optimierenden Feedback-Aufsatzes zu evaluieren und weitere konzeptionelle und prototypische Untersuchungen zum autonomen Datenbank-Tuning anzustellen.

Voraussetzung für die Spezifikation von bewährten Tuning-Praktiken sowie für eine sich anschließende autonome Analyse- und Planungsphase ist somit ein formales Modell und eine extensive Nutzung von Wissen in Form von Metadaten zur Laufzeit. In dem nachfolgenden **Kapitel 4** vertiefen wir konkret, welches Wissen zum Aufbau und Betrieb eines derartigen Systems zum autonomen (Datenbank-)Tuning nötig ist und auf welche Art und Weise man es klassifizieren kann. Es wird zudem dargelegt, inwiefern Wissen über das System sowie über die Erkennung und die Auflösung von Problemen formalisiert und als Grundlage für eine autonome Monitoring- und Tuning-Infrastruktur genutzt werden kann.

Da intelligente Entscheidungen jedoch nur auf aktuellen, verlässlichen und adäquaten Daten über den Zustand sowie das Verhaltens des Systems und der anliegenden Workload beruhen können, widmet sich der Rest der Arbeit in den **Kapiteln 5, 6 und 7** daher der Untersuchung von Techniken zur adäquaten Gewinnung, Verwaltung und Anwendung von Performance-Daten.

In **Kapitel 8** schließlich, wird unsere konzipierte und prototypisch entwickelte Monitoring- und Tuning-Infrastruktur im Detail theoretisch und anhand einiger Praxisbeispiele vorgestellt.

Kapitel 4

Wissen im autonomen Datenbank-Tuning

Eine der Grundideen des Autonomic Computing und somit auch des autonomen Datenbank-Tuning ist die Automatisierung von Tätigkeiten, die bislang von menschlichen Administratoren durchgeführt wurden. Mit dem Ziel des „Nachbaus“ der menschlichen Vorgehensweise im Sinne eines Erfahrungs- und Wissens-basierten autonomen Tuning stehen wir vor der Herausforderung, das zum Aufbau und für den Betrieb eines Tuning-Systems notwendige Tuning-Wissen zu identifizieren und maschinenles- bzw. auswertbar zu machen.

Die Fähigkeit zur adaptiven bzw. proaktiven Reaktion bzw. Reaktion auf (kommende) Änderungen in der Art der Systemlast, verlangt neben dem Wissen über die Vorgehensweisen vor allem Wissen über das System in seiner Umgebung und über die Workload. Das Kapitel gibt daher in **Abschnitt 4.1** zunächst einen systematischen Einblick in die für das autonome Datenbank-Tuning relevanten Primär-, Performance- und Metadaten sowie deren Verknüpfungs- und Anwendungsmöglichkeiten zur Unterstützung der Administratoren bzw. der autonomen Prozesse und Komponenten bei der Planung und Entscheidungsfindung. Der sich anschließende **Abschnitt 4.2** beschäftigt sich mit der für die automatische Administration notwendigen Formalisierung sowohl von Problemen als auch deren Auflösung und der Spezifikation von Zielvorstellungen mittels Policies. Daraufhin erfasst **Abschnitt 4.3** die für den Aufbau und Betrieb eines Feedback-orientierten Datenbank-Tuners zur autonomen Ausführung formalisierter Tuning-Praktiken notwendigen Wissensbereiche, deren Aufbau und Abhängigkeiten. Das Resultat bildet ein detailliertes, erweitertes Wissensmodell, dessen Schichten bereits erste Hinweise für die spätere Implementierung und die notwendigen Komponenten unseres Autonomic Tuning Expert (Abschnitt 8.3) liefern. Im finalen **Abschnitt 4.4** dieses Kapitels skizzieren wir einige spezifische Anwendungsmöglichkeiten des identifizierten Wissens zur adaptiven Regelung des Tunings und zur Unterstützung der Administratoren.

4.1 Identifikation und Klassifikation von Tuning-Wissen

Um sowohl reaktiv als auch proaktiv Performance-Probleme zu behandeln, ist ein tiefgründiges **Wissen** über das zu tunende System, die zu kontrollierenden Ressourcen, deren Umgebung, die Workload und den Problemraum erforderlich Die Wissenskomponente spielt demzufolge eine entscheidende Rolle bei der Entwicklung und Anwendung autonomer Systeme. Sie formalisiert, repräsentiert und verwaltet in geeigneter Weise alle Informationen, die für das autonome Tuning notwendig sind. Dazu gehören bspw. die Richtlinien, mit Hilfe derer Probleme identifiziert und aus den Überwachungsdaten Optimierungsaktionen abgeleitet werden können. Vor allem zählen dazu aber die für den Tuning-Prozess grundlegenden Performance-Daten, die erzeugt werden durch das *System* in

seiner Gesamtheit an Ressourcen, durch die *Workload* der End-Nutzer (SQL DML) sowie der Administratoren (SQL DDL, DCL, administrative Befehle) und durch das *Tuning-System* selbst. Die Sammlung, Verwaltung und Analyse dieser Monitoring-Daten erlaubt die Ermittlung der Workload und des Zustandes bzw. des Verhaltens des Systems, die Bestimmung der Veränderungen der Workload und die Folgen auf den Systemzustand sowie die Ermittlung der Auswirkungen der (vermeintlich erfolgreichen) Ausführung der Tuning-Aktionen.

Derartiges gesammeltes, erlerntes oder eingespeistes Wissen *über* die und *zur* Steuerung der Tuning-Prozesse bildet für uns den Schlüssel zum autonomen, Workload-basierten Datenbank-Tuning durch eine Problem-gesteuerte und -vorbeugende Ausführung von Tuning-Plänen. Die herausfordernde Aufgabe ist es demnach, die Komplexität des System- und Expertenwissens auf eine (regelhafte) Struktur abzubilden und dem intelligenten Tuning-System zur Planung- und Entscheidungsfindung sowie dem Nutzer zur Verfügung zu stellen. Die Formalisierung, Speicherung und Verwaltung des Wissens in Form von Metadaten ist wichtig für die maschinelle Abarbeitkeit, die Anwenderunterstützung und zum gemeinsamen Austausch zwischen den Komponenten sowie unter den Administratoren.

Wir bezeichnen als *Metadaten*, oder auch Daten über Daten, jede Art von Information, die für den Entwurf, die Konstruktion und vor allem die Benutzung sowie Anpassung eines autonomen Tuning-Systems benötigt wird. Dazu zählen im großen Maße Performance-Indikatoren (*Performance-Daten*) über das System und seine Ressourcen sowie die auf diesem lastende Workload.

Bevor ein autonomes System entwickelt werden kann, ist eine Analyse der anfallenden Daten und notwendigen Metadaten in dem Umfeld erforderlich. Das sollte gewissenhaft stattfinden, da spätere Modell-Anpassungen mitunter sehr aufwändig sein und einen effizienten Tuning-Betrieb behindern können. Daten und Metadaten können in verschiedenen Phasen des operationalen Betriebs und des autonomen Tunings entstehen, aktualisiert, genutzt oder ausgetauscht werden. Ihre Aufgabe besteht darin, Auskunft über Herkunft, Bedeutung, Nutzung, Struktur, Aktualität oder auch Qualität aller im System bzw. Tuning-System enthaltenen Informationen zu geben und somit sowohl die Administratoren als auch die Laufzeitumgebung bei der Evaluation, Planung und Entscheidungsfindung im Rahmen des autonomen Datenbank-Tuning zu unterstützen.

Sowohl Struktur als auch Bedeutung von Performance- bzw. Metadaten können sehr unterschiedlich beschrieben und gegliedert werden. Wir wollen daher versuchen, die für uns relevanten und für das autonome Datenbank-Tuning erforderlichen Metadaten durch die Erstellung einer Metadaten-Taxonomie (siehe **Abbildung 4.1**) annäherungsweise vollständig und disjunkt zu klassifizieren. In diesem Abschnitt werden verschiedene voneinander unabhängige **Klassifizierungsmerkmale** für Metadaten im autonomen Datenbank-Tuning vorgestellt und deren Ausprägungen stellenweise grob erläutert. Dabei steht weniger eine detaillierte Beschreibung und Begründung einzelner Aspekte im Vordergrund, als vielmehr das Vorhaben eines umfassenden Gesamtüberblicks über die Möglichkeiten der Einteilung und Betrachtung von Metadaten.

Dabei wird grundsätzlich unterschieden nach der Art der Daten (Abschnitt 4.1.1), nach deren Lebenszyklus von der Gewinnung, über die Verwaltung, bis hin zur (semi-)automatischen Anwendung (Abschnitt 4.1.2) sowie nach weiteren orthogonalen Klassifikationskriterien (Abschnitt 4.1.3).

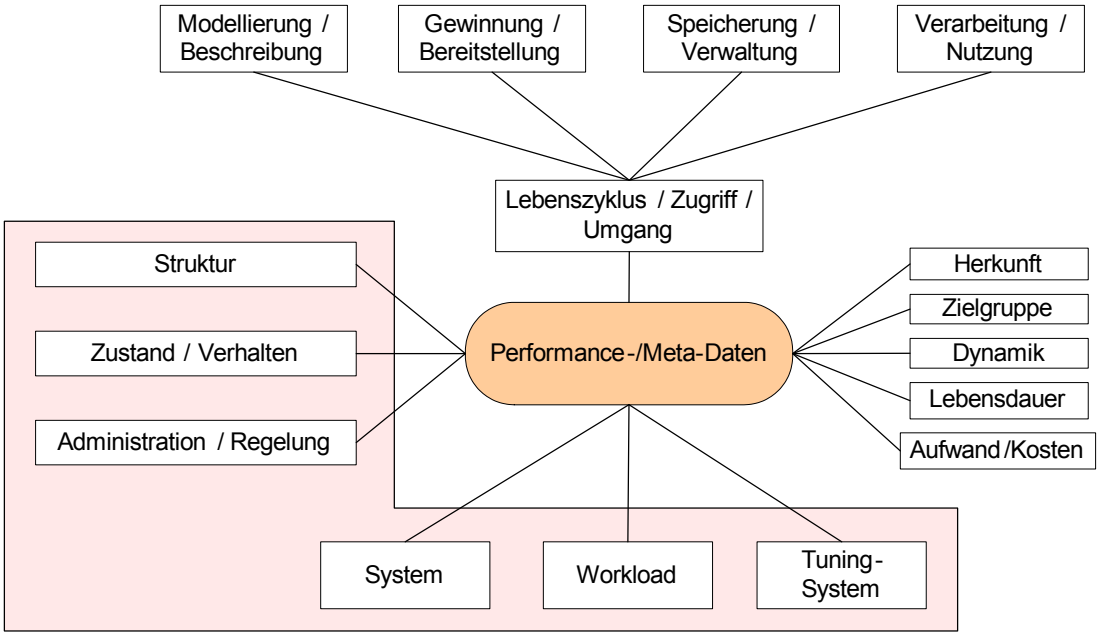


Abbildung 4.1: Metadaten-Taxonomie im autonomen Datenbank-Tuning

4.1.1 Klassifikation nach Art der (Meta-)Daten

Grundsätzlich differenzieren wir zwischen Beschreibungs-, Ausführungs- und Regelungs-Daten über das operationale System und seine Workload sowie über das Tuning-System. Konkret sind das aus unserer Sicht, in Abbildung 4.1 unten und links schematisch dargestellte:

- „normale“ operationale Daten und Metadaten
  - Primärdaten: Stammdaten der Ressourcen auf externer, konzeptioneller und physischer Ebene
    - Logische DBMS-Ressourcen (Attribute, Tupel, Tabellen, Sichten, Schemata etc.)
    - Physische DBMS-Ressourcen (Indexe, MQTs, Partitionen, Tablespace, Bufferpool etc.)
  - Metadaten: zusätzliche Daten über die Nutzdaten
    - Informativ/beobachtend
      - Performance-Daten über Zustand/Status bzw. Verhalten (Sensoren) des Systems, seiner Ressourcen und der Workload
        - Asynchrone Event-Daten
        - Synchrone Metrik-Daten
      - Topologie-Daten: über Strukturen/Hierarchien und Abhängigkeiten der Ressourcen
    - Administrativ/steuernd: zur Anpassung bzw. Konfiguration der Ressourcen über die Effektoren(-Schnittstellen)
- Workload-Daten (Anfragen, Transaktionen, Connections, Sessions, Anwendungen, Nutzer)
  - End-Nutzer-Workload (SQL DDL&DML)
  - DBAs, die auf dem System aktiv sind
  - Autonomic-Tuner-Workload (SQL DDL, SQL DCL, administrative Befehle)

- „zusätzliche“ Daten, die durch das autonome Tuning anfallen
    - Prozessdaten (zur Steuerung/Regelung der Tuning-Prozesse)
        - Beschreibungs- und Verwaltungsdaten (initial vom Mensch, dann u.U. angepasst vom System)
            - Problem-Erkennung und -diagnose
            - Problem-Auflösung
            - Policies (Richtlinien wie Ziele, Beschränkungen, (Zusatz-) Wissen über das System)
        - analytische Modelldaten (erlernt bzw. abgeleitet mittels Performance-Daten)
            - System-Modell
            - Workload-Modell
        - Konfigurations- und Administrationsdaten
            - Konfiguration und Funktionsweise der Tuning-System-Komponenten
            - Kommunikation der Tuning-System-Komponenten untereinander und mit dem operativen System
- Ausführungsdaten (über die erkannten Probleme und die Ausführung der Tuning-Prozesse, z.B. in Form einer Historie)

Performance wird auf verschiedenen (organisatorischen) Ebenen in einem DBMS (und auch in einem Software-Stack) gemessen und kann an verschiedenen Stellen beeinflusst werden. Für jede(n) Ressource(ntyp) existieren daher Sensoren bzw. Effektoren zur Bestimmung bzw. Änderung des Zustands und des Verhaltens (siehe Abschnitt 2.2.4). **Sensoren** stellen Mechanismen zur Erfassung von zumeist qualitativen und quantitativen Performance-Daten über den Status und Statuswandel des Systems und der Workload bereit. In DB2 rechnen wir bspw. Statistiken des Datenbank-Katalogs, Snapshot- und Event-Metriken des System Monitors, Traces oder auch Messages bzw. Notifications in Logs hinzu. **Effektoren** bieten Mechanismen zur Änderung des (Konfigurations-)Zustands der zu verwaltenden Ressourcen. Dazu zählen in DB2 die zahlreichen Konfigurationsparameter (DB/DBM-Cfg), die DB2-Registry, DB2-spezifische und allgemeine Umgebungsvariablen auf Betriebssystemebene, aber auch Anpassungen am physischen Design (z.B. CREATE/DROP/ALTER INDEX/TABLE/TABLESPACE/BUFFERPOOL). Da Effektoren, insbesondere die Konfigurationsparameter, genau genommen ausles- und beschreibbare Elemente darstellen und einen Wert haben, kann man sie auch den Sensoren zuordnen und zur Bestimmung des Systemzustandes verwenden.

4.1.2 Klassifikation nach Lebenszyklus der (Meta-)Daten

Neben der Strukturierung, Erfassung und Repräsentation des Wissens, der Abbildung im Computer sowie der letztlichen Darstellung, spielt im Besonderen die Verarbeitung des Wissens eine entscheidende Rolle. Die Metadaten durchlaufen nach der Modellierung, von ihren Quellen bis hin zur ihren Zielen, eine Reihe von (Lebens-)Phasen.

4.1.2.1 Gewinnung von Metadaten

Im Rahmen der Metadaten-Gewinnung ist die konsistente Bereitstellung der (aggregierten bzw. Roh-) Daten aus unterschiedlichen Quellen (von u.U. verschiedenen Systemen) über verschiedene Schnittstellen in einen oder mehrere Ziel-Datenbestände notwendig. Die Gewinnung umfasst nebst der *Sammlung* von Performance-Daten (*Monitoring*) auch die

Generierung von Wissen durch Adaption, Erlernen und Transformieren auf Basis der gesammelten Daten.

Der (Meta-)Datenfluss geht von den **Datenquellen** des Systems und des Tuning-Systems mit den darin enthaltenen Primär- und Metadaten aus. Aus mehreren zur Verfügung stehenden in Bezug auf Struktur, Inhalt und Schnittstellen meist heterogenen Datenquellen müssen die für die späteren Analysen geeigneten und ausreichenden Quelldaten ausfindig gemacht werden. Bei der Sammlung spielen neben den bereits in Abschnitt 2.3.2 genannten Aspekten des Typs, der Organisationsform oder des Ursprungs vor allem Qualitätskriterien wie Konsistenz, Vollständigkeit und Relevanz eine besondere Rolle-

Im konkreten Anwendungsfall können einige Kriterien eine wichtigere Rolle spielen als andere. Die geläufigsten **Qualitätskriterien** sind jedoch Konsistenz (Widerspruchsfreiheit), Korrektheit (Übereinstimmung mit der Realität), Vollständigkeit (z.B. Abwesenheit von fehlenden Werten oder Attributen), Genauigkeit (z.B. Anzahl der Nachkommastellen) und Granularität (z.B. tagesgenaue Daten), Zuverlässigkeit und Glaubwürdigkeit (Nachvollziehbarkeit der Entstehung, Vertrauenswürdigkeit des Lieferanten), Verständlichkeit (inhaltlich und technisch / strukturell für jeweilige Zielgruppe) sowie Verwendbarkeit und Relevanz (geeignetes Format, Zweckdienlichkeit).

Ungünstigerweise steht die Qualitität bzw. **Aktualität** der Daten bspw. im direkten Zusammenhang mit den Kosten für die Sammlung und Speicherung der Daten. Die Sammlung erzeugt einen Overhead auf dem System, den es gering zu halten gilt. Statt des intensiven Event-Monitoring, bei dem umfangreiche Daten gesammelt und übertragen, aber nur wenige benötigt werden, empfiehlt es sich daher bei DB2, auf periodische Snapshot-Daten zurückzugreifen. Werden die Snapshots jedoch zu häufig „gezogen", erhöht sich zwar die Aktualität der Daten, aber auch die zusätzliche (Überwachungs-)Last auf dem System. Die Folge kann eine sinkende Gesamtperformance sein. Erfasst man den Zustand des Systems durch Snapshots aber zu selten, verliert man u.U. an Qualität und wichtigen Informationen bzw. an Genauigkeit. Bei der Bestimmung von Systemzuständen, Werteverläufen, Schlussfolgerungen oder Vorhersagen können demnach erforderliche Werte von Metriken entweder veraltet oder aufgrund von zu hohen Sammelkosten bzw. einer verpassten Sammlung gar nicht verfügbar sein.

Die Sammlung sollte folglich den Overhead auf dem zu überwachenden System gering halten und zu jeder Zeit nur so viele Daten sammeln, wie für aktuelle oder spätere Zwecke unbedingt notwendig. Eine **adaptive Monitoring**-Strategie, wie sie in Abschnitt 5.4 beschrieben wird, ist ratsam. Die Datensammlung kann sich dabei dynamisch in folgenden Bereichen anpassen:

- Frequenz/Zeitpunkte des Monitoring (Wann/Wodurch)
  - Sofort (bei besonders hohen Anforderungen an die Aktualität der Daten)
  - Zeitfenster (periodisch, Last-abhängig, …)
  - Ereignis- bzw. Regel-gesteuert (z.B. bei Workload-Änderung oder Schwellwert-Verletzungen)
  - Anfrage-gesteuert
- Dauer des Monitoring (Wie lange)
- Umfang des Monitoring (Was)
  - Art der Daten
  - Menge / Detailgrad der Daten
- Art des Monitoring (Wie)
  - reaktiv vs. proaktiv
  - polling vs. push

- Initiator des Monitoring (Wer bzw. Welche Komponente)
    - Mensch vs. Autonomic Tuner

Die Notwendigkeit einer Adaption kann im autonomen Umfeld bspw. durch Policies, aber vor allem durch Beobachten und Lernen erkannt werden. Durch Lernen des (Normal-) Zustands und Erkennen von (Problem-)Mustern bzw. durch Vorhersage bekannter, ähnlicher Situationen können vorbeugende, intensivere Monitoring- und auch Tuning-Maßnahmen eingeleitet werden. Ebenso ermöglicht die Workload-Erkennung und -Vorhersage eine automatische Drosselung zu Spitzenzeiten bzw. während Problem-lösende Tuning-Aktionen ausgeführt werden. Informationen über abhängige Ressourcen können hilfreich sein, um die Menge an zu überwachenden Ressourcen und potentielle Auswirkungen bei (Re-)Konfiguration einzelner Ressourcen zu ergründen. Auf sich anhand diverser Indikatoren ankündigende Probleme kann man u.U. mit einer verstärkten Daten-Sammlung und -Analyse reagieren, um das bevorstehende Problem zu lindern oder gar zu beseitigen bzw. zu vermeiden.

Die Performance-Daten können, entsprechende Schnittstellen oder Tools vorausgesetzt, potentiell auf allen Ebenen des Software-Stack (siehe Abschnitt 2.1) entstehen, angefangen von der Hardware-Schicht, über das Betriebssysstem, das Datenbank-Management-System, das Netzwerk, bis hin zu Web Server oder Application Server oder auch durch die Nutzer-Applikationen. Performance-Daten über das DBMS, seine Datenbanken und die Workload in einem *DB2-Umfeld* können hierbei bspw. aus den nachfolgend gelisteten und in Abschnitt 5.3 detaillierten Quellen stammen:

- DB2-intern
    - System Monitor (Snapshot- und Event-Monitoring)
    - Health Center
    - DB2 Diagnostic und Notify Log
    - DB2 problem determination tool (db2pd)
    - DB2 Registry, DB/DBM Cfg
    - Katalog
- DB2-extern[15]
    - IBM DB2 Performance Expert
    - IBM Tivoli Monitoring for Databases
    - Betriebssystem (z.B. NETSTAT, IOSTAT, UNIX system Activity Reporter)

Die Datenquellen lassen sich zudem unterscheiden nach:

- Nutzungsebene (Primärdaten, Metadaten)
- Inhalt (Zahl, Zeichenkette, Grafik, Referenz, Dokument etc.)
- Typ (Datei, Datenbank)
- Organisationsform (ungeordnet, semi-strukturiert, strukturiert)
- Darstellung (numerisch, alphanumerisch, BLOB)
- Sprache und Zeichensatz (ASCII, EBCDIC, UNICODE etc.)
- Vetraulichkeitsgrad (z.B.: strictly confidential, confidential, public)

Typischerweise sammeln, speichern und verwalten die verschiedensten Tools auf den entsprechenden Ebenen lokal Performance-Daten über vereinzelte Ressourcen. Dies kann

[15] Eine nahezu vollständige Auflistung sämtlicher Performance-Monitoring-Tools im DB2-Umfeld findet sich unter http://www.monitortools.com/

zu einer unübersichtlichen, sowohl das menschliche als auch das autonome Tuning negativ beeinflussenden Fülle und Heterogenität führen. Die Daten sind also oft auf mehreren Systemen verteilt und müssen daher zur übergreifenden Weiterverarbeitung und Analyse in eine oder mehrere zentrale Datenbestände transportiert werden. Dabei kommt es häufig zu einer Transformation der Daten und zu einem Wechsel der Persistierungsebene (z.B. von Datei in Datenbank). Der Zugriff auf die einzelnen Metadaten in unterschiedlichen Formaten erfolgt üblicherweise mittels Schnittstellen, Dateiaustausch, Application Programming Interface (API) oder auch mittels Metadaten-Wrapper zur Abbildung zwischen verschiedenen Metadaten-Repräsentationen.

Wir verfolgen im weiteren Verlauf der Arbeit folglich die Integration und Vereinheitlichung der (Performance-)Daten aus *verschiedenen* Quellen und bedienen uns den aus dem Data Warehousing bekannten Techniken zur Extraktion, Transformation und zum Laden in *eine* globale Datenbasis (siehe Abschnitt 2.3.2). Eine entsprechende klassifizierende Einteilung kann wie folgt vorgenommen werden:

- Entdeckung von Datenveränderung/-aktualisierung in einer Datenquelle
- **Extraktion** (der relevanten Daten aus den Quellen und Abbildung auf die Ziel-Schemata)
    - Umfang der Extraktion
        - Vollständige Extraktion aller Daten
        - Partielle Extraktion der neuen/geänderten Daten
    - Strategien zur Extraktion [BaGu01, VGD99], basierend auf
        - Snapshots
        - Timestamps
        - Trigger
        - Logs
        - DBMS-Replikationsmechanismen
- Aktualisierung (des globalen Datenbestandes)
    - Nur Einfügung (z.B. Monitoring-Daten werden kontinuierlich eingespeist, aber nicht geändert)
    - Aktualisierende Änderung (z.B. Ausführungs-Metadaten über die Tuning-Pläne, Anpassungen von Modellen)
- Anreicherung (des Monitoring-Datenbestandes um semantisches Wissen)
    - Daten über die (Abhängigkeiten der) Ressourcen des Systems (siehe Abschnitt 7.5)
    - Daten über den „Tuning-Prozess“ (z.B. erkannte Problem-Situationen und ausgeführte Maßnahmen)
    - Transformierte/generierte/erlernte Daten
- **Transformation** zur
    - Bereinigung, Vereinheitlichung
    - Aufbereitung (Aggregation, Filterung, ...)
    - Ggf. „Überführung“ der Daten in ein anderes (einheitliches) Übertragungs- und Persistierungsformat
- Zuführung/**Laden** (in eine oder mehrere Ziel-Datenbestände)

Je nach Datenvolumina, Dynamik der Daten und Anforderungen (insbesondere an die Aktualität) können die Zeitpunkte der Extraktion oder des Ladens unterschiedlich gewählt werden. Analog zur obigen, im Rahmen der Sammlung genannten Übersicht kann dies demnach sofort, periodisch, Anfrage- oder auch Ereignis-gesteuert geschehen. Die genaue

technische Realisierung hängt dabei stark von der vorherrschenden Hard- bzw. Software sowie den Schnittstellen der Datenquellen ab.

4.1.2.2 Speicherung und Bereitstellung von Metadaten

Die Speicherung der gewonnenen Performance- und Metadaten dient deren Austausch bzw. der Weiterverwendung als Ausgangspunkt für den DBA und den sich dem Monitoring anschließenden Phasen des Autonomic Tuner.

Es lassen sich eine Reihe von Anforderungen an die Speicherung identifizieren:

- Effizienter Umgang mit großen Datenvolumina
- (Integrierte) und flexible Auswertbarkeit
- Historische bis hoch-aktuelle Daten
- Bereitstellung der Daten auf verschiedenen Abstraktions- bzw. Aggregations-Ebenen
- Verbessern und Beschleunigen des Entscheidungsprozesses
- Beschreibungs- (Art, Umfang, Geflecht der Ressourcen) und Laufzeitdaten (Status bzw. Verhalten der Ressourcen)
- Robustheit gegenüber Änderungen (Generizität)

In dem Metadaten-Repository liegen die Schemata und Daten der verschiedenen Quellsysteme sowie weitere generierte bzw. vorgegebene Tuning-Metadaten integriert und langfristig gespeichert vor. Damit es in einer geeigneten Umsetzung eine flexible, erweiterbare und in Struktur an die Performance-Analyse-Bedürfnisse orientierte Sicht auf die Daten bereitstellen kann, sollten Kriterien berücksichtigt werden, wie

- „Architektur“
  - Zentralisiert (ein zusammengehöriger Datenbestand)
  - Dezentralisiert (viele Einzelrepositories nahe bei den entsprechenden Komponenten/Ressourcen)
  - Föderiert (Globale, konzeptuelle Sicht auf alle Metadaten, Virtuelle Integration autonomer Repositories)
- Persistierungsebene
  - Dateisystem (verschiedene Formate)
  - Datenbanksystem (verschiedene Schemata)
- Dauer
  - Temporär
  - Short-Term vs. Long-Term (kurz- vs. langfristig)
- Format/Struktur
  - relational
  - multidimensional

In Kapitel 6 ergründen wir weitere konkrete Anforderungen an die Verwaltung von Performance-Daten und stellen Techniken des Data Warehousing zur flexiblen Speicherung vor. In Abschnitt 8.2 demonstrieren wir schliesslich den von uns eingeschlagenen konzeptionellen und architekturellen Weg einer Monitoring-Infrastruktur, um die aufgestellten Anforderungen an die Speicherung der Daten zu erfüllen.

4.1.2.3 Zugriff auf Metadaten (Verarbeitung, Anwendung, Nutzung, Austausch)

Der Wert des integrierten Datenspeichers bestimmt sich durch die darin enthaltenen Informationen und die Möglichkeiten zur flexiblen Auswertung, Analyse und Weiterverarbeitung der Daten. Ein Datenspeicher muss die Daten nicht aktiv holen, aber zumindest aktiv verwalten und für Analysen als Planungs- und Entscheidungs-Grundlage zur Verfügung stellen. Der Zugriff auf die Daten sollte hierin einerseits durch den Menschen über eine GUI, andererseits aber auch durch die autonomen Prozesse und Komponenten über entsprechende Schnittstellen erfolgen (können).

Unterscheiden kann man die Analysen nach:

- Art
  - lokal, beschränkt
  - integriert, übergreifend
  - global
- Zeithorizont
  - Retrospektiv (historische Analysen)
  - Reaktiv (z.B. Event-Erkennung und -Korrelation)
  - Proaktiv (Trend-Erkennung und Vorhersage)
- Ziel/Absicht (Auslesen und Generierung neuer Daten)
  - Problem-Erkennung und Problem-Diagnose / Ursachen-Analyse
  - Planung/Entscheidungs-Findung
  - Erkenntnisgewinn/Verständnis
  - Kontrolle/Feedback/Evaluation
  - Lernen und Vorhersagen (Trends)
- Ort
  - Auf den Quellen/Ressourcen (sinnvoll für Echtzeit-Problem-Erkennung)
  - Auf dem/den zentralen Repository/ies (sinnvoll z.B. für mittel- bis langfristige retrospektive, übergreifende (Ursachen-)Analysen)

Das vordergründige Ziel ist in unserem Szenario natürlich die kurzfristige, reaktive, manuelle bzw. automatische Problem-Erkennung und Problem-Diagnose anhand der gesammelten Daten. Unabhängig von der Verteilung und Struktur der dafür notwendigen Datenbestände muss eine zeitnahe, *integrierte* und übergreifende *Auswertbarkeit* möglich sein.

Mittel- bis langfristig verspricht das *Performance Data Mining* einen weiteren Nutzen, durch Extrahieren neuer Erkenntnisse bzw. Generieren von Wissen aus den (integrierten) Performance-Daten (siehe Abschnitt 7.3). Beispiele für eine derartige Knowledge Discovery sind:

- Mining auf den „normalen“ Performance-Daten
  - Erstellen eines System-Modells
  - Erstellen eines Workload-Modells zur Workload-Erkennung und -Vorhersage (siehe Abschnitt 7.4)
  - Ermitteln der Abhängigkeiten zwischen Ressourcen sowie zwischen Effektoren und Sensoren (siehe Abschnitt 7.5)
  - Automatisches Lernen eines Modells normaler Aktivität / normalen Verhaltens (Normalzustand) aus den Monitoring-Daten
- Mining auf den Daten über die Tuning-Prozesse
  - Laufzeit-Statistiken über Ausführung und Effizienz von Tuning-Plänen
  - Automatisches Ableiten von Problem-Situationen und Ereignissen

  - Automatische Bestimmung von Event-Korrelationen
  - Ermittlung von Events, die häufig auftreten bzw. die nicht/gut aufgelöst wurden
  - Erkennen von konfligierenden Regeln bzw. Tuning-Zielen

4.1.3 Weitere orthogonale Klassifikationskriterien

Die Metadaten lassen sich zudem nicht ausschließlich nach den obigen Kriterien unterscheiden, sondern auch wie folgt strukturieren und kombinieren:

- Dynamik (Bestimmen, Aktualisieren, Zugreifen etc.)
  - Statisch (z.B. einmal zu bestimmen)
  - Dynamisch (z.B. häufig zu aktualisieren)
- Herkunft
  - Mensch (Policies, Best-Practices, Thresholds, Events etc.)
    - End User mit Erwartungen/Anforderungen
    - Anwendungsentwickler mit Qualifikationen
    - Administratoren mit ihrem Qualitätsanspruch
  - Maschine (Erlernen/Bestimmen/Auslesen)
    - (Zur Laufzeit) vom Tuning-System
    - Vom zu überwachenden System
  - Hybrid (vom Mensch default-Wert vorgeben und diesen durch die Maschine anpassen lassen)
- Zielgruppe/Verwendung
  - Business-Metadaten (z.B. Reports, Analysen) für den Menschen
  - Technische Metadaten für Nutzung durch Prozesse/Komponenten (maschinelle Weiterverarbeitung)
- Typ
  - Struktur-Metadaten zu Primärdaten (Ressourcen, Quellen der Sensoren, Effektoren, Workload, Strukturdefinitionen, Topologie ...)
  - Prozess-Metadaten (Daten zur Steuerung der und über die Ausführung der Prozesse: ETL-Regeln, Logs/History, Ausführungspläne, ...)
- Nutzungsart
  - Passiv: als Dokumentation der verschiedenen Aspekte des Systems
  - Aktiv: Speicherung semantischer Aspekte (z.B. Transformationsregeln oder erwartete Kosten) sowie deren Interpretation zur Laufzeit
  - Semiaktiv: Speicherung von Strukturinformationen (Tabellendefinitionen, Ressourcenbeschreibungen, Konfigurationsspezifikationen) und Nutzung zur Überprüfung (nicht direkt zur Ausführung)

Die Aufzählung soll lediglich einen weiteren Einblick in zusätzliche Sichtweisen auf die Daten vermitteln und erhebt keinen Anspruch auf Vollständigkeit.

4.2 Formalisierung von Experten-Tuning-Wissen

Daten und Metadaten können in verschiedenen Phasen des operationalen Betriebs und des autonomen Tunings entstehen, aktualisiert oder genutzt werden und sind in Struktur und Art vielfältig. Im Sinne des Nachbaus der menschlichen Tuning-Vorgehensweise mit einem Feedback-orientierten Ansatz ist neben dem Wissen über das System maßgeblich (Tuning-) Wissen über typische Performance-Probleme, deren Diagnose und deren Lösungsbehandlung nötig.

Heutzutage stehen vielfältige Informationen über derartige bewährte Datenbank-Tuning-Praktiken außer in den Köpfen der Administratoren in elektronischer Form typischerweise textuell in Produktdokumentationen, Publikationen oder auch Foren zur Verfügung. All diese Informationen sind jedoch zum größten Teil unstrukturiert, d.h. nicht formalisiert und somit auch nicht maschinell abarbeitbar [WiRa07, WRRA08, WiRa09].

Dieses Wissen muss formalisiert, materialisiert und maschinell auswertbar den einzelnen Komponenten zur Verfügung gestellt werden, damit ein autonomes Tuning-System wie ein erfahrener DBA auf erkannte Problem-Situationen mit bewährten Tuning-Maßnahmen reagieren kann [RWRA08]. Die Formalisierung, sprich die strukturierte Abbildung, und Speicherung typischer DBA-Vorgehensweisen bietet den Administratoren zudem eine Möglichkeit zur Erhaltung, Verwaltung und zu einem potentiellen Austausch ihres Wissens.

4.2.1 Tuning-Richtlinien (Policies)

Autonome Systeme sollen sich selbstständig auf neue Anforderungen einstellen, an veränderte Randbedingungen anpassen und sich in dynamisch veränderliche Umgebungen einbetten können. Bei den meisten für den praktischen Einsatz geeigneten Ansätzen spielen dabei sogenannte (Management-)**Policies** eine zentrale Rolle. Dies sind, vereinfacht gesagt, aus den unternehmerischen Zielen unter Beachtung technischer Gegebenheiten abgeleitete Richtlinien, nach welchen die (autonome) Verwaltung und Optimierung eines IT-Systems durchgeführt werden sollen. Somit beeinflussen in einem Autonomic-Computing-System die Policies die Handlungen der Autonomic Tuner.

Der Begriff Policy bezeichnet allgemein Regeln, die Abläufe beeinflussen. Solche Regeln können sowohl einschränkender als auch erweiternder Natur sein [Dem99]. Zusätzlich wird oftmals auf Basis der Abstraktionsstufe noch zwischen High-Level- und Low-Level-Policies unterschieden [MoSl93]. Abstrakte (**High-Level-**)**Policies** definieren grobe systemweite Zielvorgaben, Richtlinien und Rahmenbedingungen (WAS soll erreicht werden, z.B. „Datenverlust durch Feuer oder defekte Datenträger vermeiden“, „Antwortzeit des Systems verbessern“). Detaillierte, Technik-nahe (**Low-Level-**)**Policies** haben hingegen den Charakter von Handlungsanweisungen, Regeln und konkreten Betriebsbedingungen. Sie entsprechen oft den Regelalgorithmen, d.h. Abfolgen von Aktionen (WIE soll es erreicht werden, z.B. „Führe jeden Sonntag 22:00 Uhr das Backup-Skript aus“).

Um High-Level-Policies anwendbar zu machen, müssen diese schrittweise in weniger abstrakte Policies überführt werden. Dies kann durch eine Verfeinerung der Aufgaben, Aufteilung der Ziele oder Weitergabe der Zuständigkeit an andere Personen bzw. Objekte erfolgen [MoSl93]. Idealerweise sollten lediglich abstrakte und übersichtliche Policies erstellt bzw. editiert und daraus technisch adäquate Low-Level-Policies generiert bzw. abgeleitet, verteilt und schließlich zur Laufzeit sowohl ausgewertet als auch durchgesetzt werden. Bekanntestes, aber funktionell stark eingeschränktes Beispiel eines solchen Verfahrens sind Netzwerk-Firewalls, deren Filter-Listen unmittelbar interpretierbare Low-Level-Policies darstellen.

Policies werden üblicherweise auf der Basis von Policy-Modellen und Policy-Sprachen, oft mit Hilfe eines **Policy-Editors** erstellt. In [Rei08] sowie in [Alg10] stellen wir eine Reihe typischer Anwendungsfälle im Datenbank-Tuning auf und analysieren einige der bekanntesten (teils standardisierten) Policy-Sprachen und -Modelle (z.B. Ponder, Web Service Level Agreement, Autonomic Computing Policy Language, CIM Policy Model) hinsichtlich ihrer Anwendbarkeit in unserem Szenario.

Der Begriff der Policy wird hierbei (für die nachfolgenden Betrachtungen) weit gefasst. Für uns stellt sie in umfassender Weise ein Konstrukt dar, um dem Administrator die Möglichkeit zu geben, sein Wissen über das System, die Umgebung, die gestellten Ziele (Soll-Werte) und über die Prozesse zu formulieren. Wir unterscheiden daher

(1) **Goal Policies** zur Angabe von Tuning-Zielen (z.B. „Nebenläufigkeit optimieren", „Sortiervorgänge optimieren", „Durchsatz erhöhen" etc.),

(2) **Information Policies** zur Angabe von Zusatzinformationen über das zu tunende System (z.B. Zeit- bzw. Anwendungs-abhängige Workload-Informationen über typische bzw. bekannte Auslastungszeiten, erwartete Workloads, Zugriffsprofile von Anwendungsgruppen, Wartungs-Fenster etc.),

(3) **Priority Policies** zur Festlegung von Prioritäten bei der Abarbeitung der Tuning-Abläufe bzw. bei der Betrachtung von Ressourcen (z.B. „Bufferpool A hat eine höhere Priorität bei der Speicherverteilung als Bufferpool B") sowie

(4) **Constraint-Policies** zur Definition von Beschränkungen, die während des Tunings eingehalten werden sollen (z.B. „Bufferpool A darf eine bestimmte Anzahl von Seiten nicht unterschreiten" oder „es dürfen keine Indexe von Tabelle2 gelöscht werden").

Einige Policy-Sprachen lassen sehr große Spielräume für eigene Erweiterungen zu, dennoch sind (fast) alle zu sehr spezialisiert. Aus diesem Grund definieren wir in [Rei08] ein eigenes Modell sowie eine darauf aufbauende Policy-Sprache, mit deren Hilfe das Tuning gesteuert werden kann. Dabei bildet das **Modell** ab, wie die Richtlinien verwendet werden und wie das Tuning im Ganzen koordiniert wird. Die XML-basierte **Sprache** dient dem Datenbank-Administrator zur Spezifikation der Tuning-Richtlinien, mit denen er das Tuning nach seinen Zielvorstellungen beeinflussen kann. Wir ermöglichen damit die Definition und Anwendung der vier vorgestellten Tuning-Richtlinien zur Berücksichtigung von bestimmten Performance-Problemen und zur gezielten Auswahl von Tuning-Abläufen sowie die Überwachung der im Rahmen des Tunings vorgenommenen Änderungen unter Berücksichtigung gewisser Constraints. Im letzten Teil der Arbeit werden die Implementierung und Integration in den bestehenden ATE-Protoypen sowie einige Beispiele zur Verdeutlichung des Ablauf und der Nutzung erläutert.

4.2.2 Problem-Definition und -Erkennung

Erfahrene DBA reagieren auf ihnen bekannte Problem-Situationen mit bewährten Tuning-Maßnahmen. Typischerweise dient die Analyse von protokollierten Ereignissen dem Erkennen von derartigen Problemen und Fehlfunktionen in IT-Systemen. Ereignisse oder auch **Events** sind als besondere Geschehnisse anzusehen, denen ein Zeitpunkt zugeordnet werden kann und die wesentliche Systemaktivitäten oder auch kritische Problem-Situationen widerspiegeln. Man kann sie auch als Indikatoren (Menge von Symptomen) auffassen, die das Eintreten einer Situation signalisieren, auf die unter Umständen reagiert werden muss [RWRA08]. Symptome können durch Beobachten und Lernen automatisch erkannt, aus den Policies abgeleitet oder durch die Administratoren vordefiniert sein. Beispielhaft für Ereignisse seien der Beginn bzw. Abschluss von Transaktionen, der Start von Systemkomponenten und die Verbindungsaufnahme zu einer Datenbank genannt. Ist in einer Log-Datei bspw. der Beginn, (noch) nicht aber das Ende einer Transaktion vermerkt, kann das auf ein Problem hinweisen, das man näher untersuchen sollte.

Grundsätzlich können für das zu optimierende System kritische Situationen durch die Endnutzer, den Administrator oder das System selbst identifiziert werden. Da die

Endnutzer in der Regel die Probleme nicht genau spezifizieren, u.U. gar zu spät erkennen können und oft nur die (un-)mittelbaren Auswirkungen spüren (z.B. Antwortzeit steigt), ist diese Art der Problemidentifikation nur bedingt zur automatischen Administration einsetzbar.

Komplexe Systeme sind daher nur noch wirtschaftlich betreib- bzw. administrierbar, wenn derartige kritische Situationen automatisch erkannt und entsprechende (korrigierende) Verwaltungs- und Optimierungsmaßnahmen automatisiert eingeleitet werden können. Insbesondere sollte durch zusätzliche Mechanismen gewährleistet sein, dass (1) rechtzeitig vor drohenden Problemen gewarnt wird (z.B. Datei-Kapazitätsgrenzen), (2) die Administratoren über fehlerhafte bzw. ausgefallene Prozesse und Komponenten informiert werden, (3) die eigentlichen Ursachen von Problemen (root causes) schnellstmöglich bestimmt werden können, (4) System(komponenten)-übergreifende Betrachtungen und Analysen erfolgen und (5) automatische Problemlösungen angestoßen werden.

Die Event-Verarbeitung ist oft streng verbunden mit der Ursachen-Analyse (Root Cause Analysis), also der Bestimmung der eigentlichen Ursache des Auftretens von einem oder mehrerer Events. Beispielsweise erzeugt eine Fehler-Situation in einem Netzwerk eine Menge von Alerts, von denen jedoch nur einer als Hauptursache in Frage kommt. Aufgrund der kausalen Abhängigkeitskette können wiederum andere Komponenten Fehler werfen oder gar ausfallen. Da auf die Komponenten nicht mehr zugegriffen werden kann, werden diesbezüglich wenig hilfreiche und u.U. viele Events erzeugt. Erleichtern kann man sich die Ursachen-Analyse beispielsweise durch Wissen über Topologien und Abhängigkeiten zwischen den Komponenten bzw. Beziehungen zwischen Events. Ein Ressourcen-Modell, wie das in Abschnitt 7.5 vorgestellte, kann bei der Ermittlung solcher Wechselbeziehungen hilfreich sein.

Die Log-Analyse (auf UNIX-Systemen) ist eine der Vorreiter und Haupt-Anwendungsgebiete der Event-Verarbeitung. Logs beinhalten reichhaltige Informationen über den Zustand und das Laufzeitverhalten des Systems. Prinzipiell kann jeder Log-Eintrag als ein Event gesehen werden. In einer Vorverarbeitungsphase wird die gängigerweise hohe Anzahl an Log-basierten Events für die weitere Verarbeitung verringert. Dabei werden sowohl unwichtige (Spam-)Events, als auch sehr ähnliche oder gar doppelte gruppiert und in ein standardisiertes Format überführt. Dabei kommen sowohl Techniken von Spam-Filtern als auch reguläre Ausdrücke zum Einsatz. Für eine Übersicht und eine Einführung in die wesentlichen Konzepte der Log-Analyse und -Verarbeitung sei auf [KeSo06] verwiesen.

Wir wollen im Folgenden vermehrt solche Events betrachten, die dem Prozess der Performance-Problem-Erkennung und -Lösung behilflich sind und die während des normalen operationalen Betriebs durch Logging (z.B. DB2 diagnostic log) oder auch im Falle eines Fehlers bzw. auf gesonderte Anfrage zur gezielteren Diagnose durch die Systemkomponenten erzeugt werden. Dies sind u.a. (1) Events über den operationalen Status oder Zustands-Änderungen von überwachten Komponenten (z.B. Start/Stop, ungeplanter Ausfall etc.), (2) Grenzwertverletzungen von Performance-Indikatoren (z.B. Anzahl gehaltener Sperren überschreitet einen Schwellwert), (3) durch Endnutzer vorgenommene Aktionen, auf die u.U. reagiert werden muss (z.B. Connect/Disconnect, DDL-Anweisungen, Passwort-Änderungen etc.), aber auch (4) Ereignisse, die durch Verwaltungs- bzw. Optimierungsaktionen der Administratoren oder der Automatismen entstehen. Auch die autonomen Komponenten müssen (auf einer höheren Meta-Ebene) überwacht werden, um bei abnormalen Situationen entsprechend reagieren zu können (bspw. bei zu vielen Event-Benachrichtigungen oder auch Logging-Problemen aufgrund von Plattenspeichermangel).

Zur Formalisierung von Problemen verwenden wir daher in unserem Szenario den Systemzustand repräsentierende Metriken und definieren „ungesunde" Werte bzw. Werte-Bereiche (durch Schwellwerte), auf die mittels Tuning-Aktionen reagiert werden soll. Darüber hinaus kann ein Problem aber auch durch eine bestimmte Nachricht bzw. einen Log-Eintrag signalisiert werden. Events können zudem nicht nur Probleme darstellen, sondern auch für erfolgreich abgeschlossene Aktionen (z.B. Reboot des Servers) generiert werden.

Die auf Probleme hindeutenden Werte sowie Wertebereichs-Überschreitungen können in einer komplexen System-Umgebung hinreichend oft auftreten und eine entsprechend hohe Last erzeugen. Zudem muss nicht jedes einzelne Ereignis eine Alarmierung auslösen und stellt oft nur einen Bruchteil eines größeren Gesamtbildes dar. Oft führt erst die Verknüpfung einzelner Alarmsituationen zu einem kritischen Event, bei dessen Eintreten eine Reaktion Sinn macht. Nur wenn bestimmte zusammengehörige Events erfasst werden, kann man sich ein treffenderes Gesamtbild der Situation verschaffen und entsprechend reagieren. Daher reichen primitive Ereignisse oftmals nicht aus, wenn auf komplexe Situationen reagiert werden soll. Beispielsweise kann eine Abfolge von einzelnen Schwellwert-Überschreitungen auf ein schwerwiegendes Problem hindeuten, während die einzelnen Übertretungen für sich betrachtet nur wenig interessant scheinen.

Um anspruchsvollere Situationen zu charakterisieren, den entsprechenden Kontext zu berücksichtigen (z.B. „waren noch andere Schwellwerte überschritten, während die interessierende Metrik den „gesunden" Wertebereich verlassen hat") und die Event-Flut zu reduzieren, lassen sich Events zu komplexen Problem-repräsentierenden, korrelierten Events konsolidieren. **Korrelation** kann erheblich die durch die Datenflut entstehende Belastung auf dem System und den Administratoren und somit das „Verpassen" wichtiger Events aufgrund des hohen Grundrauschens reduzieren. Es steigt die Chance, dass relevante Ereignisse erkannt und rechtzeitig Reaktionen ausgelöst werden.

Mit Hilfe einer **Ereignis-Algebra** lassen sich die Verknüpfungsmöglichkeiten von Ereignissen zu neuen korrelierten Ereignissen präzise beschreiben. Hierbei werden Ereignisse unter Verwendung von *Konstruktoren* zu neuen Ereignissen zusammengefügt. Die Mächtigkeit einer typischen Ereignis-Algebra [CKAK94, DiGa00] erstreckt sich dabei über die folgenden Bereiche, wobei sich beliebige Kombinationen von den Ereignis-Konstruktoren und damit folgende Arten der Analyse definieren lassen:

- **Filterung**/Unterdrückung: Aus wiederholt auftauchenden Ereignissen, die dasselbe Problem identifizieren bzw. in Relation zueinander stehen, werden die redundanten Informationen ausgeblendet und nur wichtige Informationen (Ereignisse) herausgefiltert. Beispiele sind das Verwerfen von irrelevanten Events mit niedriger Priorität, das Verwerfen von Events, die von anderen bereits gefeuerten Events abhängen bzw. einem generalisierten Event zugeordnet werden können oder das Ignorieren von veralteten Events. Typischerweise ist die Filterung mittels regulärer Ausdrücke als (Vor-) Verarbeitungsschritt vor der Notifizierung anzusehen.
- **Throttling**: Spezialisierung der Filterung. Es werden nur Events berücksichtigt, die einige Male auftauchen oder nach einem bestimmten Zeitintervall (Calm Down Period) nicht verschwinden. Beispielsweise ist es bei einem gescheiterten Ping sinnvoll, eine Weile zu warten und es erneut zu versuchen, bevor man gleich (re)aktiv zu handeln versucht. Wenn der Ping in diesem Zeitintervall nicht erfolgreich ist, kann man den Verlust der Verbindung signalisieren.

- **Schwellwertbildung** (Thresholding): Bei Erreichen eines bestimmten Schwellwertes werden daraus die zusammengesetzten Ereignisse ermittelt, beispielsweise wenn innerhalb einer Zeitspanne eine gewisse Anzahl an gleichen atomaren Ereignissen aufgetreten ist.
- **Sequenzbildung** (Sequencing): Aus bestimmten vollständigen oder unvollständigen Ereignis-Sequenzen werden zusammengesetzte Ereignisse gebildet und somit das Problem identifiziert. Das komplexe Ereignis tritt ein, wenn alle Einzel-Ereignisse in einer bestimmten Reihenfolge und ggf. bestimmten Zeit eingetreten sind.
- **Aggregation**/Generalisierung: Zusammenfassen von mehreren einander verschiedenen Events zu einem abstrakteren Event. Die Aggregation umfasst erfahrungsgemäß boolesche bzw. Wiederholungs-Konstruktoren, wie bspw.:
    - Negation: Das komplexe Ereignis tritt ein, wenn das Teil-Ereignis innerhalb eines definierten Intervalls nicht eintritt. Zur Angabe des Intervalls wird der Intervall-Konstruktor verwendet.
    - Disjunktion: Das komplexe Ereignis tritt ein, wenn mindestens eines der Einzelereignisse eintritt. Der Disjunktions-Konstruktor ist sinnvoll, wenn bei mehreren Ereignissen die gleichen Bedingungen geprüft und dieselben Aktionen ausgeführt werden sollen. Dadurch lässt sich die Regelmenge leicht reduzieren und übersichtlicher gestalten.
    - Konjunktion: Das komplexe Ereignis tritt ein, sobald alle der Einzelereignisse eingetreten sind. Der Konjunktions-Operator ist wichtig, um Ereignisse beschreiben zu können, für die mehrere Geschehnisse notwendig sind
    - Wiederholungs-Konstruktoren: Diese Konstruktoren werden auch als Zählkonstruktoren bezeichnet, da das Eintreten der Teil-Ereignisse mitgezählt wird. Für das komplexe Ereignis werden nur bestimmte Auftreten eines wiederholt vorkommenden Teil-Ereignisses herangezogen. Dadurch ist es möglich, das mehrfache Eintreten des Teil-Ereignisses nicht immer, sondern nur ab und zu (erstmalig, bei jedem n-ten Eintreten bzw. zwischen dem mindestens n-maligen und maximal m-maligen Eintreten) zur Kenntnis zu nehmen.
- **Kompression**: Die Kompression meint das Zusammenfassen von mehreren gleichen oder ähnlichen Events und kann als eine Art Generalisierung der Entfernung von Duplikaten betrachtet werden.
- **Temporale Logiken**: Betrachtung der zeitlichen Informationen bzgl. des Auftretens einzelner Events (z.B. zeitliche Reihenfolge der Events), wenn die Eintrittsreihenfolge der Teil-Ereignisse relevant ist. Oft finden die temporalen Bezüge bei der Definition von Aggregations-, Kompressions- und Filterungs-Konstruktoren Anwendung. Beispiele sind vielfältig:
    - Intervallgrenzen: Angabe von Intervallgrenzen, innerhalb derer die Teil-Ereignisse überwacht werden sollen. Event A sollte bspw. stets im Intervall zwischen T1 und T2 beobachtbar sein. Oder aber, Event A wurde in einem bestimmten Zeitintervall T (nicht) beobachtet/erkannt. Es können absolute Zeitpunkte für die Intervallgrenzen angegeben werden oder die Eintrittszeitpunkte anderer Ereignisse bilden die Intervallgrenzen.
    - Relative Zeiten: Das komplexe Ereignis tritt in einem spezifizierten Zeitabstand zu dem Teil-Ereignis ein.
    - Festlegung einer niedrigen Priorität für bestimmte Events während eines Wartungsfensters.

- Behandlung des Events von Typ A und Unterdrücken von Events der Typen B, C, D für n Sekunden.
- Auf Event A folgt stets Event B (Paar-Events).

Es existieren eine Reihe an Tools und Techniken zur Korrelation bzw. Erkennung von Events. Neben den verbreiteten Muster-Erkennungs-Strategien reicht das Spektrum von Regel-basierten Experten-Systemen[16], dem Einsatz von für den Praxiseinsatz fraglich komplizierter Prolog-basierter Prädikaten(logik) in Tivoli TEC[17], über Syntax-Parsing[18] bis hin zu statistischen Algorithmen zur Anomalie-Erkennung. Bei letzteren ist es möglich, auch ohne Wissen über die zu erkennenden Events, durch Beobachten des Systems den Normalzustand und Abweichungen davon zu erkennen.

Im Rahmen unserer Entwicklungsarbeiten an der Infrastruktur zum autonomen Datenbanktuning (siehe Abschnitt 8.3) setzen wir die Korrelationstechnologie IBM Tivoli Active Correlation Technology (**ACT**) ein [BiGa05]. ACT ist kein eigenständiges Produkt, vielmehr eine Technologie, die Methoden zur Verarbeitung zusammengesetzter Ereignisse anbietet und in Bereichen verschiedenster Art eingebettet werden kann. Die Implementierung autonomer Computersysteme und Systeme zur Übermittlung von Nachrichten in einem Computernetzwerk (Message Broker) sind einige beispielhafte Szenarien, bei denen ACT Anwendung finden kann.

In ACT werden die ankommenden atomaren Events in einer Queue angesammelt. Diese wird mit einem Satz von vordefinierten, mit einer XML-basierten Regelsprache definierten Mustern (Korrelationsregeln) analysiert und auf Übereinstimmung verglichen. Bei jedem erkannten Muster werden die entsprechenden in der Regel enthaltenen Aktionen automatisch ausgeführt.

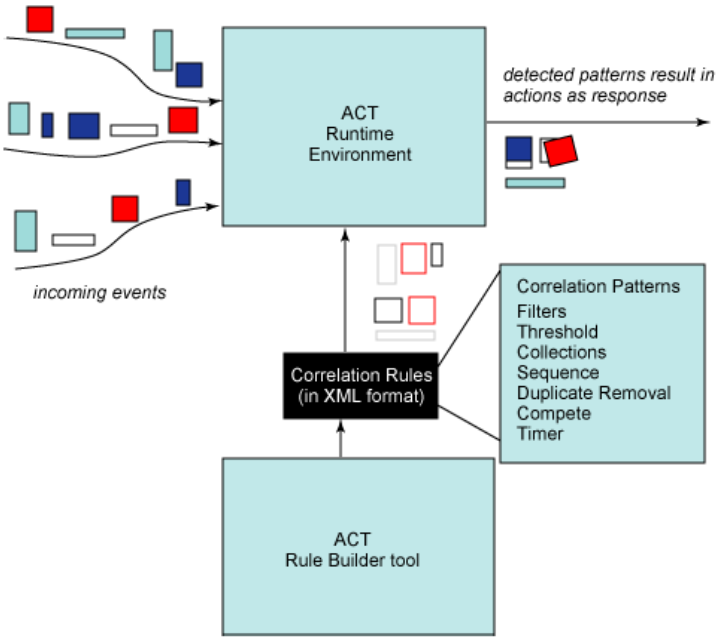


Abbildung 4.2: Architektur der Korrelations-Engine ACT [BiGa05]

[16] http://www.cs.nott.ac.uk/~sxp/ES3/index.htm
[17] http://www.softpanorama.org/Admin/Tivoli/TEC/index.shtml
[18] http://www.softpanorama.org/Admin/Event_correlation/perl_based_event_correlation.shtml

Die aus [BiGa05] entnommene **Abbildung 4.2** veranschaulicht die ACT-Architektur. Im Mittelpunkt steht eine Laufzeitumgebung (Runtime Environment), in der die Korrelations-Technologie eingebettet ist. Ein weiterer wichtiger Bestandteil von ACT ist die XML-basierte Regelsprache, in der die verwendeten Korrelationsregeln (Correlation Rules) definiert sind. Sie können mittels des zur Architektur gehörenden Regelerzeugers (Rule Builder tool) spezifiziert werden. Hereinkommende erkannte, atomare Ereignisse werden in der Laufzeitumgebung gesammelt. Diese werden anschließend mit Hilfe der Korrelationsregeln analysiert, um zusammengesetzte Ereignisse identifizieren zu können. Nähergehende Details zu ACT finden sich unter [IBM06f, IBM06g].

Für die Spezifikation zusammengesetzter Ereignisse bietet ACT sieben verschiedene Regelarten (Rule Patterns) an. Die von ACT unterstützten Korrelationsregeln umfassen dabei die Filter Patterns, Collection Patterns, Duplicate Patterns, Computation Patterns, Threshold Patterns, Sequence Patterns sowie Timer Patterns [BiGa05]. Der angebotene Umfang hat sich bewährt, um die meisten Ereigniskorrelationssituationen zu erfassen und eine Abbildung auf die bereits vorgestellten, aus der Literatur bekannten Event-Konstrukte zu ermöglichen [Jäg07]. Durch Zurückführen der korrelierten Events in die ACT-Queue lassen sich die Korrelationsmuster derart miteinander kombinieren, um noch komplexere Muster zu erhalten.

Inzwischen besitzt beinahe jedes DBMS aktive Mechanismen, i.d.R. in Form von Triggern, die reaktives Verhalten unterstützen. Insbesondere sogenannte **Aktive Datenbanksysteme** bieten viele Konzepte und Strategien für selbstständiges Handeln und sind dazu fähig, vorher definierte, eingetretene Situationen zu erfassen und anschließend selbstständig definierte Folgeoperationen auszulösen [DiGa00]. In [Jäg07] zeigen wir, dass die aktiven Mechanismen zwar nicht versuchen, das Problem der manuellen Administration zu bewältigen, sich aber durchaus auf autonome Ansätze übertragen lassen.

Ziel der Arbeit war es vor allem, in beiden Welten im Zusammenhang mit der Event-Verarbeitung vorherrschende Konzepte zu evaluieren und zu untersuchen, inwieweit sie auf das autonome Datenbank-Tuning anwendbar sind. Es zeigte sich, dass man bei der Entwicklung von autonomen Systemen auf aktive Mechanismen zurückgreifen kann, wodurch allerdings auch deren Probleme geerbt werden. Diese kann man danach unterteilen, an welcher Stelle der Regelbearbeitung sie auftreten. Problembehaftet können die *Ereigniserkennung*, die *Regelauffindung* bzw. Regelauslösung und die *Regelausführung* sein. Eine zentrale Fragestellung bei der Regelausführung ist jene der Konfliktauflösung (Konfluenz), sprich in welcher Reihenfolge die ausgelösten Regeln ausgeführt werden sollen, wenn zum Zeitpunkt der Regelausführung mehrere Regeln angestoßen wurden [DGG95]. Ferner können Regeln ihrerseits (indirekt) weitere Regeln auslösen, wenn durch die Aktionsausführung ein Ereignis eintritt, das eine andere Regel auslöst. Durch das gegenseitige Anstoßen der Regeln können Zyklen entstehen. Ein solcher Zyklus kann dazu führen, dass die Regelausführung nicht terminiert. Dieses Problem wird in der Literatur unter dem Begriff Terminierung behandelt [Wei97]. Die Praxis geht eher pragmatisch damit um, bspw. in Form von Rekursionstiefen-Begrenzungen.

4.2.3 Problem-Auflösung

Das Grundkonzept unseres Feedback-orientierten Autonomic Tuner kann vereinfacht auch beschrieben werden als das kontinuierliche Analysieren von Ereignissen in Echtzeit auf der Basis von benutzerdefinierten Regeln und das entsprechende Ausführen von (Folge-)Aktionen. Durch eine **Event Subscription** können die Komponenten des autonomen Systems auf auftretende Events mit einer Abfolge von Aktionen reagieren.

Wenn bspw. der überwachte Server unerwartet herunterfährt, möchte man am Bildschirm oder auch per Pager darüber informiert werden.

Als **Tuning-Plan** bezeichnen wir einen auf die Lösung eines bestimmten Problems bzw. einer Problemklasse zugeschnittenen, regelmäßig, einmalig oder Event-getriggert ausgeführten Workflow [Ehl07, RWRA08, WRRA08]. Dessen Aktivitäten setzen atomare (Datenbank-)Tuning-Aufgaben um (prozedurales Wissen). Er lässt sich in einen Header, welcher die Metadaten des Tuning-Plans enthält sowie einen den eigentlichen Tuning-Algorithmus enthaltenden Body aufteilen.

Die im Tuning-Plan-Header enthaltenen Metadaten können u.a. zur Laufzeit dazu genutzt werden, den optimalen Tuning-Plan zur Lösung eines bestimmten Problems zu ermitteln. Sie lassen sich nach [Rab11, WiRa07] aktuell in folgende Kategorien einteilen:

- *Statische* Verwaltungsdaten: Titel, Beschreibung, Autoren, Erstellungs-/ Änderungsdatum, Dokumentation sowie die Tuning-Absicht (semantische Annotation des Tuning-Plans)
- Auf Statistiken basierende *dynamische* Ausführungsdaten: Ausführungsdauer, bisherige Anzahl der Ausführungen, Ausführungskosten, Auswirkungen der Ausführung auf die Ressourcen etc.
- Assoziierte korrelierte Events (Event Subscription)
- Input-/Output-Parameter: erlauben die Nutzung von Platzhaltern und damit mehr Dynamik innerhalb des Tuning-Workflows durch Parametrisierung
- Abhängigkeit zu anderen Tuning-Plänen (z.B. bestimmte Ausführungsreihenfolge, Seiten-Effekte)
- Assoziierte Sensoren/Effektoren: Angabe der Ressourcen und der Schnittstellen, auf die lesend bzw. schreibend zugegriffen wird
- Rollback-Informationen: Im Falle von umkehrbaren Aktionen können die Kompensationsbefehle hinterlegt werden.

Für die Umsetzung des Tuning-Plan-Bodys kann eine beliebige Programmiersprache eingesetzt werden [Ehl07, Kar08]. Wir beschränken jedoch die Komplexität bei der Erstellung von Tuning-Plänen durch Konzeption und Bereitstellung einer Menge von vordefinierten Datenbank-Tuning-spezifischen Schritten, die der DBA mit Hilfe von Kontrollstrukturen zu einem komplexen Tuning-Ablauf verknüpfen kann [Rab11, WRRA08].

Neben der Wiederverwendbarkeit einzelner Workflow-Aktivitäten in unterschiedlichen Tuning-Plänen spielt das Abstraktionsniveau einzelner Workflow-Aktivitäten eine wichtige Rolle. Beide Faktoren haben direkten Einfluss auf die Benutzerfreundlichkeit der Tuning-Plan-Modellierung. Die Aktivitäten der Tuning-Pläne lassen sich nach dem Grad ihrer Wiederverwendbarkeit in zwei Kategorien einordnen: (1) globale, d.h. Tuning-Plan-übergreifende Aktivitäten, welche typische Datenbank-Tuning-Aufgaben kapseln und (2) lokale, d.h. für einen bestimmten Tuning-Plan erstellte, Aktivitäten.

Die globalen Workflow-Aktivitäten lassen sich entsprechend dem Abstraktionsniveau und der Sichtweise weiter kategorisieren. Um die Übersichtlichkeit zu wahren und die Integration neuer Schichten in die ATE-Infrastruktur zu vereinfachen, schlagen wir die Kapselung von Tuning-Schritten in entsprechende für jede Systemkomponente zu definierende Schnittstellen vor [RWRA08]. Die Tuning-Schritte setzen hierbei jeweils die Sensor- und Effektor-Funktionalitäten der Schnittstellen um und ermöglichen sowohl lesenden als auch schreibenden Zugriff auf ATE-externe Komponenten (Betriebssystem,

Applikationen), den ATE selbst sowie Interaktionen mit den DBA. Nachfolgend werden die für unser System identifizierten **Schnittstellen** exemplarisch (und ohne einen Versuch der Generalisierung auf beliebige System-Umgebungen) beschrieben:

- Windows (WIN): Tuning-Schritte zum Lesen und Setzen von Umgebungsvariablen bzw. Lesen und Schreiben von Dateien; Ausführen von Skripten; etc.
- IBM DB2 Universal Database (DB2): Absetzen von DB2-System-Befehlen (z.B. DB2START/DB2STOP, DB2ADVIS, DB2EXPLN, DB2SET), DB2-CLP-Befehlen (z.B. UPDATE DBM/DB CFG, RUNSTATS/REORG/REBIND) oder auch DB2-SQL-Anweisungen (z.B. CONNECT, COMMIT; Anlegen von bzw. Zugreifen auf User-Defined Functions und Stored Procedures; DML-Statements sowie DDL-Statements).
- Datenbank- und Anwendungsadministratoren (DBA): Tuning-Schritte zur Einbeziehung von Administratoren (z.B. E-Mail-/SMS-Benachrichtigung, Aufforderung zur Handlung).
- IBM DB2 Performance Expert (PE): Zugriff auf den in unserer Architektur eingesetzten und im Abschnitt 5.3.3 vorgestellten Performance Monitor (z.B. Performance-Indikatoren auslesen, Konfigurieren/Anstoßen des Exception Processing).
- ATE: Zugriff auf Metadaten des ATE und Konfiguration seines Tuning-Verhaltens (z.B. Modifizieren der Events und Tuning-Pläne bzw. der Zuordnungen).

Während die vorgestellte Schnittstellen-Sicht die Weiterentwicklung des ATE unterstützen soll, erscheint es aus Sicht des ATE-Benutzers und im Sinne der Tuning-Plan-Formalisierung sinnvoll, die Tuning-Schritte nach ihren Anwendungsgebieten zu kategorisieren, pro Anwendungsgebiet dedizierte Workflow-Aktivitäten bereit zu stellen und dem Administrator entsprechend zu präsentieren[RWRA08]. Eine derartige **Kategorisierung** ist im Folgenden angegeben:

- **Monitoring-Schritte** zum Sammeln von Überwachungsdaten aus verschiedenen Quellen bzw. zum Anstoßen von Datensammlungs-Prozessen. Beispiele sind:
  - Zugriff auf den Datenbank-Katalog
  - Monitoring-spezifische SQL-Statements
  - Zugriff auf (Snapshot- oder Event-)Monitoring-Daten (bereits gesammelt oder durch unmittelbare Datenanstoßung)
  - (De-)Aktivieren vom Logging/Tracing sowie Zugriff auf die Log-/Trace-Dateien
  - Aufruf von DB2-internen (db2eva, db2diag, db2pd) bzw. -externen (iostat, perfmon) Tools
  - Zugriff auf verschiedene Datenquellen wie IBM DB2 Performance Expert bzw. Tivoli Monitoring for Databases bzw. Betriebssystem-Tools
  - Anweisungen zur temporären bzw. langfristigen Speicherung der Daten (z.B. CREATE TABLESPACE, CREATE TABLE, …)
- **Datenbank-Design-Schritte** zum Erstellen/Löschen von Tuning-spezifischen Datenbank-Objekten (z.B. Indexe, materialisierte Sichten, Partitionen, Tablespaces, Bufferpools); Aufruf der Explain Facility sowie Verwertung ihrer Ergebnisse; Aufruf des DB2 Design Advisor.
- **Konfigurations-Schritte** zum (Re-)Allokieren existierender Ressourcen (z.B. Sortheap, Bufferpool, Lock-Liste, Package Cache etc.) bzw. (De-)Aktivieren von

Mechanismen (z.B. Automatic Statistics Collection) durch Zugriff auf die Konfigurationswerte.
Z.B. lesender/schreibender Zugriff auf die Umgebungsvariablen bzw. die Datenbank- sowie Datenbankmanager-Konfiguration; Operatoren zum komfortablen absoluten bzw. relativen Ändern der Konfigurationswerte; Aufruf des DB2 Configuration Advisor.

- **Schritte zur Entscheidungsunterstützung** zum Initiieren der Sammlung bzw. Interpretation/Mining von Daten zur Einbindung der Entscheidungsunterstützungs-Funktionalitäten in die Tuning-Pläne und ggf. der Verwendung ihrer Ergebnisse in Kontrollstrukturen.
  Z.B. Einbindung der Trendanalyse bzw. der Workload-Klassifikation.
- **Maintenance-Schritte** für typische Datenbank-Verwaltungsaktivitäten wie RUNSTATS, REORG, LOAD/IMPORT/EXPORT, BACKUP/RESTORE; DB2-Systemkommandos wie db2stop/db2start; sowie verwaltungsspezifische Kommandos wie CONNECT/TERMINATE, COMMIT/ROLLBACK.
- **Schritte zur Interaktion mit Nutzern** zum Informieren der Administratoren über die vorgenommenen Änderungen oder aufgetretenen Probleme bzw. Einbeziehen der Administratoren in die Tuning-Abläufe, die nicht automatisch durchgeführt werden können (beispielsweise bei möglicherweise Semantik-verändernden Operationen wie dem Ändern des DB2 Isolation Level). Der Mensch gilt vor allem in einem automatisierten System als letzte und (vertrauenswürdigere) Entscheidungsinstanz.
  Z.B. Bereitstellen von voraggregierten, Problem-beschreibenden Daten dem DBA; Auffordern des DBAs zur Bestätigung von Tuning-Aktionen.
- **Schritte zur Interaktion mit Nutzeranwendungen** zum Aufrufen von Tools und Skripten, deren Semantik dem ATE unbekannt ist.
  Z.B. DB2-externe Tools, Skripte und Services sowie DB2-interne Aufrufe von Stored Procedures bzw. User-Defined Functions.
- **Schritte zum Zugriff auf die ATE-Metadaten** ermöglichen es, auf die Metadaten aller ATE-Komponenten sowie Metadaten ihrer Objekte (Events, Tuning-Pläne. Log-Dateien) zuzugreifen.
- **ATE-Rekonfigurations-Schritte** sind sog. Meta-Schritte zum Rekonfigurieren des ATE-Systems in Abhängigkeit von der Ausführungshistorie der Tuning-Pläne, der vergangenen und der aktuellen Workload-Information sowie anderen den Systemzustand repräsentierenden Daten.
  Z.B. Anpassen von PE-Schwellwerten bzw. ACT-Korrelationsregeln; (De-)Aktivieren von Tuning-Plänen und entsprechenden Events; Beeinflussen der Planungslogik; Anpassen von Ressourcen-Restriktionen (siehe Abschnitt 8.3).

Durch die Verwendung von vordefinierten atomaren Tuning-Schritten bei der Definition von Tuning-Abläufen bekommt der ATE Informationen über die Semantik der einzelnen Tuning-Pläne. Diese automatisch abgeleiteten Informationen sowie vom Nutzer vorgegebenen Annotationen können anschließend bei der Planung bzw. Ausführung der Tuning-Pläne berücksichtigt werden [Rab11].

Die Abbildung zwischen den Tuning-Schritt-Kategorien und den Schnittstellen ist in einigen Fällen nicht trivial. In **Tabelle 4.1** wird diese Zuordnung daher zusammenfassend dargestellt. Ein Plus in der Zelle von Maintance und DB2 bedeutet bspw., dass sich die

Tuning-Schritte der Kategorie *Maintenance* durch Funktionalitäten der *DB2-Schnittstelle* umsetzen lassen.

Schnittstelle / Kategorie	WIN	DB2	DBA	PE	ATE
Monitoring	+	+		+	
Konfiguration	+	+			
Entscheidungsunterstützung	+	+		+	+
Datenbank-Design		+			
Maintenance		+			
Nutzer-Interaktion			+		
Tool-Interaktion	+	+		+	
ATE-Metadaten-Zugriff				+	+
ATE-Rekonfiguration				+	+

Tabelle 4.1: Zuordnung von Tuning-Schritt-Kategorien zu Komponenten-Schnittstellen

Wie schon aus der Beschreibung der einzelnen Kategorien ersichtlich ist, können Tuning-Schritte Daten sammeln bzw. auswerten, was **Datencontainer** (Variablen sowie komplexe Datenstrukturen, wie beispielsweise temporäre Tabellen) unabdingbar macht. Des Weiteren werden Workflow-typische **Kontrollstrukturen** wie Schleifen, Verzweigungen und Zeitkonstrukte (wie z.B. Warten auf ein Event, Warten auf bestimmte Daten, o.ä.) benötigt, um einzelne Tuning-Schritte in Abhängigkeit von den Datenauswertungsergebnissen auszuführen bzw. zu überspringen [AaHe02, JBS97].

In einigen Fällen können auch Aktionen auf Objekten hervorgerufen werden, die nicht aktiv sind (z.B. versuchter Insert in eine Tabelle eines heruntergefahrenen Servers). Daher ist ein Mechanismus erforderlich, solche Situationen (mittels spezieller Events) zu erkennen, die Aktionen zu speichern und sobald wie möglich auszuführen.

Es bleibt zu erwähnen, dass obwohl die Tuning-Pläne eigene Kontrollstrukturen beinhalten können, sie dennoch keine eigenen MAPE-Schleifen umsetzen sollen. Vielmehr sollen diese iterativ, durch Events angestoßen, dem allseits bekannten Tuning-Prinzip folgend jeweils nur eine Änderung am System zu einem Zeitpunkt vornehmen, um die Wahrscheinlichkeit des Auftretens von Seiten-Effekten sowie Deutungs- und Zuordnungsproblemen zu reduzieren.

4.3 Ein erweitertes Wissensmodell

Das Wissen, das zur autonomen Verwaltung von Ressourcen notwendig ist, bildet einen zentralen Bestandteil eines jeden sowohl reaktiv als auch proaktiv agierenden Autonomic Tuners. Dieses gemeinsame, geteilte Wissen steht sämtlichen Komponenten in allen vier Phasen des MAPE-Kreislaufes zur Verfügung und repräsentiert ein formales Verständnis für den Problem-Raum. Die Phasen der MAPE-Schleife tauschen akkumuliertes Wissen und Daten (über eine Wissens-Basis) aus, um die Funktionalität des Regelkreises zu gewährleisten.

Das Wissen umfasst, wie bereits in Abschnitt 4.1 beschrieben, u.a. Informationen, Daten und Metadaten über das zu überwachende System, Log-Daten vergangener Aktionen und Probleme, aber auch Metriken und die definierten Symptome sowie Regeln und Richtlinien, wie in bestimmten Situationen zu verfahren ist. Üblicherweise unterscheidet

man die Wissens-Bereiche Topology Knowledge, Policy Knowledge und Problem Determination Knowledge [Mil05b].

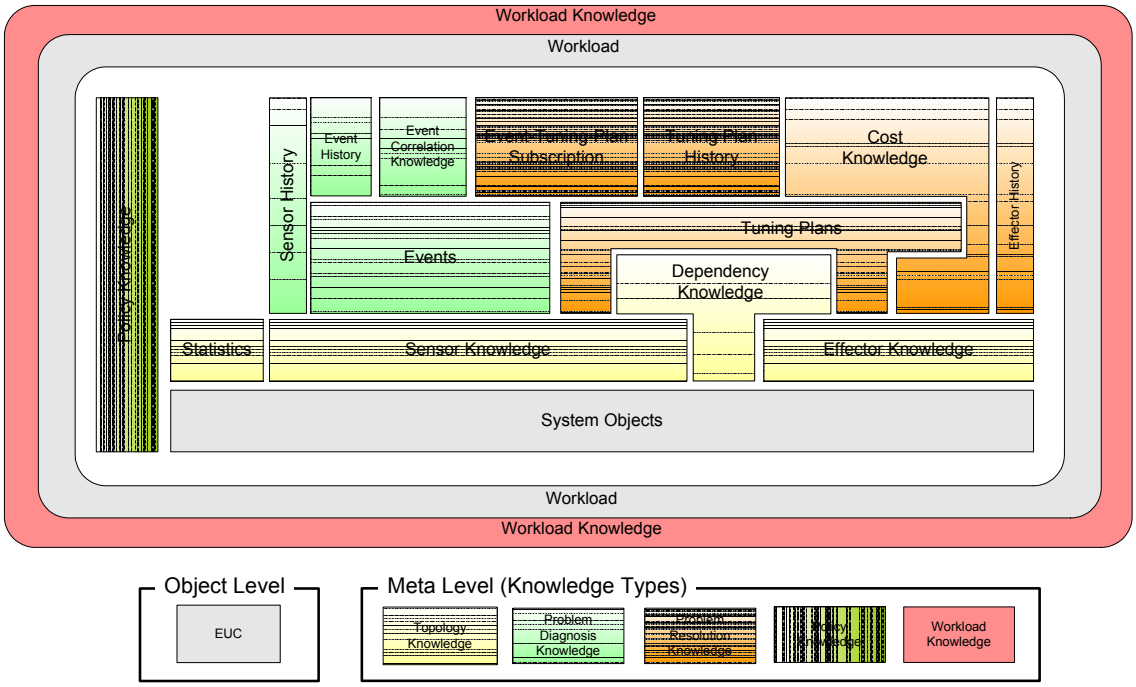


Abbildung 4.3: Ein erweitertes Wissensmodell für das autonome Tuning [WiRa09]

Um sowohl die in den letzten Abschnitten eingeführten Performance-Daten bzw. Metadaten geordneter zusammenzufassen und in Beziehung zu bringen, als auch einen Ausgangspunkt für die Konstruktion und Implementierung eines MAPE-orientierten Autonomic Tuner zu haben, schlagen wir eine detaillierte Verfeinerung der gängigen drei Wissens-Säulen im Rahmen unseres erweiterten, Komponenten-basierten **Wissensmodells** vor [WiRa07, WiRa09].

Im Unterschied zu der typischen Dreiteilung des Wissens unterscheiden wir die zu überwachende Umgebung (environment under control) und fünf Meta-Ebenen, die jeweils verschiedene Wissens-Bereiche enthalten. Dazu zählen Topology Knowledge, Problem Diagnosis und Problem Resolution Knowledge sowie Policy Knowledge und Workload Knowledge.

Abbildung 4.3 stellt die aufeinander aufbauenden Schichten inklusive ihrer verschiedenen Teil-Blöcke unseres Wissensmodells dar. Von unten nach oben betrachtet, steigt die Spezialisierung des Wissens. Generisches und vielfach wiederverwendetes Wissen befindet sich eher in den unteren Ebenen, während das spezifische, wenig wiederverwendbare Wissen weiter oben angesiedelt ist. Jeder einzelne Wissensblock reichert das Wissen, auf dem er beruht, anhand einer gewissen Zielvorstellung und einem spezifischen Anwendungsbereich, weiter an.

Unser Schichtenmodell stellt deutlich heraus, welche Wissensbausteine zum autonomen Tuning benötigt werden und in welchen Beziehungen sie stehen, lässt aber die Details einer konkreten Implementierung offen. Es bildet demgemäß eine Basis zur Entwicklung einer erweiterten Architektur zur Sammlung, Aufbereitung, Verwaltung, Speicherung und Anwendung von (Performance- bzw.) Metadaten sowie zur automatischen Problem-Erkennung und -auflösung.

Die Grundlage dieser Wissensbasis bildet die Objektebene („**Environment under Control**“, EUC), welche sowohl die Workload als auch das zu tunende System mit all seinen Ressourcen und den Primärdaten repräsentiert. Alle anderen Schichten bauen auf der Objektebene auf und stellen Wissen über spezifische Bereiche des Systems in seiner Umgebung und über das Tuning der einzelnen Komponenten heraus.

Eine der wesentlichen Anforderungen an ein autonomes System ist das Erfordernis alle (eigens initiierten bzw. fremdveranlassten) Änderungen der Umgebung, des Systems und der Daten zu verfolgen. Erst die Protokollierung der Aktionen und des Kontext erlaubt über (Miß-)Erfolg bzw. Effizienz der Tuning-Aktionen zu urteilen und diese Ergebnisse in der Zukunft bspw. bei der Priorisierung auszunutzen. Daher existiert in nahezu jeder der nachfolgend vorgestellten Schichten ein entsprechender Block bzw. ein Repository über die bzw. mit den protokollierten, historischen (Meta-)Daten/Aktionen. Änderungen an den System-Ressourcen bspw. können indirekt über eine Effektor-/Sensor-Historie oder auch direkt über die **Protokollierung** von Tuning-Aktionen und Änderungen in den Workload-Charakteristika erfasst werden.

Das **Workload Knowledge** umfasst sämtliches Wissen zur Bestimmung, Klassifikation und Vorhersage der Last auf dem System. Die Workload kann als maßgeblich beeinflussender, allumfassender Faktor auf das ganze System(verhalten) gesehen werden, ohne die kein kontinuierliches und adaptives Tuning notwendig wäre. Sie ist nicht direkt beeinflussbar und häufiger Änderung unterworfen. Unmittelbare bzw. indirekte Auswirkungen können über spezielle Workload-Sensoren oder auch die gängigen Performance-Indikatoren ausgelesen werden. Der performante Umgang mit der Last ist abhängig von Systemcharakteristika, dem Datenvolumen, den Datenstrukturen, aber auch durch die vorgenommenen Tuning-Maßnahmen. In Abschnitt 7.4 demonstrieren wir potentielle, über die OLAP-OLTP-Differenzierung hinaus gehende, Klassifikationskriterien und Möglichkeiten zur *System- und Tuning-unabhängigen* Bestimmung der Workload auf dem System. Neben den Klassifikationskriterien und -regeln enthält das Workload Knowledge noch Klassen-spezifische Schwellwerte und Parametrisierungen sowie die beobachteten und erwarteten Effekte bzw. Abhängigkeiten auf dem System sowie die relevanten Sensor-Elemente. Da für jede Workload-Klasse eine Reihe an Metadaten gespeichert werden muss, empfiehlt es sich zudem, die Zahl der Klassen gering und deutlich unterscheidbar zu halten. Je feiner die Klassenabstufung, desto schwieriger wird die Einordnung und Unterscheidung zur Laufzeit.

In die Kategorie des **Policy Knowledge** fallen sämtliche (von Experten oder auch dem System erstellten bzw. angepassten) Regeln und Richtlinien zur Steuerung des Entscheidungsprozesses eines autonomen Systems, indem diese angeben, welche Ressourcen zu überwachen und ob und wie Änderungen am System vorzunehmen sind (siehe Abschnitt 4.2.1). Je nach Granularität werden die Performance-Ziele entweder durch anzustrebende Werte bzw. Wertebereiche (fuzzy values) für die wichtigen Performance-Indikatoren (z.B. Bufferpool Hit Ratio, Anzahl an Sort Overflows), oder aber für abgeleitete bzw. berechnete, High-Level-Metriken (z.B. Durchsatz, Antwortzeit) vorgegeben.

Eine entscheidende Charakteristik autonomer, selbstverwaltender Systeme und Grundlage für viele Entscheidungsprozesse ist das Bewusstsein über Aufbau und Wirkungsweise ihrer (dynamischen) Umgebung sowie ein allgemeines Ressourcen-Verständnis. Dazu gehören Informationen über die einzelnen Ressourcen, ihre Abhängigkeiten untereinander sowie Möglichkeiten zur Beeinflussung deren Verhaltens. Das **Topology Knowledge** beinhaltet Performance-relevante Informationen über die einzelnen verwalteten Ressourcen, den Status sowie das Verhalten, deren Aufbau, (qualitative und quantitative) Abhängigkeiten untereinander, Installation und Konfiguration sowie über deren Verbindungen zur Umwelt.

Zum Topology Knowledge zählen wir neben den Katalogstatistiken *(Statistics)* über die Verteilung und Struktur der Primärdaten und den Möglichkeiten zur Bestimmung des Zustands bzw. Verhaltens über Sensoren *(Sensor Knowledge)*, auch die Mechanismen bzw. Schnittstellen zur Anpassung bzw. Konfiguration der Ressourcen mittels Effektoren *(Effector Knowledge)*.

Das Topology Knowledge ist die essentielle Basis für die autonome Erkennung, Diagnose und Problem-Auflösung und wird somit von allen anderen Wissensblöcken verwendet. Ursachen für inakzeptable Performance können vielfältig und häufig nicht offensichtlich sein. Beziehungen und Abhängigkeiten unter den Ressourcen sind i.d.R. komplex, wenig verstanden und nur mühevoll manuell ergründbar, modellierbar und quantifizierbar [Ben05]. Aus diesem Grund erscheint uns ein Ressourcen-Modell unentbehrlich. Das *Dependency Knowledge* unterscheidet zwischen drei verschiedenen Arten von qualitativ und quantitativ beschreibbaren Abhängigkeiten zwischen den Ressourcen und zwischen ihren Sensoren und Effektoren. Im Rahmen der Vorstellung unseres Ressourcen-Modells in Abschnitt 7.5 gehen wir ausführlich auf die Details ein.

Das **Problem Diagnosis Knowledge** enthält Informationen über Best-Practices-Methoden der Administratoren zum Erkennen, Diagnostizieren und Interpretieren von Performance-Problemen sowie zum Finden von Ursachen mittels aktueller und vergangener Sensor-Daten und dem Wissen über Ressourcen-Abhängigkeiten (siehe Abschnitt 4.2.2). Die kritischen Situationen (z.B. „ungesunde“ Werte-Kombinationen, unerwünschte Änderung des Systemzustands), die einer automatischen Reaktion bedürfen, werden mittels atomarer bzw. korrelierter Events beschrieben.

Das **Problem Resolution Knowledge** beschreibt Aktionen bzw. Folgen von Aktionen, die notwendig sind, um die erkannten Performance-Probleme zu beheben (siehe Abschnitt 4.2.3). Mittels der Event-Tuning-Plan Subscription (einer n:m-Beziehung zwischen Events und Tuning-Plänen) können Tuning-Pläne in Reaktion auf vergangene, aktuelle oder bevorstehende Probleme ausgeführt werden. Die in den Tuning-Plänen enthaltenen Tuning-Aktionen können dabei auf Sensoren und das Dependency Knowledge zugreifen um ihre entsprechenden Änderungen über die Effektoren zu bestimmen. Die Anwendung einzelner Effektoren (und damit auch von Tuning-Plänen) verursacht für einen bestimmten Nutzen (z.B. Zeitgewinn) gewisse (erlernte bzw. geschätzte) Kosten (z.B. im Sinne von Anzahl an Operationen, CPU, I/O, Speicher etc.) und kann zu Seiten-Effekten auf andere Ressourcen (bzw. Sensoren) führen. Mit Hilfe eines *Kosten-Nutzen-Modells* können sowohl der (Gesamt-)Nutzen als auch die (Gesamt-)Kosten der einzelnen Effektoren und der Tuning-Pläne im Allgemeinen ermittelt werden. Diese Meta-Informationen über die Effektoren bzw. die Tuning-Pläne können durch den Menschen vorgegeben sein oder zur Laufzeit berechnet bzw. aufgrund von Erfahrungen ermittelt/erlernt werden. Basierend auf den vorberechneten Kosten der Ausführung von Tuning-Plänen kann die Plan-Komponente eine Priorisierung vornehmen und die Problemlösung mit den geringsten Kosten auswählen. In einigen kritischen Fällen akuten Handlungsbedarfs kann es jedoch auch sinnvoll sein, unabhängig von den Kosten, den Tuning-Plan mit dem höchsten Nutzen in der kürzesten Zeit zu wählen.

4.4 Nutzung von Wissen im autonomen Tuning

Das Wissen im autonomen Datenbank-Tuning wird benötigt, um eine MAPE-basierte Infrastruktur zu entwickeln, die das automatische Erkennen bzw. Vorhersagen von Performance-Problemen, das Auffinden derer Ursachen sowie das letztliche Auflösen der

Probleme ermöglicht. **Tabelle 4.2** stellt den vier MAPE-Phasen die jeweiligen anwendbaren Bereiche des Wissensmodells gegenüber.

MAPE-Phase	Wissensmodell-Block
Monitor	Sensor Knowledge, Sensor History, Workload Knowledge, Policy Knowledge
Analyze	Events, Event Correlation Knowledge, Event History, Workload Knowledge, Policy Knowledge
Plan	Statistics, Dependency Knowledge, Tuning Plans, Event-Tuning-Plan Subscription, Sensor History, Tuning Plan History, Cost Knowledge, Decision Logic, Workload Knowledge, Policy Knowledge
Execute	Tuning Plans, Tuning Plan History

Tabelle 4.2: Abbildung der Wissensmodell-Blöcke auf die MAPE-Phasen

Der Hauptfokus des eingeführten Wissensmodells liegt in der extensiven Nutzung bzw. Verknüpfung der identifizierten Metadaten zur Planung und Entscheidungsfindung. Hierbei sollen sowohl die autonome Analyse- und Plan-Phase unterstützt, als auch die Administratoren beim manuellen Tuning und bei der Definition von Policies entlastet werden. In den kommenden beiden Unterabschnitten geben wir einen kurzen Einblick in für uns relevante, mögliche *Tuning-Plan-spezifische* Anwendungsbeispiele und -szenarien der Metadaten, die darüber hinaus im Verlauf der Arbeit noch aufgegriffen und vertieft werden.

4.4.1 Unterstützung der autonomen Analyse und Planung (Run-Time)

Insbesondere zur Unterstützung der Plan-Phase sind die anschließenden Einsatzmöglichkeiten denkbar:

- **Schaffung einer Tuning-Plan-Auswahl und -Priorisierung zur Laufzeit**
  Wenn für ein und dieselbe Event-Konstellation mehrere Tuning-Pläne zur Auswahl stehen, kann zwischen diesen basierend auf den Kosten/Nutzen, dem aktuellen Systemzustand, der ermittelten Last auf dem System, den Policies oder auch den Ressourcen-Abhängigkeiten eine Entscheidung getroffen werden. Dies umfasst unter anderem das Auswählen/Ausschließen und Ausführen eines Subsets bzw. einer Permutation von Plänen sowie das Auswählen/Ausschließen von Pfaden in Plänen und auch das Ausschließen von einem Event zugeordneten Plänen ohne Mehrwert/Nutzen. Ausgehend von einer Relevanz der Reihenfolge der Plan-Ausführung, können die zur Ausführung bereiten Pläne für einen maximalen Nutzen zudem priorisiert angeordnet werden.
- **Ermittlung von kooperativen Tuning-Plänen**
  Es gibt Konfigurationsparameter, die erst dann wirksam werden, wenn auch ein oder mehrere andere Parameter bestimmte Werte annehmen. Wenn also ein Tuning-Plan das Ändern eines solchen Parameters, welches erst durch das bestimmte Setzen anderer Parameter wirksam wird, beinhaltet, so muss es kooperative Tuning-Pläne geben, die diese anderen Parameter ändern. Diese kooperativen Tuning-Pläne sollen ermittelt werden können, um die Wirkung des Ausgangs-Tuning-Plans zu gewährleisten.

- **Bestimmung von neuen (Sub-)Plänen**
  Erzeugen von Plänen für Events, die bisher noch keinen assoziierten Plan haben oder Pläne haben, die das durch das Event dargestellte Problem erfahrungsgemäß nur unzureichend lösen.
- **Komposition von (Super-)Plänen**
  Pläne, die immer zusammen ausgeführt werden sollten und kooperativ an der Erfüllung der Policies arbeiten, können bestimmt und in einem (Super-)Plan gruppiert werden.
- **Ermitteln/Abschätzen von geeigneten Werten**
  Bevor ein Tuning-Plan zur Ausführung kommt, stellt sich die Frage, welchen neuen Wert der zu ändernde Parameter annehmen soll. Um dabei eine Anwendung irgendeines Näherungsverfahrens zu vermeiden, soll eine möglichst genaue Abschätzung des neuen Parameterwerts vorgenommen werden können.
- **Ermittlung der Auswirkungen auf andere Datenbank-Objekte** (Seiten-Effekte)
  Wenn ein Tuning-Plan zur Ausführung kommt, wäre es naiv anzunehmen, dass er nur Auswirkungen auf die Performance des Datenbankobjekts hat, an dem er Änderungen vornahm. Ein Tuning-Plan nimmt zumeist ein Anpassen einer oder mehrerer Effektoren einer Ressource vor. Diese Effektoren können aber auch in Beziehung zu Sensoren (und auch Effektoren) anderer Ressourcen stehen, auf denen wiederum weitere Events definiert sein können. Es gehört zur Kontrolle der Wirkung eines Tuning-Plans daher auch dazu, durch Traversierung der Abhängigkeiten herauszufinden, bei welchen Datenbank-Objekten sich die Performance in welchem Maße verändert (u.U. verschlechtert) bzw. welche (vermeintlich unabhängigen anderen) Events unmittelbar nach Anpassen des Effektors aktiv werden, um die Wirkung im Voraus abschätzen zu können.
- **(Kontext-abhängige) Event- und Tuning-Plan-Adaption (Parametrisierung)**
  Denkbar sind bspw. Events mit dynamischen, Workload-abhängigen Schwellwerten. Sinnvoll erscheint auch die Ad-hoc-Instantiierung von konkreten Tuning-Aktionen (z.B. Parameter-Werte/Schrittweiten) innerhalb von Tuning-Plänen zur Laufzeit in Abhängigkeit von der Ressourcenausnutzung, der Last auf dem System und der Auswertung weiterer Laufzeit-Tuning-Metadaten. Es lassen sich durch Beobachten der Reaktion des Systems auf die Ausführung der Tuning-Pläne über die Sensor- und Event-Historie Überreaktionen und auch Oszillationen vermeiden. Vergrößert sich bspw. der Abstand des Auftretens zwischen verschiedenen Instanzen des gleichen Event-Typs, braucht das Tuning u.U. nicht mehr in so großen Schritten erfolgen. Wenn die Events jedoch unaufhörlich weiter ausgelöst werden, trotz mehrfacher und langwieriger Ausführung von Tuning-Plänen, ist eine Adaption der Pläne oder auch der Schwellwerte durchaus überlegenswert.
- **Lernen und Vorhersagen**
  Historisches Wissen über getriggerte Events, gewählte und ausgeführte Pläne und beobachtete Tuning-Auswirkungen können zur Gewinnung neuen Experten-Wissens und der Vorhersage verwendet werden.

4.4.2 Unterstützung der Administratoren (Build-Time)

Den Administratoren kann Entlastung und Hilfe zukommen durch:

- **Visualisierung, Maskierung, Validierung und Korrektur**
  Das gesammelte bzw. erlernte Wissen kann zur visuellen Unterstützung (z.B. mit einer Art Dependency Visualizer), zur Förderung des Verständnisses (über das System und die Abhängigkeiten) und zur Maskierung der Komplexität bei der Definition von Events bzw. Tuning-Plänen oder auch bei der Evaluation genutzt werden. Darüber hinaus kann zusätzliches Wissen hilfreich sein zum automatischen Erkennen und Korrigieren von falschen bzw. ungünstigen Entscheidungen des DBA und auch der indirekten Konsequenzen seiner Aktionen. Auch konfligierende Tuning-Pläne (z.B. Vergrößerung vs. Verkleinerung des Speichers) könnten so bereits frühzeitig zur Build-Time (Definitionsphase) erkannt und zur Laufzeit vermieden werden.
- **Automatische Policy-Transformation**
  Administratoren definieren Policies typischerweise auf einer abstrakten Ebene. Ein automatisches Mapping von nutzerdefinierten High-Level-Policies auf maschinenles- und -ausführbare Low-Level-Konstrukte sowie ein automatisches Erkennen von Diskrepanzen bzw. Konflikten erscheint vielversprechend.
- **Fragment-Bildung**
  Mittels der Metadaten über Abhängigkeiten zwischen Ressourcen, Events und Tuning-Plänen lassen sich insbesondere im Hinblick auf eine Community zum globalen und gemeinsamen Austausch von Experten-Tuning-Wissen Übertragungsgranulate (Plan Bundle) von „zusammengehörigen" Objekten bestimmen (siehe Abschnitt 9.2.3). So kann es sein, dass Plan 1 einen Plan 2 aufruft oder aber Plan 2 setzt die Ergebnisse von Plan 1 voraus, sodass sich eine einzuhaltende Reihenfolge ergibt. In beiden Fällen sollte man Plan 1 und 2 zusammen übertragen, austauschen und später zur Laufzeit anwenden.

Natürlich können diverse automatische Mechanismen zum Datenbank-Tuning die Tätigkeiten der Administratoren auch zur Laufzeit vereinfachen und bei der oft fehleranfälligen Problem-Analyse und -Auflösung unterstützend beraten bzw. eingreifen.

Die Zeit- und Wissens-intensive Herausforderung ist die Abbildung des Wissens der Administratoren über potentielle Ereignisse, über wirkungsvolle Tuning-Aktionen sowie über Beziehungen zwischen den Event-Mustern und den durchzuführenden Aktionen. Ausführlichere Analysen, detaillierte Betrachtungen und ein Vergleich bestehender Ansätze zur Formalisierung von Tuning-Wissen bzw. -Abläufen finden sich in [Rab11]. Darin wird auch der Versuch unternommen, sowohl ein Modell als auch eine gemeinsame Event- und Tuning-Plan-Definitions-Sprache zu etablieren. Letztere soll es auf einfache Weise ermöglichen, alle potentiellen Events mit Hilfe einer auf unser Szenario zugeschnittenen Event-Algebra sowie die Tuning-Workflows zu spezifizieren und an die Laufzeitkomponenten unserer gemeinsam entwickelten Infrastruktur (siehe Abschnitt 8.3) weiterzuleiten.

Eine Community-Plattform (siehe Abschnitt 9.2.3) würde es Datenbank-Administratoren darüber hinaus ermöglichen, ihr Wissen in strukturierter Form auszutauschen und zu verbreiten. Für eine solche Community-Plattform werden Konzepte zur Verwaltung, zum Austausch und zur Evolution des formalisierten Tuning-Wissens benötigt [Rab11].

Kapitel 5

Gewinnung von Performance-Daten

Wie vorangehend bereits dargestellt, bildet die kontinuierliche bzw. Ereignis-gesteuerte Sammlung, Speicherung und Verwaltung von Performance- und Metadaten über den Zustand sowie das Verhalten des Systems bzw. die Charakteristika und Auswirkungen der Workload die Grundvoraussetzung zur Selbst-Verwaltung. Das Wissen soll aus verschiedenen Daten-sammelnden bzw. -bereitstellenden Quellen gewonnen, mittels einer adäquaten Datenstruktur einheitlich abgebildet und an zentraler Stelle (zugreifbar) verwaltet sowie letztlich autonom weiterverarbeitet bzw. zur manuellen Bewertung durch den Benutzer interaktiv präsentiert und zeitnah ausgewertet werden können.

Einige der Inhalte sind bereits in vorangegangenen Abschnitten angedeutet wurden. Sie sollen in diesem Kapitel veranschaulicht vertieft und in einen verständlichen Gesamt-Kontext gebracht werden. In **Abschnitt 5.1** werden zunächst Voraussetzungen an die Datenquellen sowie die idealen Anforderungen an Sammlung und Gewinnung von Performance-Daten in unserem Kontext eingeführt. **Abschnitt 5.2** gibt einen Einblick in mögliche Verfahrensweisen, Formate und Standards zur Beschreibung der Ressourcen, ihrer Daten und gegenseitiger Interaktionen. Nachfolgend erarbeiten wir in **Abschnitt 5.3** eine weit gespannte Auflistung der gängigsten Performance-Daten-Quellen sowie deren Möglichkeiten zur Datengewinnung im DB2-Umfeld. Daraufhin fassen wir im letztlichen **Abschnitt 5.4** kurz zusammen, mit welcher Methodik bzw. unter welchen Voraussetzungen die Tools idealerweise eingesetzt werden können, um unseren, eingangs in diesem Kapitel, gestellten Anforderungen an die Quellen und die Datensammlung zu genügen.

5.1 Einführung

Das Administrieren, Konfigurieren und vor allem das Optimieren von modernen Informations-Management-Systemen wird mit dem stetig wachsenden Datenvolumen, der zunehmenden Heterogenität und der daraus resultierenden Gesamtkomplexität herausfordernder und arbeitsintensiver. DBAs oder auch auf dem Kontroll-Schleifen-Paradigma basierende autonome Mechanismen initiieren im Falle eines die Verfügbarkeit und/oder Performance gefährdenden Problems umfassende Daten-übergreifende Analysen auf den Performance-Daten. Konkrete Ziele dabei sind u.a., Engpässe in den Ressourcenallokationen und mögliche Problemursachen zu erkennen bzw. Hypothesen aufzustellen, Schlüsse zu ziehen und Entscheidungen über sich anschließende, möglicherweise nicht revidierbare Tuning-Aktionen mit oft weitreichenden und schwer vorhersehbaren Konsequenzen zu treffen.

Folglich sollte bei der manuellen bzw. automatischen Problem-Erkennung und -Diagnose zunächst auf eine adäquate, den Anforderungen und Gegebenheiten angepasste, **Monitoring-Strategie** sowie zugehörige aktuelle und konsistente Datenbestände Wert gelegt werden.

Performance-Problem- und Ursachen-Analysen erfordern die systematische Betrachtung vieler verschiedener Performance-Metriken sowie die Korrelation von Ursache und Wirkung. In der Praxis kommen die für das Performance Management relevanten Daten daher meistens nicht aus einem zentralen System, sondern sind auf verschiedene **heterogene** Subsysteme, oftmals redundant, verteilt (siehe **Abbildung 5.1**). Verschiedenste System-interne bzw. von Drittanbietern angebotene Tools sammeln, visualisieren und speichern die Performance-Daten einzelner Ressourcen bzw. konsolidiert einzelner Ebenen im Software-Stack.

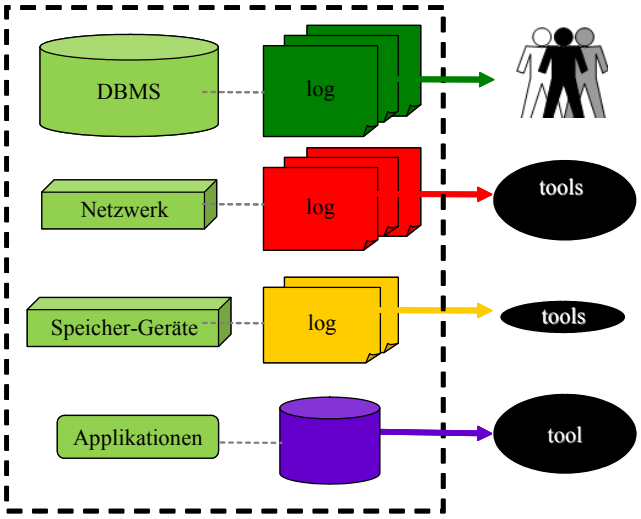


Abbildung 5.1: Variantenreichtum bei der Performance-Daten-Gewinnung

Aufgrund der komplexen Natur eines DBMS in seinem Umfeld kommt es täglich zu einer großen Anzahl von Fehlermeldungen bzw. kritischen Situationen. Gerade in heterogenen IT-Landschaften ist die gezielte Abfrage nach bestimmten Informationen durch die Fülle an vielfältigen Daten, unterschiedlichsten Ursprungs und Formats schwierig realisierbar und fehlerbehaftet. Verschiedene Datenquellen können zudem unterschiedliche Ergebnisse bzw. Sichtweisen auf Daten liefern. Informationsverlust und Widersprüche sind damit nicht ausgeschlossen.

In der Regel liefert kein einzelner Performance-Monitor alle für die Problem-Analyse in einem IT-System notwendigen Daten. Die relevanten Daten aus einer Vielzahl an heterogenen, internen und externen Quellen müssen idealerweise also erst einmal sinnvoll zusammengestellt, bereinigt, geordnet und an zentraler Stelle restrukturiert und konsistent bereitgestellt werden, um eine vernünftige und flexible Analyse sowie die Verknüpfung von Informationen zu erlauben.

5.1.1 Voraussetzungen an die zu verwaltenden Systeme (Datenquellen)

Bevor die gesammelten (oder generierten) Daten effizient **verwaltet** (, zugreifbar **gespeichert**) und auch sinnvoll zur Laufzeit **eingesetzt** werden können, sind einige Voraussetzungen an die zu verwaltenden Systeme zu erfüllen:

- Das Vorhandensein, die Zahl und Eigenschaften der (konkreten) Ressourcen mit ihren Metriken sollen *automatisch*, idealerweise ohne große Verzögerungen und im laufenden Betrieb erkannt sowie regelmäßig zur Laufzeit *aktualisiert* werden (**Automatic Discovery**). Dadurch wird die einfache Erweiterbarkeit ermöglicht, um neue Ressourcen und deren assoziierte Metriken sofort zu überwachen.

- Um eine Automatisierung der Datensammlung sowie eine Automatic Discovery zu ermöglichen, müssen dem Monitoring- bzw. Tuning-System die Architektur- und **Topologie-Daten** des zu verwaltenden Systems zur Verfügung stehen. Diese beinhalten den logischen Aufbau der zu überwachenden System-Elemente, ihre möglichen Verknüpfungen untereinander sowie die Sensor- bzw. Effektor-Schnittstellen zum Auslesen bzw. Verändern ihres Zustands.
- Eine Ressource muss (Sensor-)Schnittstellen zum Zugriff auf ihre Performance-Daten zur Verfügung stellen. Dabei kann es sich um regelmäßig anfallende Performance-Indikatoren (**Metriken**) zur Repräsentation des Status bzw. der Zustands-Änderungen oder aber um Informationen über kritische Problem-Situationen (sog. **Events**).
- Neben den obligatorischen Performance-Daten (über die Gesundheit) des Systems, sind vor allem Laufzeit- und **Ausführungsdaten über das Tuning-System** (z.B. Event- und Tuning-Plan-Historie) zu protokollieren, verwalten und im Sinne eines „Meta-Tuning“ auszuwerten.
- Die verschiedenen Typen von Daten werden **unterschiedlich häufig** generiert und müssen damit auch zu verschiedenen Zeiten, unabhängig voneinander gesammelt bzw. dem Repository zugeführt werden können. Eine Zuführung der Daten aus den Quellen in ein Ziel-Repository soll dabei sowohl automatisch per **Push** oder auf Anfrage per **Pull** realisierbar sein.

5.1.2 Anforderungen an die Datensammlung (und –zuführung)

Ziel der Datensammlung ist die (möglichst automatische) Gewinnung von relevanten Daten **verschiedenster Quellen**, ohne limitiert zu sein auf eine bestimmte, etwaige starre Form der Daten. Dabei erscheinen Mittel und Techniken zur gleichzeitigen Sammlung von Informationen über mehrere Ressourcen und die Betrachtung der folgenden Punkte wichtig:

- Performance kann und sollte auf verschiedenen (organisatorischen) Ebenen gemessen und auch auf diesen wiedergegeben werden können. **Performance-Daten** sind **verteilt** über den ganzen Software-Stack (vom Betriebssystem über das DBMS bis hin zu Daten über die Anwendungen und deren Workload) und können von verschiedensten (externen und internen) Daten- und Informations-Quellen ausgelesen und bereitgestellt werden.
- Die Performance einer Ressource lässt sich typischerweise nicht anhand eines einzelnen Metrik-Werts charakterisieren, sondern vielmehr stochastisch über den **zeitlichen Verlauf** (Historie) und die Werte-Verteilung.
- Die unterschiedlichen Methodiken zur Gewinnung der Laufzeitdaten motivieren zwei **Kategorien** von Performance-Daten und damit auch verschiedene Typen von Monitoren:
    - **Performance-Indikatoren** (Metriken), die per Polling und auf Anfrage den Status einer Ressource widerspiegeln.
    - **Event-Daten**, die per Publish-Subscribe und Pushing automatisch über relevante, System-kritische Situationen „berichten“.
- Dafür ist ein im gesamten System einheitlicher, konfigurierbarer **Event-Verwaltungs-Mechanismus** von Nöten. Events sollten an zentraler Stelle verwaltet, kor-

reliert und automatisch an verantwortliche Entitäten weitergeleitet werden. Für retrospektive Analysen sind daneben alle erkannten Events in einer Historie zusammen mit den zugrundeliegenden bzw. beeinflussten Performance-Metriken in einem Repository zu speichern. Es sind daher Mechanismen (wie bspw. eine API) zur Definition und Korrelation nutzerdefinierter Events bzw. zur Einbindung von beliebigen existierenden, weiterzuleitenden Events sowie zur Notifikation und dem Abonnement zu schaffen. Die Administratoren, aber auch die Tuning-System-Komponenten und die Tuning-Pläne müssen die Möglichkeit haben zu definieren, welche Events von Interesse sind und automatisch (wohin) weitergeleitet werden sollen. Die Überwachung des Systems erfolgt demnach idealerweise in Kombination abgestimmter Polling-Funktionen und Auswertung von Event-Meldungen. Treten kritische Ereignisse auf, übernimmt eine verantwortliche Komponente die Aufgabe, entsprechend zu reagieren bzw. zu alarmieren.

- Die **Datensammlung** sollte **verteilt** geschehen, um eine **gleichzeitige** und **zuverlässige** Sammlung von Informationen über mehrere Ressourcen zu ermöglichen. Ein einzelner, zentralisierter Monitor zur Sammlung der dynamischen Daten kann zu zwei eklatanten Nachteilen führen. Zum einen stellt er einen Single Point of Failure dar, so dass im Fehlerfalle keine Daten gesammelt und zur Speicherung weitergegeben werden können. Zum anderen kann er in Zeiten hoher Last einen Flaschenhals darstellen, wenn in Spitzenzeiten eine Vielzahl von Daten transferiert und gespeichert werden müssen. Daher ist es erforderlich, dass Monitore mehrere Ressourcen überwachen können. Eine Ressource darf prinzipiell aber auch von mehr als einem Monitor überwacht werden.
- Die Komponenten eines komplexen IT-Systems existieren nicht in Isolation, sondern unterliegen Abhängigkeiten bzw. Interaktionen mit anderen (Sub-)Komponenten. Die Isolation eines Subsystems zur vereinfachten Analyse kann zur Vernachlässigung wichtiger Interaktionen mit anderen (Sub-)Systemen und damit zum fehlenden Verständnis beitragen. Gerade weil das Ganze mehr als die Summe seiner Teile ist, darf die Betrachtung des **Gesamtsystems** und seiner **Interaktionen** daher grundsätzlich nicht aus den „Augen“ verloren werden.
- Dabei können die Tools/Monitore durchaus neben der reinen Datenbereitstellung und -sammlung auch eine **(Vor-)Aggregation** der Daten vornehmen und über eine entsprechende Schnittstelle ebenfalls Events in das Tuning-System einspeisen.
- Es ist zu beachten, dass mit der Anzahl der (externen) Daten-Monitore auch der (nicht vollständig kontrollierbare) **Overhead** auf dem System steigt. Gerade bei häufiger Performance-Daten-Sammlung darf der operationale Betrieb in den Daten-liefernden Systemen nicht merklich beeinflusst werden. Leistungsdaten vieler Prozesse und Ressourcen müssen in Echtzeit erfasst, gespeichert und analysiert werden können, ohne dass es zu Performance-Einbußen kommt
- Die Monitoring-Mechanismen sollten im Idealfall über einen **adaptiven Drosselungsmechanismus** verfügen, um nur einen adäquaten (Last- oder Rechenzeit-) Anteil der verfügbaren Ressourcen zu beanspruchen bzw. den Umfang der Performance-Daten einzuschränken oder ggf. auszuweiten. In jedem Fall muss jedoch die durch das Monitoring zusätzlich erzeugte Last anhand geeigneter Performance-Indikatoren überwacht werden.

Im Anschluss an die Sammlung ist im Hinblick auf korrelierte und Ressourcen-übergreifende Analysen eine Integration der Daten in einen (zumindest logisch bzw. virtuell) gesamtheitlichen Datenbestand zu gewährleisten. Es obliegt dabei der konkreten Imp-

lementierung, ob und wie einzelne Teilbereiche physisch separiert werden. Die Datenzuführung ist

- **einfüge-lastig**: Die Performance-Daten werden nicht bzw. selten geändert. Insbesondere die Metriken werden nie verändert und lediglich kontinuierlich in ein Repository eingefügt. Einfüge-Operationen finden demnach weit häufiger statt als Lese-Operationen. Daher sind auf Einfüge- und Aktualisierungs-Operationen spezialisierte Monitoring- und Repository-Lösungen, die nicht ausschließlich zum schnellen Lesezugriff optimiert sind, geeignet.
- einer Forderung nach **geringer Latenz** unterworfen**:** Typischerweise sind Performance-Daten im Problemfall nur für einen kurzen Zeitraum relevant. Daher müssen diese Daten, im Sinne hoher Zugriffsraten, so schnell wie möglich von Quelle (wo sie gesammelt werden) zu Ziel (wo sie benötigt werden) übermittelt werden.

5.2 Beschreibung und Modellierung der Ressourcen, ihrer Daten und Interaktionen

Die Überwachung und Verwaltung (der Ressourcen) einer IT-Umgebung ist seit je her eine schwere und unübersichtliche Aufgabe. Jede Problemdomäne hat ihre eigene Terminologie und Management-Anforderungen. Mit Hilfe eines oder mehrerer **Ressourcen-/Datenmodelle** können die Ressourcen eines Systems sowie ihre Schnittstellen beschrieben und nutzbar gemacht werden. Dabei sind ihre Eigenschaften, Performance-Metriken, potentielle Events, die möglichen Konfigurations- und Änderungs-Operationen und ihre Beziehungen untereinander, im Sinne von quantitativen und qualitativen Abhängigkeiten, von Bedeutung. All diese Informationen über die (Performance-)Daten-Quellen sollten durch ein Modell einheitlich und möglichst adäquat abgebildet sowie maschinenauswertbar zur Verfügung gestellt werden.

Damit die Heterogenität der verschiedenen Komponenten adressiert werden kann, sollen offene, am aktuellen Stand der Technik orientierte **Standards** zur Beschreibung der Struktur, Kommunikation und des Datenaustauschs verwendet werden.

Die nachfolgenden Abschnitte basieren im Wesentlichen auf den Ergebnissen unserer Arbeiten [Alg10, Böt07, Ell10]. Darin haben wir die wichtigsten Industrie-Standards zur Modellierung, Informationsdarstellung und zur Verwaltung von Ressourcen im Umfeld des autonomen Datenbank-Performance-Tuning einzeln vorgestellt, verglichen und deren Rolle sowie möglichen Einsatz anhand kurzer Beispiele erläutert. Wir wollen daher an dieser Stelle nur einen kurzen, exemplarischen Überblick geben und einige im Umfang reduzierte Beispiele aufzeigen.

5.2.1 Standards zur Ressourcen-Modellierung

Zur *Beschreibung von Ressourcen* wurden in [Alg10] u.a. die Standards WSRF (Web Services Resource Framework, [CFF+04]), RDF (Resource Description Framework, [MaMi04]), SNMP MIBs (Simple Network Management Protocol Management Information Bases, [Gue97]) und CIM (Common Information Model, [DMTF99]) hinlänglich diskutiert.

Daneben seien der Vollständigkeit halber noch die folgenden, eher „artfremden“, aber durchaus in unserem Kontext anwendbaren, Ansätze aufgelistet:

- JMX [19](Java Management eXtensions) JSR77 als Ressourcen-Modell für die J2EE-Platform
- UNICORE[20] Resource Schema
- Globus RSL[21] (Resource Specification Language) sowie das GLUE[22] (Grid Laboratory Uniform Environment) Schema
- JSDL[23] (Job Submission Description Language)
- CDL[24] (Configuration Description Language)
- DCML[25] (Data Center Markup Language)

Aufgrund der recht verschiedenen Einsatzgebiete und Zielsetzungen der einzelnen Standards zur Modellierung von Ressourcen und Daten, besitzen die erzeugbaren Datenmodelle eine sehr unterschiedliche Komplexität. Während sich die restlichen Standards aufgrund ihrer Einfachheit besonders in simplen Szenarien einsetzen lassen, ist das objektorientierte *Common Information Model* durch seine Mächtigkeit und die weite Verbreitung auch für die Abbildung sehr komplexer IT-Systeme geeignet.

Das **Common Information Model** (CIM) der Distributed Management Task Force (DMTF) stellt ein objektorientiertes Datenmodell dar, welches die für die Verwaltung erforderlichen Informationen und Funktionen eines IT-Systems (abstrahiert und repräsentativ) beschreibt. Dazu zählen die verwalteten Elemente in einer IT-Umgebung, deren Eigenschaften sowie unterstützte Operationen und die Beziehungen zwischen den Elementen. Durch CIM wird eine einheitliche Anbieter-, Plattform- und Implementierungsunabhängige Management-Schnittstelle ermöglicht, welche die *Struktur* des abgebildeten IT-Systems beschreibt. So können Beschreibungsdaten für Komponenten aus den Bereichen Netzwerk, Betriebssystem, Geräte und Anwendungen verwaltet werden. Die Verwaltung selbst ist allerdings nicht Aufgabe von CIM. Es stellt durch die konzeptionelle Sicht auf die physischen und logischen Komponenten lediglich eine einheitliche Struktur des zu verwaltenden Systems bereit, mit dem Ziel, administrative Aufgaben zu vereinfachen.

CIM liefert eine Sammlung bestehend aus Klassen mit Eigenschaften und Assoziationen, welche eine verständliche Grundstruktur auf konzeptioneller Ebene darstellt, um die Informationen bzgl. des zu verwaltenden Systems organisieren zu können. Die CIM-Beschreibungen liegen prinzipiell in UML-Notation vor. Mithilfe des XML-basierten Managed Object Formats (MOF) lassen sich CIM-Klassen schließlich beschreiben, auswerten und austauschen.

Das vielschichtige CIM-Datenmodell weist mehrere Ebenen auf [Alg10]:

- Das **Core Model** beschreibt einige grundlegende, generalisierte (logische und physische) Objekte, die in allen Bereichen des System-Managements benötigt werden und bildet damit den Grundstein des Common Model und des Managements von Informations-Systemen mit Hilfe von CIM.
- Das **Common Model** kann als Spezialisierung zum Core Model angesehen werden und enthält mehrere Teile, die jeweils auf einen bestimmten Bereich des Managements zugeschnitten sind. Diese Teile bauen auf den Basis-Objekten des Core

[19] http://download.oracle.com/javase/1.5.0/docs/guide/jmx/index.html
[20] http://www.unicore.eu/
[21] http://www.globus.org/toolkit/docs/2.4/gram/rsl_spec1.html
[22] http://www.hicb.org/glue/glue.htm
[23] http://forge.gridforum.org/projects/jsdl-wg
[24] https://forge.gridforum.org/projects/cddlm-wg
[25] http://www.dcml.org/

Models auf und sind unabhängig von der verwendeten Technologie und Implementierung. Während die 26 Klassen des Core Models *generelle* Klassen für das System-Management umfassen, repräsentiert das Common Model einen vollständigen, *konkreten* Rahmen für das Management von Systemen. Anhand der nachfolgenden Schemata versucht es sämtliche Bereiche des System-Managements abzudecken und die Modellierung verschiedener Architekturen zu ermöglichen:

- Das CIM **Physical** Common Model beschreibt Informationen der Hardwarekomponenten, wie z.B. Festplatten, Netzwerk-Karten, Prozessoren.
- Durch das CIM **System** Common Model werden insbesondere Konzepte in Computer-Systemen beschrieben. Dazu zählen bspw. Prozesse/Jobs/ Dienste, Protokolle, Datei-Systeme, Log-Dateien, Betriebssysteme etc.
- Das CIM **Network** Common Model definiert Klassen für die Repräsentation von Netzwerk-Geräten und -Strukturen, z.B. Routing-Beschreibungen, Topologien, SNMP-Integration und Netzwerkfilterung, usw.
- Das CIM **Event** Common Model definiert die Ereignis-Behandlung und dient der Beschreibung von Ereignissen, die in einer CIM-Umgebung ausgelöst werden können (sog. *Indications*).
- Das CIM **Metrics** Common Model beschreibt die Klassen zur Verwaltung und Erfassung der dynamischen Performance-Metrik-Informationen.
- Durch das CIM **Policy** Common Model wird den Anwendungs-Entwicklern, Netzwerk- und Policy-Administratoren die Festlegung und Verwaltung bestimmter Policies, bestehend aus Bedingungen und Aktionen, ermöglicht.
- Mit Hilfe des CIM **Application** Common Model werden Klassen für die Verwaltung der Software-Produkte und Applikationen, ebenso wie Abhängigkeiten von Software-Paketen untereinander und zu bestimmten Betriebssystemen definiert.
- Das CIM **Database** Common Model beschreibt die Modellierung von Datenbank-Umgebungen. Es beinhaltet Informationen zur Datenbank-Software, den Instanzen sowie verfügbaren Diensten und lässt sich durch beliebige Schemata, deren Struktur und Abhängigkeiten klar definiert werden können, erweitern.

- Die **Extension Schemata** werden nicht durch die DMTF festgeschrieben, sondern sind Implementierungs- und Architektur-abhängige Erweiterungen der Hersteller. Sie bauen wiederum auf dem Common Model auf, so dass auch herstellerspezifische Objekte im Rahmen der Modellierung und Beschreibung verwendbar sind.

In die Windows-Betriebssysteme von Microsoft ist CIM bereits seit Windows NT 4 Fixpack 4 integriert und dort unter dem Namen *Windows Management Instrumentation* (*WMI*) bekannt. Über WMI lassen sich konkret Hardwaregeräte, der Betriebssystemkern, die Registry und Netzwerk-Interfaces überwachen und verwalten. Mit Hilfe des Kommandozeilenprogramms „wbemdump.exe" lassen sich Instanzen und Klassen abfragen, Methoden aufrufen und komplexe Anfragen formulieren. Ein WMI-Browser erlaubt die Betrachtung der CIM-Klassenhierarchie. Das darauf aufbauende CIM Studio bspw. bietet die Möglichkeit, auch Instanzen zu bearbeiten. Haupt-Verwendungsmöglichkeit für WMI ist jedoch die

mitgelieferte VisualBasic-Script-API, die es ermöglicht, Script-basierend beliebige Management-Tätigkeiten durchzuführen.

5.2.2 Standards zur Modellierung von Laufzeitdaten, Events und Metriken

Neben Standards zur Modellierung der verfüg- und verwaltbaren Ressourcen spielen Standards zur Beschreibung und Darstellung der Laufzeitdaten über den Zustand bzw. das Verhalten eine bedeutsame Rolle.

Das **CIM Metrics Model** [DMTF99], als Teil des CIM Common Model, unterstützt bspw. die Erfassung und Verwaltung von Performance-Metrik-Werteverläufen für Ressourcen und die Workload.

Von besonderer Bedeutung in komplexen IT-Sytemen sind, wie bereits erläutert, Indikatoren für bestimmte Ereignisse (**Events**), welche ebenfalls den Status verschiedener System-Elemente widerspiegeln, aber primär auf kritische Problem-Situationen hinweisen. Dabei beschreibt ein Ereignis eine bestimmte Situation zu einem bestimmten Zeitpunkt an einem bestimmten Ort. Das Event als Datenstruktur, um Nachrichtendaten einzukapseln und um das Ergebnis einer Situation darzustellen, repräsentiert das Fundament für die Kommunikation dieser komplexen Systeme. Im übertragenen Sinne lassen sich diese Events auch als die elektrischen Impulse des Nervensystems des menschlichen Organismus auffassen (vergleiche Abschnitt 2.2). So wie sie die Kommunikation und Koordination innerhalb des Körpers ermöglichen, ermöglichen Events den Nachrichtenaustausch zwischen Anwendungen in komplexer Informationstechnologie.

Viele Applikationen bzw. Ressourcen erzeugen beim Auftreten von Fehlern bzw. abnormalen Situationen Logfile-Einträge. Das Konzept der Performance-Daten- und Ereignis-Protokollierung ist weit verbreitet, aber der **Variantenreichtum** der Log-Einträge korreliert zumeist stark mit der Anzahl der Komponenten, die diese Einträge generieren. Dabei gibt es jedoch selten ein einheitliches Format, oder Ressourcen-übergreifende Konventionen, welche Informationen in welcher Art in diesen Logfiles gespeichert werden. Gerade in heterogenen, komplexen Systemen wird die Analyse derartiger Daten durch eine solche Uneinheitlichkeit erheblich erschwert. Aus Mangel an Standardisierung und um dennoch eine zentrale Verarbeitung von sich aus unterschiedlichsten Logfiles zusammensetzenden Informationen zu ermöglichen, wurde das so genannte **Common-Base-Event-**Format (CBE) entwickelt [OKSC04].

Die Common-Base-Event-Spezifikation beschreibt ein allgemeines Format für das Logging (Protokollieren) im Autonomic-Computing-Umfeld. Common Base Events ermöglichen es, Informationen über auftretende Ereignisse und Fehler, die von Applikationen bzw. Ressourcen in herkömmliche Logfiles geschrieben werden, in einem einheitlichen Format zu beschreiben und auszutauschen. Das Common Base Event Model wird in XML definiert. Es erfasst alle nötigen Informationen beim Auftreten eines Events in Form eines 3-Tupels:

- Die Identifizierung der Komponente, die das Event berichtet. Dazu kommen u.a. Informationen zum Namen, Typ und zur Ausführungsumgebung der Komponente.
- Die Identifizierung der Komponente, die von dieser Situation betroffen ist. Dies kann mitunter auch die berichtende Komponente sein.
- Die entstandene Situation.

Es gibt noch zusätzliche Informationen, die mit dem Common Base Event assoziiert werden können. Sie enthalten beispielsweise Schlüssel-Attribute, die das Event eindeutig identifizieren oder seine Priorität beschreiben. Es existieren weitere optionale Klassen, die zusätzliche Funktionalität oder herstellerspezifische Anforderungen und auch eine Klassifizierung der Nachrichten in Kategorien unterstützen.

Außer den CBEs gibt es noch viele weitere, leistungsfähige Formate und Standards zur System-Überwachung und -Verwaltung mit breitem Anwendungs-Spektrum. Dazu zählen beispielsweise die Systems Network Architecture (SNA) Alerts, Common Information Model Events/Indications, WS-Notifications oder Java Management Extensions (JMX) [Alg10].

Der Standard *WS-Notification* bspw. definiert den Transport von Events mittels des Publish/Subscribe-Messaging Musters. Publish/Subscribe-Messaging wird verwendet, um eine Nachricht mit einem bestimmten Thema für mehrere interessierte Konsumenten zugänglich zu machen und automatisch über das Auftreten eines bestimmten Events zu benachrichtigen.

Sowohl CBE als auch WS-Notification verwenden XML zur Datenbeschreibung. Darüber hinaus sind CBE und WS-Notification schlecht zu vergleichen, da sie in sehr unterschiedlichen Bereichen eingesetzt werden: Mit CBE wird die Struktur der Events definiert, während WS-Notification die Kommunikationsart zwischen Web Services mittels Event-Benachrichtigungsmechanismen beschreibt. Das CBE ist allerdings so modelliert, dass andere Formate ohne Informationsverlust darauf abgebildet werden können. Der von IBM bereitgestellte **Generic Log Adapter** [IBM05] dient dazu, beliebige (Log-)Formate mit nur geringem administrativen Aufwand automatisiert in das CBE-Format zu konvertieren. Somit stellt die Tatsache, dass nicht alle Event-Quellen nativ ein CBE generieren, nur eine minderschwere Herausforderung dar und CBEs lassen sich leicht in bestehende Systeme integrieren.

Insbesondere für die Sammlung und Überwachung von Informationen über die Performance von Applikationen hat sich die Methode des **Application Response Measurement**[26] (ARM) durchgesetzt. Im Falle einer problematischen Situation werden die relevanten ARM-Daten typischerweise in ein CBE-Event gekapselt und an entsprechende Instanzen weitergeleitet.

Die hier angedeuteten und ausführlich in [Alg10] vorgestellten Standards zur Modellierung von Metriken und Events sind alle im autonomen Datenbank-Performance-Tuning einsetzbar. Sie sind, wie erwähnt, schwer miteinander zu vergleichen, weil sie für verschiedene Einsatzgebiete entworfen wurden. CIM-Metrics ist das Mittel der Wahl, um auf der Datenmodellierungs-Ebene die Metrik-Werte der per CIM beschriebenen Komponenten des zu überwachenden Systems zu definieren. Es ist aber auch denkbar und vielfach üblich, einzelne Metrik-Werte ebenfalls als Ereignis aufzufassen und in ein Event zu verpacken. Somit erspart man sich verschiedene Datenformate und erreicht eine Homogenisierung der Laufzeitdaten(-Verarbeitung). Events können im CBE-Format erzeugt und an die Monitor-Komponente des Autonomic Tuners (siehe Abschnitt 3.3) geschickt werden, der jene sammelt und zum Analysieren an die Analyse-Komponente weiterleitet. Mit WS-Notification bspw. kann die Monitor-Komponente Event-Benachrichtigungen abonnieren, oder es können zwischen der Execute-Komponente des Autonomic Managers und den Effektor-Schnittstellen der Ressourcen Nachrichten ausgetauscht werden.

[26] http://www.opengroup.org/tech/management/arm/

5.2.3 Standards zur Instrumentierung von Systemkomponenten

Damit eine Vielzahl an Systemkomponenten im Zusammenspiel überhaupt durch einen Provider verwaltbar werden, müssen sie neben dem reinen Datenmodell auf Ressource-Ebene auch Schnittstellen und erweiterte Funktionalitäten auf der infrastrukturellen Ebene bieten, über die Verwaltungs- und Kommunikations-Informationen mit der Umgebung ausgetauscht werden können. Daher sind für die Verwaltung sehr komplexer Systeme eher die Standards *WBEM*[27] *(Web Based Enterprise Management)* und *WSDM*[28] *(*Web Services Distributed Management*)* geeignet, weil sie durch ihre flexible Struktur Plattform-, Programmiersprachen-, Laufzeitumgebungs-, Betriebssystem- sowie Protokoll-Unabhängigkeit anstreben. WBEM bedient sich hierfür eines Client-Server-Konzepts, während WSDM das Konzept von Web Services zur Kapselung der Ressourcen verwendet.

Die Erweiterung von CIM zu einem Gesamtframework wurde 1998 auf Initiative der DMTF vollzogen, um eine Interoperabilität auf allen Netzwerkschichten zu erreichen. Diese Architektur wurde auf den Namen **Web Based Enterprise Management (WBEM)** getauft, auch wenn dieser Name ein wenig irreführend ist, da es keinen Zusammenhang mit dem World Wide Web gibt.

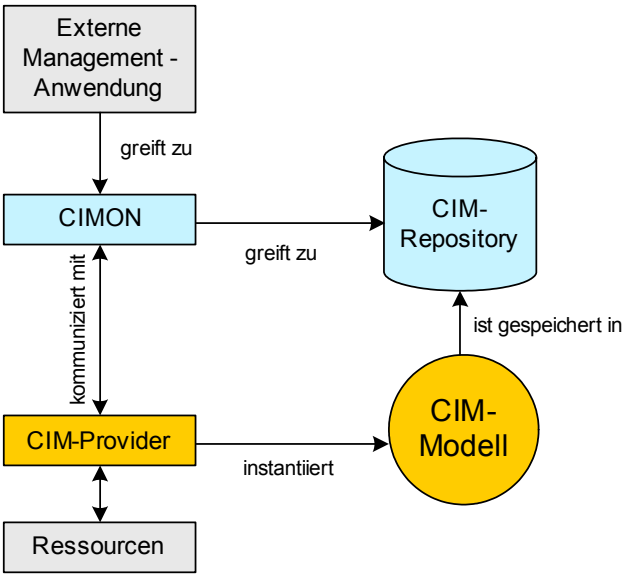


Abbildung 5.2: WBEM-Architektur zur Verwaltung von CIM-Ressourcen (nach [Jäh03])

WBEM bedient sich eines Client-Server-Konzepts und unterscheidet Client-seitig die Management-Applikation und Server-seitig die mittels CIM bzw. WBEM zu verwaltende(n) Ressource(n). Während für das Management (auf Client-Seite) beliebige Applikationen zum Einsatz kommen können, die gegebenenfalls spezielle Verwaltungsaufgaben erfüllen (bspw. die Überwachung von kritischen Systemzuständen oder die Konfiguration von Anwendungen), bedarf es Server-seitig einer verwobeneren Infrastruktur, damit ein System „WBEM-tauglich“ verwaltbar ist. Im Zentrum dieser Infrastruktur steht der CIM Object Manager (CIMOM), ein Softwareprogramm, das alle WBEM-Anfragen - deren Ziel das lokale System ist, auf dem er läuft - entgegennimmt und bearbeitet. Der CIMON fungiert als zentrales Schaltwerk, das Kenntnis darüber besitzt, welche Klassen existieren und mit welchen verwalteten Systemkomponenten sie korrelieren.

[27] http://www.dmtf.org/standards/wbem
[28] http://www.oasis-open.org/committees/tc_home.php?wg_abbrev=wsdm

Eine Übersicht über diese Gesamtarchitektur zeigt **Abbildung 5.2**. Der Fluss von Management-Informationen erfolgt dabei zwischen Manager und Ressource(n) auf der linken Seite von oben nach unten über den CIMON, den Implementierungs-abhängigen CIM-Provider und der entsprechenden Schnittstelle, bis hin zur Ressource. Dem korrespondiert auf der rechten Seite die Beschreibung dieser Informationen. Im Repository ist das CIM-Modell der lokal verwalteten Ressourcen hinterlegt.

Im Rahmen der Arbeit [Böt07] haben wir theoretisch und anhand einiger praktischer Beispiele untersucht, in welchem Umfang die Ziele des Autonomic Computing mit Hilfe von **Web Services** realisiert werden können und inwieweit der Einsatz von Web Services dem Autonomic Computing Nutzen verspricht. Schwerpunkt der Untersuchungen bildet das Web Services Distributed Management (**WSDM**). Zusammen mit anderen Web-Services-Spezifikationen sollen verschiedene proprietäre Management-Lösungen bestehender Systeme im Sinne einer Service-orientierten Architektur Plattform-unabhängig gemacht und in ihrer Nutzung homogenisiert werden. Ziel der WSDM-Spezifikationsfamilie ist es, Verwaltungsfunktionen mittels Web Services zu beschreiben sowie das Verwalten von Web Services zu ermöglichen. WSDM-Management-Anwendungen besitzen somit typische Eigenschaften von Web Services, wie z.B. Interoperabilität, lose Kopplung und Implementierungs-Unabhängigkeit.

Dafür definiert WSDM lediglich die Web-Service-Schnittstellen für die zu verwaltenden Ressourcen. Es spezifiziert nicht den Inhalt der zugreifbaren Management-Informationen und stellt somit nicht direkt ein Daten- bzw. Informationsmodell dar. Die zu verwaltenden Ressourcen können mit einem beliebigen Datenmodell, wie CIM, SNMP etc., beschrieben werden.

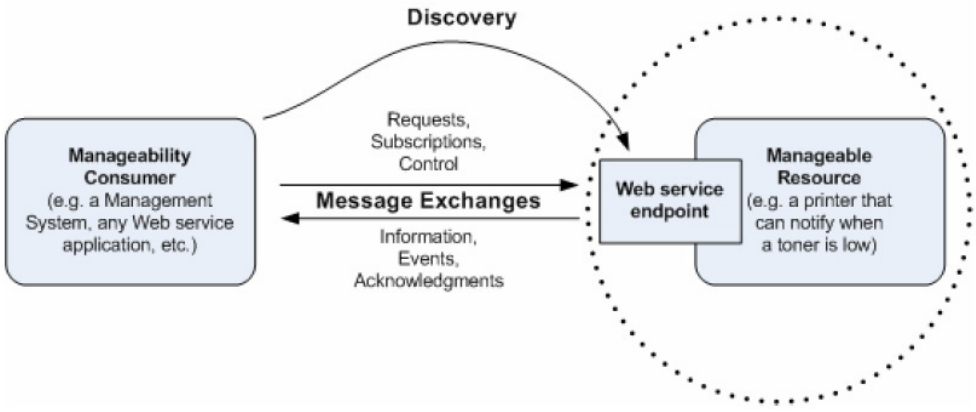


Abbildung 5.3: Schema der WSDM-Architektur [OAS06a]

Der WSDM-Standard besteht dabei i.W. aus zwei Teil-Spezifikationen, die die Grundlage dafür sind, dass die Nutzung des Web Services ermöglicht wird: *Management Using Web Services* (MUWS, [OAS06a]) und *Management Of Web Services* (MOWS, [OAS06b]). MUWS stellt dabei die Basis-Spezifikation dar. Sie beschreibt, wie und mit welchen Schnittstellen eine an ein Netzwerk angebundene IT-Ressource unter Zuhilfenahme von Web Services beschrieben, genutzt und verwaltet werden kann. MUWS beschreibt, wie Management-Schnittstellen von Ressourcen als Web Services bereitgestellt werden können. MOWS definiert, wie Web Services als Ressourcen verwaltet werden können und wie man mit Hilfe von MUWS darauf zugreifen kann.

Demzufolge sind alle verwaltbaren Ressourcen entweder Web Services oder repräsentiert über Web Services. Grundlage hierfür ist das Web Service Ressource Framework (WSRF,

[CFF+04]), welches ermöglicht, standardisiert auf Ressourcen zuzugreifen, diese zu beschreiben und die Verknüpfung bzw. Kommunikation dieser mit Web Services herzustellen. Daneben kommen verschiedene grundlegende Web-Services-Spezifikationen zum Einsatz, hauptsächlich für den Nachrichtenaustausch (WS-Messaging), für das Entdecken von zu verwaltenden Ressourcen (WS-Discovery), Benachrichtigungen (WS-Notifications) und für Sicherheitsaspekte (WS-Security).

Abbildung 5.3 skizziert das Basis-Schema der WSDM-Architektur. Ein *Web Service Endpoint* bietet nach [OAS06a] einen Zugang zu einer *Manageable Ressource*. Ein Beispiel einer Manageable Ressource wäre ein Drucker, welcher die Fähigkeit besitzt, einen Alarm auszulösen, sobald der Toner einen kritischen Füllstand erreicht. Denkbar ist auch eine Magnetspeicherplatte, welche in Form einer Web-Service-Operation über ihre interne Temperatur benachrichtigt. Ein *Manageability Consumer (MC)* entdeckt den Web-Service Endpoint und tauscht Nachrichten mit diesem Endpoint aus, um Informationen abzufragen, Events zu subskribieren oder die Manageable Ressource zu steuern, die mit diesem Endpoint assoziiert ist. Der MC erhält hierzu sowohl Antworten auf seine Anfragen, als auch Benachrichtigungen (wenn signifikante Ereignisse auftreten).

Im direkten Vergleich ist WBEM etwas starrer als das WSDM-Management-Framework konzipiert. Die Beschreibung von Ressourcen muss in WBEM ausschließlich mit CIM, kann allerdings in WSDM durch ein beliebiges Modell erfolgen. Auch wenn letzteres für eine hohe Flexibilität sorgt, so sehen wir doch Kompatibilitäts-Grenzen bei der Abbildung auf eine Web-Services-basierte Darstellung. Dies mag wohl auch einer der Gründe sein, dass derzeit noch keine vollautomatische Lösung für die Kombination beider Standards existiert.

Im Zuge unserer Betrachtungen in [Böt07] kommen wir zu dem Schluss, dass Web Services eine vielversprechende Grundlage bilden, auf der die Automatisierung der Verwaltung von IT-Systemen erreicht werden kann. Wie auch in den Kerngedanken des Autonomic Computing beschrieben, basieren Web Services auf offenen Standards. Mit der damit gewonnenen Plattform-Unabhängigkeit kann eine flexible Architektur geschaffen werden, die für eine Vielzahl unterschiedlichster IT-Systeme eingesetzt werden kann, ohne dass jeweils gravierende Änderungen vorgenommen werden müssen. Auf Grundlage dieser gewonnenen Unabhängigkeit ist es möglich, die Heterogenität von Systemkomponenten bewältigen zu können und eine einheitliche Beschreibung der einzelnen Ressourcen durchzusetzen.

Derzeit gibt es jedoch eine Menge an Spezifikationen, die Web-Service-Mechanismen für den Umgang mit Ressourcen, die Kommunikation mittels Events und das übergreifende Ressourcen-Management ermöglichen [Cli06, HuGa06]. HP, IBM, Intel und Microsoft haben bereits eine Vielzahl an Plattform-übergreifenden Implementierungen für diese Spezifikationen in Form von Produkten und Developer Kits angeboten. Erfahrungen der Kunden und Nutzer dieser Systeme lassen jedoch den Wunsch einer stetigen und evolutionären Vereinheitlichung der vielen, teils konkurrierenden Spezifikationen aufkommen.

Es wird sich daher in den nächsten Jahren erst noch zeigen müssen, ob WSDM bspw. den erhofften Erfolg haben wird. Tatsache ist allerdings, dass es einen Schritt in die richtige Richtung darstellt, um den Vorgang der Beschreibungen von Ressourcen zu vereinheitlichen. Die sehr abstrakten und eingeschränkten Möglichkeiten zur Beschreibung der einzelnen Ressourcen und deren Abhängigkeiten untereinander, der große Einarbeitungsaufwand sowie das derzeitige Fehlen von Entwicklungsumgebungen lassen uns jedoch von einer Integration des Ansatzes vorerst absehen.

5.2.4 Standardisierte Ressourcen- und Event-Modellierung am Beispiel von DB2

Um die aufgegriffenen Standards zur Ressourcen- und Event-Modellierung anschaulich demonstrieren zu können, wird in [Alg10] sowie in [Ell10] an praktischen Beispielen die Abbildung von Eigenschaften einiger Ressourcen in DB2 vorgestellt. Wir wollen die Ergebnisse an dieser Stelle in Kürze vorstellen.

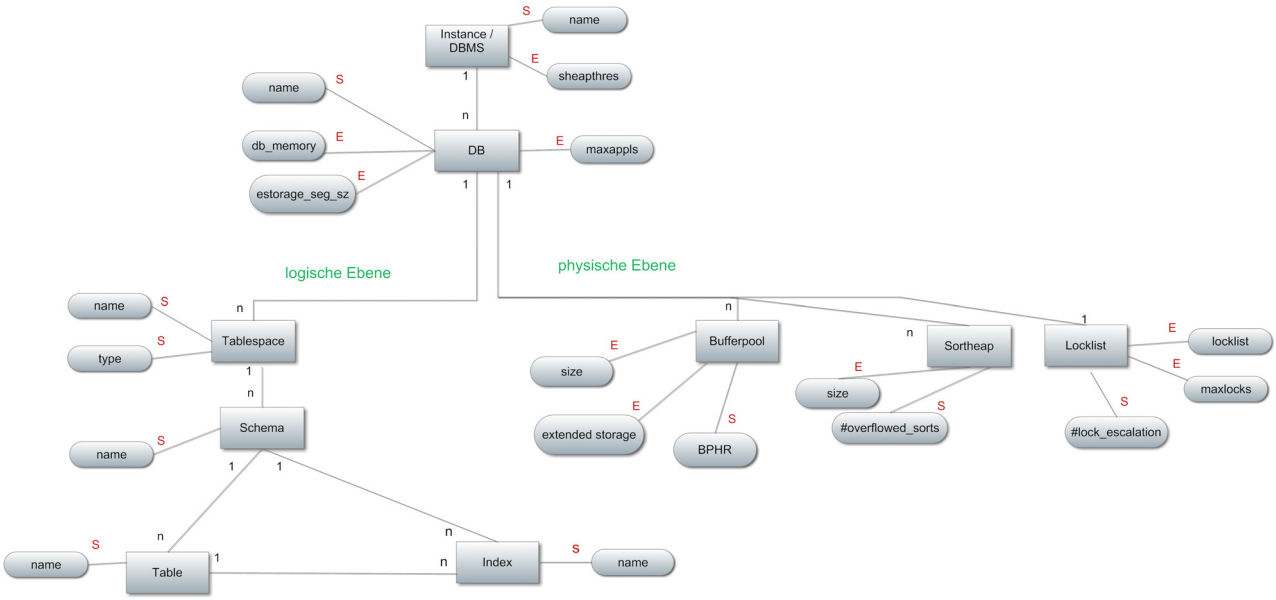


Abbildung 5.4: Konzeptionelles E/R-Abbild einiger DB2-Ressourcen [Alg10]

Abbildung 5.4 stellt einen Ausschnitt der DB2-Objekt-Hierarchie dar. Entity-Typen werden als Rechtecke, hierarchische 1:n-Beziehungs-Typen als ungerichtete Kanten mit den entsprechenden Kardinalitäten dargestellt. Beispielsweise liegt an oberster Stelle in der DB2-Hierarchie eine Instanz. Diese besitzt eine oder mehrere von ihr verwaltete Datenbanken. Eine DB2-Instanz kann mehrere Datenbanken enthalten, aber eine Datenbank darf nur zu genau einer Instanz gehören. Auf der logischen Ebene besteht eine Datenbank aus mehreren Tablespaces. Auf der physischen Ebene können einer Datenbank mehrere Bufferpools und Sortier-Bereiche sowie genau eine Sperren-Liste zugeordnet werden . Für nähere Details sei auf [Alg10] verwiesen.

Die Sensoren (S) und Effektoren (E) der DB2-Objekte sind in Form von Attributen skizziert. Jede DB2-Ressource hat ihre eigenen, charakteristischen Sensor-Metriken und Effektoren in Form von Konfigurationsparametern. Alle hierbei aufgeführten DB2-Komponenten haben als Sensor-Element zumindest einen Namen.

Basierend auf dem vereinfachten DB2-Abbild soll ein CIM-Modell zur Beschreibung der Struktur (und der Abhängigkeiten) sowie ein CBE-Modell zur Beschreibung der auszutauschenden Daten entwickelt werden.

5.2.4.1 Eine beispielhafte CIM-Modellierung

CIM zeigt sich nach ausführlicher Analyse in [Al10, Ell10] für den Einsatz im (autonomen) Datenbank-Tuning prädestiniert, weil hier Tausende von Konfigurationsparametern und sensorische Daten aller Art erfasst und miteinander korreliert werden müssen. Hierfür haben wir in [Alg10] die Problematik der Überwachung von Sensoren und automatischen Anpassung von DB2-Parametern beispielhaft mittels diverser CIM-Schemata modelliert.

Abbildung 5.5 zeigt die UML-Darstellung[29] des zugrunde gelegten CIM-Modells. Die Idee der „Individualisierung“ für unser Szenario liegt in der Möglichkeit der Vererbung. Von bereits im CIM Schema existierenden Klassen abgeleitete, eigene Klassen erben die Eigenschaften ihrer Vorfahren und erlauben spezifischere Darstellungen. Die Klassen, deren Name mit „DB2_ “ beginnt und die in hellblauer Farbe dargestellt sind, sind neu definierte Klassen des Extension Schemas. Alle anderen Klassen sind Klassen des CIM Core bzw. Common Schemas. Die Entwurfs-Entscheidungen sowie die einzelnen gewählten Klassen und Assoziationen werden in [Alg10] ausführlich erklärt. Auf einige Besonderheiten wollen wir jedoch hier in Kürze eingehen.

Um die physischen Speicherbereiche Shared Memory, Bufferpool, Locklist und Sortheap zu realisieren, wurden von der CIM-Klasse *PhysicalElement* des CIM Core Schemas die Extension-Klassen *DB2_SharedMemory, DB2_Bufferpool*, *DB2_Locklist* und *DB2_Sortheap* abgeleitet und über Assoziationen entsprechend verbunden.

Zur Modellierung von DB2-Datenbankmanager- bzw. DB2-Datenbank-Konfigurationsparameter sind die Extension-Klassen *DB2_DBMSParameter* bzw. *DB2_DBParameter* zuständig, die wiederum von der Extension-Klasse *DB2_ConfigurationParameter* abgeleitet wurden. Da die Größe des Bufferpools nicht über einen Konfigurationsparameter, sondern über eine SQL-Anweisung gesteuert wird, haben wir eine zusätzliche Extension-Klasse *DB2_BufferpoolParameter* von der CIM-Klasse *ScopedSettingData* abgeleitet.

Grundsätzlich trennen wir hierbei die statischen Metadaten der Metriken (bspw. Typ, Name oder auch die Formel bei zusammengesetzten Metriken) von den tatsächlichen, dynamischen Werten und Entstehungszeiten. Der Ursprung bzw. die Ressourcen-Zuordnung der Metriken wird über die Assoziationen zu den entsprechenden Managed Elements identifiziert. So wurde z.B. die Metrik des Sortheaps, sprich dem Speicherbereich für Sortiervorgänge, über die Assoziation *DB2_MetricValueForSortheap* realisiert.

Eine mögliche Instantiierung dieser Vorgabe auf Typebene findet sich exemplarisch in [Ell10]. In Abschnitt 6.5 stellen wir darüber hinaus ein davon abstrahiertes, *generisches CIM-Datenmodell* vor, das sämtliche Eigenschaften und Assoziationen von Ressourcen sowie deren Laufzeitdaten darstellen kann und als Basis für unser Performance-Daten-Repository dient.

[29] Für eine Einführung in die UML-Notation sei bspw. auf http://www.omg.org/technology/documents/-modeling_spec_catalog.htm#UML verwiesen.

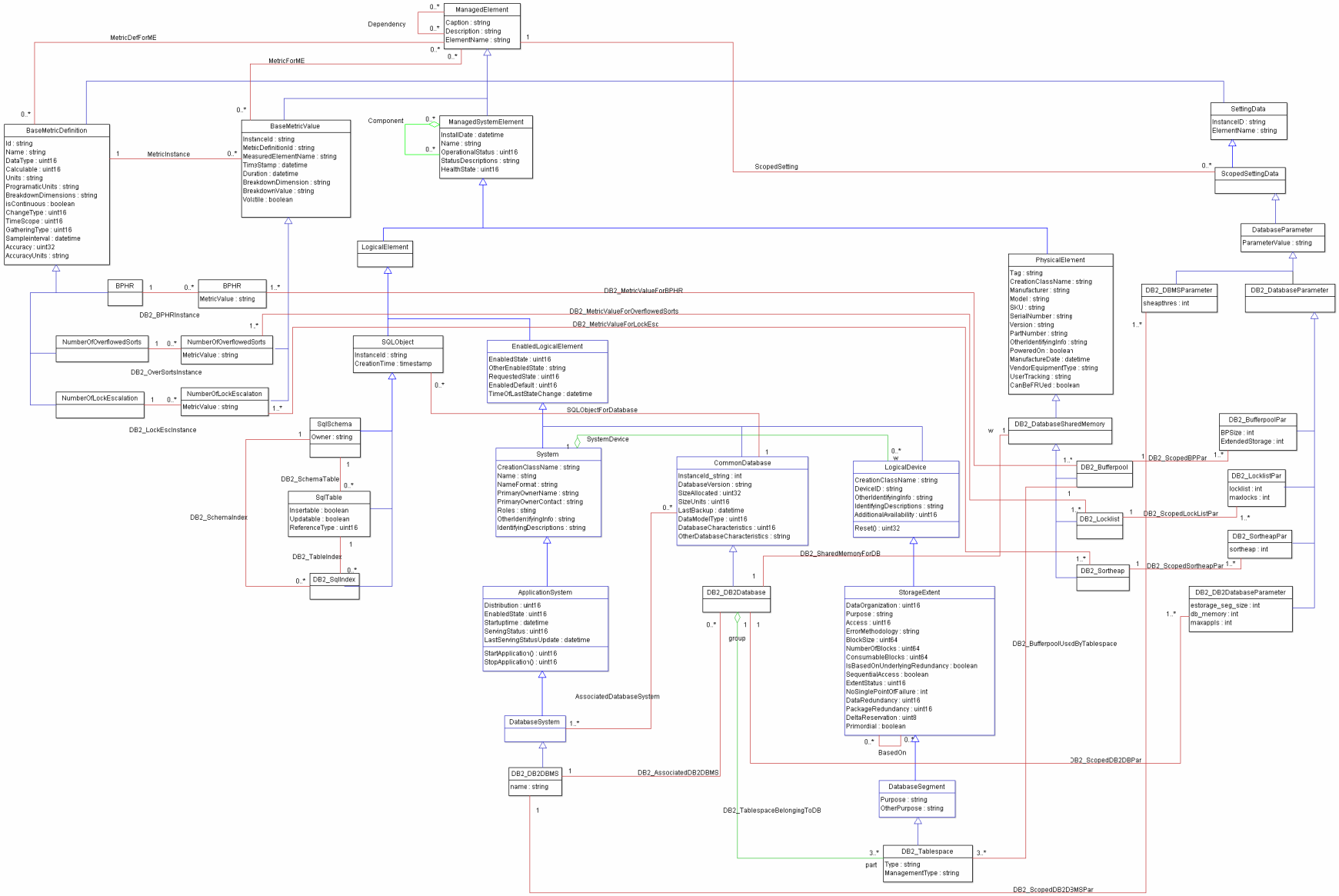

ManagedElement
Caption : string
Description : string
ElementName : string
Dependency
MetricDefForME
MetricForME
BaseMetricDefinition
Id : string
Name : string
DataType : uint16
Calculable : uint16
Units : string
ProgrammaticUnits : string
BreakdownDimensions : string
isContinuous : boolean
ChangeType : uint16
TimeScope : uint16
GatheringType : uint16
SampleInterval : datetime
Accuracy : uint32
AccuracyUnits : string
MetricInstance
BaseMetricValue
InstanceId : string
MeticDefinitionId : string
MeasuredElementName : string
TimeStamp : datetime
Duration : datetime
BreakdownDimension : string
BreakdownValue : string
Volatile : boolean
Component
ManagedSystemElement
InstallDate : datetime
Name : string
OperationalStatus : uint16
StatusDescriptions : string
HealthState : uint16
ScopedSetting
SettingData
InstanceID : string
ElementName : string
ScopedSettingData
DatabaseParameter
ParameterValue : string
LogicalElement
PhysicalElement
Tag : string
CreationClassName : string
Manufacturer : string
Model : string
SKU : string
SerialNumber : string
Version : string
PartNumber : string
OtherIdentifyingInfo : string
PoweredOn : boolean
ManufactureDate : datetime
VendorEquipmentType : string
UserTracking : string
CanBeFRUed : boolean
DB2_DBMSParameter
sheapthres : int
DB2_DatabaseParameter
BPHR
BPHR
MetricValue : string
DB2_BPHRInstance
DB2_MetricValueForBPHR
DB2_MetricValueForOverflowedSorts
DB2_MetricValueForLockEsc
NumberOfOverflowedSorts
NumberOfOverflowedSorts
MetricValue : string
DB2_OverSortsInstance
NumberOfLockEscalation
NumberOfLockEscalation
MetricValue : string
DB2_LockEscInstance
SQLObject
InstanceId : string
CreationTime : timestamp
EnabledLogicalElement
EnabledState : uint16
OtherEnabledState : string
RequestedState : uint16
EnabledDefault : uint16
TimeOfLastStateChange : datetime
SQLObjectForDatabase
SystemDevice
SqlSchema
Owner : string
DB2_SchemaTable
SqlTable
Insertable : boolean
Updatable : boolean
ReferenceType : uint16
DB2_SchemaIndex
DB2_TableIndex
DB2_SqlIndex
System
CreationClassName : string
Name : string
NameFormat : string
PrimaryOwnerName : string
PrimaryOwnerContact : string
Roles : string
OtherIdentifyingInfo : string
IdentifyingDescriptions : string
CommonDatabase
InstanceId_string : int
DatabaseVersion : string
SizeAllocated : uint32
SizeUnits : uint16
LastBackup : datetime
DataModelType : uint16
DatabaseCharacteristics : uint16
OtherDatabaseCharacteristics : string
LogicalDevice
CreationClassName : string
DeviceID : string
OtherIdentifyingInfo : string
IdentifyingDescriptions : string
AdditionalAvailability : uint16
Reset() : uint32
DB2_DatabaseSharedMemory
DB2_BufferpoolPar
BPSize : int
ExtendedStorage : int
DB2_Bufferpool
DB2_ScopedBPPar
DB2_LocklistPar
locklist : int
maxlocks : int
DB2_Locklist
DB2_ScopedLockListPar
DB2_SortheapPar
sortheap : int
DB2_Sortheap
DB2_ScopedSortheapPar
DB2_SharedMemoryForDB
DB2_DB2DatabaseParameter
estorage_seg_size : int
db_memory : int
maxappls : int
ApplicationSystem
Distribution : uint16
EnabledState : uint16
Startuptime : datetime
ServingStatus : uint16
LastServingStatusUpdate : datetime
StartApplication() : uint16
StopApplication() : uint16
DB2_DB2Database
group
StorageExtent
DataOrganization : uint16
Purpose : string
Access : uint16
ErrorMethodology : string
BlockSize : uint64
NumberOfBlocks : uint64
ConsumableBlocks : uint64
IsBasedOnUnderlyingRedundancy : boolean
SequentialAccess : boolean
ExtentStatus : uint16
NoSinglePointOfFailure : int
DataRedundancy : uint16
PackageRedundancy : uint16
DeltaReservation : uint8
Primordial : boolean
BasedOn
DB2_BufferpoolUsedByTablespace
AssociatedDatabaseSystem
DatabaseSystem
DB2_DB2DBMS
name : string
DB2_AssociatedDB2DBMS
DB2_ScopedDB2DBPar
DB2_TablespaceBelongingToDB
DatabaseSegment
Purpose : string
OtherPurpose : string
DB2_Tablespace
Type : string
ManagementType : string
part
DB2_ScopedDB2DBMSPar

Abbildung 5.5: Ein DB2-spezifisches CIM-Modell (in Anlehnung an [Alg10])

5.2.4.2 Ein beispielhaftes CBE-Event

Um das Konzept der CBEs anhand eines Beispiels zu verdeutlichen, wird als ein simples Szenario die autonome Korrektur der Bufferpool-Größe im Falle einer zu geringen Bufferpool Hit Ratio (BPHR, siehe Abschnitt 8.3) aufgegriffen.

Das im Folgenden veranschaulichte Beispiel-Event im CBE-Format wird automatisch durch die User-Exit-Routine des **DB2 Performance Expert** erzeugt und weitergeleitet, sobald die BPHR einen zuvor definierten Schwellwert unterschreitet. Es sollen nur die wichtigsten Ausschnitte aus dem Beispiel-CBE erklärt werden. Eine vollständige Auflistung und Beschreibung findet sich in [Alg10].

```
<extendedDataElements name="monitored_instance"
type="noValue">
     <children name="db2_version" type="string">
          <values>V9R5FP0</values>
     </children>
     <children name="host_name" type="string">
          <values>iibm04.inf-db.uni-jena.de</values>
     </children>
     <children name="instance_alias" type="string">
          <values>DB2INST</values>
     </children>
</extendedDataElements>
```

Listing 5.1: Exemplarisches CBE - Header-Informationen

Die Information der überwachten Komponente ist in **Listing 5.1** unter ExtendedDataElements **"monitored_instance"** beschrieben. Das Konstrukt ExtendedDataElement dient für beliebige Event-Daten, die in keine der anderen (vordefinierten) Klassen passen. Es handelt sich hier demnach bei der Event-verursachenden Ressource um eine DB2-Instanz (DB2INST), die auf dem Host „iibm04.inf-db.uni-jena.de“ läuft.

```
<extendedDataElements name="threshold_exception"
type="noValue">
     <children name="category" type="string">
          <values>statistics</values>
     </children>
     <children name="exception_field" type="string">
          <values>pool_hit_ratio</values>
     </children>
     <children name="subcategory" type="string">
          <values>buffer pools</values>
     </children>
</extendedDataElements>
```

Listing 5.2: Exemplarisches CBE - Schwellwert-Informationen

Die Information des ExtendedDataElements „**threshold_exception**“ in **Listing 5.2** verdeutlicht, dass es sich bei dem Event um eine Schwellwertverletzung (Unterschreitung) der BPHR handelt.

```
<extendedDataElements name="datagroup" type="string">
<values>pe_excplog</values>
     <children name="pel_startts" type="string">
          <values>2010-09-18 15:27:44.521924</values>
     </children>
     <children name="pel_warningvalue" type="string">
          <values>90</values>
     </children>
     <children name="pel_currentvalue" type="string">
          <values>80</values>
     </children>
     <children name="bp_name" type="string">
          <values>IBMDEFAULTBP</values>
     </children>
     <children name="db_name" type="string">
          <values>DBUC</values>
     </children
</extendedDataElements>
```

Listing 5.3: Exemplarisches CBE - Event-Informationen

Im ExtendedDataElements „**datagroup**“ werden schließlich sowohl die genauen Daten der aufgetretenen Event-Instanz als auch die Definition des Events erfasst. Demnach ist die BPHR des Bufferpools IBMDEFAULTBP der Datenbank DBUC in dem **Listing 5.3** am 18.09.2010 um 15:27 Uhr auf 80% und damit unter den kritischen Schwellwert (warningvalue) von 90% gesunken.

Wie aus diesem Beispiel ersichtlich ist, wird das aufgetretene Ereignis sehr detailliert in einem klar strukturierten Format beschrieben. Es bietet somit dem Autonomic Tuner oder anderen Event Engines eine gute Voraussetzung für die weitere Verarbeitung und Korrelationen mit anderen Events. In Abschnitt 8.3 zeigen wir, wie eine auf dem CBE-Format basierende Event-Erkennung im Umfeld des autonomen Datenbank-Tuning realisiert werden kann.

5.2.5 Ontologien

Interessant scheint neben dem Einsatz von Standards auch die Rolle von Ontologien im Autonomic Computing [Sto04]. In [GrLe02] werden drei Anwendungsfelder von Ontologien unterschieden: Zusätzlich zur Unterstützung der *(1) Kommunikation* unter Personen, Organisationen, Prozessen und Systemen durch Definition eines gemeinsamen Vokabulars scheinen Techniken zur Formulierung komplexer logischer Abhängigkeiten bzw. semantischer Relationen und das Ziehen von *(2) Schlussfolgerungen* im Hinblick auf die konzeptionelle System-Modell- und Ressourcen-Beschreibung von Vorteil. Neben der Repräsentation, hat vor allem die *(3) Wiederverwendung* dieses Wissens eine enorme Bedeutung.

Typischerweise müssen zwei miteinander kommunizierende Programme (z.B. Software-Agenten) selbst die Vorschrift zur Interpretation ihrer ausgetauschten Daten in sich tragen. Alternativ können sie diese aber auch in Form von Metadaten aus einer, beiden Seiten zugänglichen, Ontologie mitliefern. Mithilfe der dann schon aufgrund der per Ontologie

bekannten Ableitungsregeln können die Programme dann bereits automatisch logische Schlüsse ziehen. Ähnlich ist es bei der Wissensrepräsentation und -wiederverwendung.

Ontologien bieten den Vorteil, dass Wissen auf eine Art und Weise beschrieben werden kann, wie es sowohl von Menschen als auch intelligenten Agenten genutzt werden kann. Zur Definition von Ontologien stehen nach [Neu06] u.a. folgende Möglichkeiten zur Verfügung (in absteigender Reihenfolge bzgl. ihrer Mächtigkeit):

- die Web Ontology Language (OWL)
- ein UML Modell
- ein Datenbank-Schema
- XML Schemas, welche zueinander in Verbindung stehen

Laut [Neu06] kann das Common Information Model im Speziellen als eine standardisierte Ontologie aufgefasst werden, welche die zu verwaltende Umgebung als eine Sammlung von verbundenen Systemen strukturiert. Aus unserer Sicht sind jedoch AI-basierte Sprachen (z.B. Ontolingua, LOOM, OCML, FLogic,…) und Ontologie Markup-Sprachen (z.B. RDFS, DAML+OIL, OWL, …) besser zur Repräsentation von Ontologien geeignet.

Der Fokus bei den etwaigen Betrachtungen sollte hierbei auf der geschickten Symbiose von Standard-Formaten und Ontologien liegen, mit dem Zweck der konzeptionellen Beschreibung des System- bzw. Ressourcen-Modells sowie der Beschreibung der Art der Daten und der Daten selbst, die von den Ressourcen von unterschiedlichen Tools erzeugt und zur späteren Problem-Analyse verwendet werden können. Die Vision ist dabei von unserem Standpunkt, das System-Modell implementierungsunabhängig mittels einer Ontologie darzustellen, um aus dieser doch eher konzeptionellen Sicht eine „physische(re)“ Repräsentation durch Verwendung von Standards und Formaten abzuleiten.

5.2.6 Fazit

Viele der Ansätze zur Beschreibung von Ressourcen sind nicht als Modelle zu verstehen, enthalten aber ein implizites Modell, das bspw. beschreibt, welche Entitäten existieren und welche Attribute sie besitzen.

Wünschenswert in Sachen Interoperabilität ist die Verwendung eines einzelnen, ganzheitlichen Ressourcen-Modells. In der Realität jedoch muss man davon ausgehen, gerade in komplexen IT-Umgebungen, verschiedenen, gleichzeitig aktiven und untereinander inkompatiblen Ressourcen-Modellen zu begegnen. Dies macht Techniken zur Koordination und zum Anpassen/Angleichen von Semantiken zwischen den Modellen erforderlich, beispielsweise mittels Ontologien.

Daher sollte man bei der nicht immer vermeidbaren Erzeugung von neuen bzw. angepassten Ressourcen-Modellen auf die Wiederverwendung bzw. Spezialisierung vorhandener Modelle achten. Dies erhöht die Kompatibilität und verringert den (zeitlichen) Aufwand zur Erstellung. Beispielsweise kann ein neues Ressourcen-Modell als Teilmenge bzw. Obermenge eines anderen erzeugt werden. Es ist auch denkbar, dass verschiedene Ausprägungen auf demselben (Teil-)Modell basieren, aber ihre eigene Syntax verwenden (z.B. eigenes XML-Schema). Während ein CIM-Modell bspw. die Semantik der Ressourcen beschreibt, stellen die XML-Repräsentation und das HTTP-Mapping verschiedene syntaktische Ausprägungen dar. Die Abbildung zwischen derartig verschiedenen (syntaktischen) Ausprägungen eines Ressourcen-Modells, die auf der gleichen Semantik beruhen, kann im Gegensatz zur Angleichung verschiedener Semantiken als relativ unkritisch angesehen werden. Bei äquivalenter Semantik ist in einigen Fällen sogar eine automatische Translation mög-

lich. Als reale Anwendungs-Beispiele seien zum einen das Mapping zwischen Globus[30]- und UNICORE[31]-Ressourcen im Rahmen des GRIP-Projekts [BFGG04] und zum anderen CIM-Mechanismen zur Abbildung der eigenen Semantik auf andere Ressourcen-Modelle [DMTF99] genannt.

Die untersuchten Standards wurden alle für unterschiedliche Anwendungsgebiete entwickelt und optimiert. Insofern ist verständlich, dass sich keiner der vorgestellten Ansätze als alleinige („optimale“) Lösung für den Einsatz im autonomen Datenbank-Performance-Tuning herausgestellt hat. Aus unserer Sicht erscheint jedoch die Kombination aus der Ressourcen- und Metrikdatenmodellierung in CIM und der Event-Darstellung bzw. -Verarbeitung mit Hilfe des CBE-Formats am geeignetsten.

5.3 Performance-Daten-Quellen

Typischerweise kann der Ist-Zustand eines IT-Systems mit seinen Ressourcen zur Laufzeit durch eine Menge von Performance-Daten (Metriken und Events) charakterisiert werden. Diese Performance-Daten können potentiell auf allen Ebenen des Software-Stacks (siehe Abschnitt 2.1) entstehen und, entsprechende Schnittstellen oder Tools vorausgesetzt, auch ausgelesen und weiterverwendet werden.

Eine adäquate Teilmenge der Performance-Daten über die von der Workload der End-Nutzer „betroffenen“ Ressourcen auf der operativen Ebene soll dauerhaft bzw. langfristig überwacht, analysiert und gespeichert werden. Das bereits vielfach angesprochene Ziel ist es, dadurch Performance-Probleme zu erkennen, einen störungsfreien Betrieb sowie eine effiziente Ressourcen-Nutzung zu gewährleisten und in Zukunft Problem-Situationen effektiver zu erkennen oder gar im Voraus zu vermeiden.

Aus mehreren zur Verfügung stehenden, in Bezug auf Struktur, Inhalt und Schnittstellen meist heterogenen Datenquellen müssen die für die späteren Analysen geeigneten zunächst ausfindig gemacht werden. Am Beispiel von IBM DB2 gibt es verschiedene DBMS-interne und externe Tools, die sich der Aufgabe der Überwachung annehmen und damit als Quelle für Performance-Daten eine wichtige Grundlage unserer zu entwickelnden Performance-Monitoring- und -Tuning-Architektur bilden können.

5.3.1 DBMS-interne Monitoring- (und Analyse-)Tools

Die Überwachungs-Werkzeuge in IBM DB2 lassen sich in zwei Kategorien einteilen [Ree04, Ree05]. Als **Performance-Monitoring-Tools** bezeichnen wir die Werkzeuge, welche lediglich Daten sammeln und Informationen liefern, die mehr oder weniger strukturiert gespeichert werden können. Zu dieser Kategorie kann man den in DB2 integrierten Database System Monitor mit Snapshot-, Event- und Health-Monitor sowie Explain Snapshots, Protokolldateien und das Tracing zählen. **Performance-Diagnose-Tools** dagegen stellen neben den reinen Sammlungs- und Speicherungs- auch Analyse-Möglichkeiten und oftmals reaktive, automatische Problem-Erkennungs- und -Auflösungs-Mechanismen zur Verfügung. Hierzu zählen im DB2-Umfeld bspw. der Activity Monitor, der Memory Visualizer, das Health Center, der Event Analyzer, die Explain Tools und das Problem Determination Tool.

Daneben existieren noch eine Menge proprietäre Tools von IBM oder Drittherstellern, die das Monitoring und damit die Diagnose der verschiedenen Schichten der operativen Ebene

[30] http://www.globus.org/
[31] http://www.unicore.eu/

unterstützen. Wir wollen uns im Folgenden lediglich einige wenige herausgreifen und kurz vorstellen. Für noch umfangreichere Details sei auf die Arbeiten [Ehl07, Hen07, Wei10] bzw. die Original-Dokumentationen verwiesen.

5.3.1.1 System Monitor

DB2 verfügt über ein mächtiges, integriertes Werkzeug für System- und Datenbank-Administratoren, den sogenannten System Monitor [IBM06i]. Der System Monitor sammelt Zustands- und Performance-Informationen über den Datenbankmanager, dessen Datenbanken und die verbundenen Anwendungen. Um auf diese Informationen zugreifen zu können, gibt es den **Snapshot Monitor**, den **Event Monitor** und den **Health Monitor**. Der *Snapshot Monitor* (Abschnitt 5.3.1.2) macht eine Momentanaufnahme des Systemzustands bzw. der -aktivitäten zu einem bestimmten Zeitpunkt, während der *Event Monitor* (Abschnitt 5.3.1.3) bei Auftreten bestimmter Ereignisse fortlaufend Laufzeitdaten protokolliert. Der *Health Monitor* (Abschnitt 5.3.1.4) sammelt Informationen, wenn „Fehler" oder Auffälligkeiten in Form von Überschreitungen definierter Schwellwerte in den überwachten Datenbanken auftreten.

Alle Arten von Monitoren können über Befehle, Anwendungsprogramm-Schnittstellen oder über graphische Benutzer-Oberflächen der DB2 Steuerzentrale (DB2 Control Center) gesteuert und kontrolliert werden. Die Monitor-Informationen können dabei in Dateien bzw. in Tabellen gespeichert, direkt auf dem Bildschirm angezeigt oder an Anwendungen zur weiteren Verarbeitung gesendet werden.

In der Regel lassen sich die DB2-Monitor-Informationen, in Form von Metriken oder Events, dabei einer der folgenden **Typen** zugeordnet [IBM06i]:

- *Counter*: Eine Metrik vom Typ Counter gibt Häufigkeiten einer bestimmten überwachten Aktivität einer Ressource zurück und kann ausschließlich steigen. Als Beispiel sei die Anzahl der gelesenen Tabellen-Zeilen oder die Zahl gehaltener Sperren zu nennen.
- *Gauge*: Dieser Typ wird genutzt, um den aktuellen Status einer (Ressourcen-)Aktivität anzuzeigen. Dieses Element kann im Gegensatz zum Counter steigen und sinken. Ein Beispiel für einen Gauge-Wert wäre die Bufferpool Hit Ratio. In jenem Fall bestimmt sich der Gauge-Wert aus einer Berechnung mehrerer Counter-Metriken.
- *Watermark*: Eine Watermark-Metrik dient dazu, den höchsten (Maximum-) oder niedrigsten (Minimum-) Wert zu bestimmen, den ein Element seit dem Starten der Überwachung erreicht hat. Exemplarisch sei hier die bisher maximale Anzahl an zu einer Datenbank verbundenen Applikationen aufgeführt.
- *Information*: Metrik-Werte dieses Typs geben Auskunft über qualitative Informationen eines überwachten Objekts und sind in der Regel im Vergleich zu allen anderen eher statischer Natur. Als Beispiel seien hier der Name der Datenbank und der Pfad der Log-Dateien genannt.
- *Timestamp*: Mittels Timestamp kann der Zeitpunkt des Auftretens einer Aktivität bzw. eines Statuswechsels angezeigt werden. Dies kann wichtig sein, um bestimmte Aktivitäten, etwa seit Beginn des Verbindungs-Aufbau einer Applikation mit der Datenbank, nachverfolgen zu können.
- *Time*: Hiermit kann die Zeit(dauer) angegeben werden, die eine bestimmte Aktivität braucht. Beispielsweise ist die Zeitspanne, die eine Anwendung auf die Freigabe einer Sperre wartet (lock_wait_time) eine Metrik vom Typ *Time*.

Um den Overhead beim Sammeln von Daten zu reduzieren und den Detail-Grad zu kontrollieren, gibt es sog. **Monitor Switches.** Die Monitor Switches können auf Instanz-Ebene (per UPDATE DBM CFG) oder Applikations-Ebene (per UPDATE MONITOR SWITCHES) an- und ausgeschaltet werden und kontrollieren den Umfang sowie Art der gesammelten Monitor-Elemente. Jede Anwendung hat somit ihre eigene Sicht auf die Monitor-Elemente und kann selbst bestimmen, was sie überwachen möchte. Dazu zählen u.a. Dauer und Anzahl von Lese- und Schreib-Operationen auf den Bufferpools, Anzahl an Deadlocks und Länge der Wartezeit auf Sperren sowie Informationen über SQL-Statements. Die Switches haben jedoch keinen Einfluss auf die von dem Event Monitor aufgenommenen Daten.

5.3.1.2 Snapshot Monitor

Der Snapshot Monitor kann verwendet werden, um Daten über den Datenbankmanager, die Datenbank oder bestimmte Anwendungen zu einem **bestimmten Zeitpunkt** zu sammeln [IBM06i]. Das Ziel dieser Momentaufnahmen-Überwachung mittels Snapshot Monitor ist die Bereitstellung von Informationen über den Zustand der Datenbank-Ressourcen und der eigentlichen (Nutz-)Daten sowie ein Hinweisen auf abnormale Situationen. Werden diese Schnappschüsse in regelmäßigen Abständen ermittelt, bietet sich eine gute Möglichkeit, um Trends und mögliche Probleme vorherzusagen.

Im Gegensatz zu potentiell vielen individuellen Ereignis-Monitoren gibt es nur einen Snapshot Monitor, der auf einfache Weise eingeschaltet werden kann, um verschiedene messbare Quantitäten in der Instanz zu protokollieren. Neben den Monitor Switches gibt es acht spezifische Snapshot Monitor Level um Monitor-Informationen auf unterschiedlichen Ebenen zu erhalten und den Detailgrad der Sammlung zu beeinflussen (siehe **Tabelle 5.1**). Wenn der einem Monitoring Level übergeordnete Switch jedoch nicht aktiviert ist, werden auch keine Daten zurückgeliefert. Ausführliche Informationen diesbezüglich finden sich unter [IBM06i, Wie05]. Auch in [Eat04] wird eine kurze Einführung in den Snapshot Monitor gegeben. Beschrieben werden u. a. die Auswirkungen der Monitor Switches auf die Performance der überwachten Datenbank. Diese bewegen sich je nach konkretem Switch und Workload zwischen 4% und 10%.

Monitor Level	Bereitgestellte Informationen
Database Manager	Sammelt Informationen über eine aktive Instanz
Database	Sammelt Datenbank-Informationen
Application	Sammelt Applikations-Informationen
Bufferpool	Sammelt Bufferpool-Aktivitäts-Informationen
Tablespace	Sammelt Informationen für die Tablespaces in der Datenbank
Table	Sammelt Informationen über die Tabellen in der Datenbank
Lock	Sammelt Informationen über die Sperren, die von den Applikationen gehalten werden
Dynamic SQL Cache	Sammelt Statement-Informationen aus dem SQL Statement Cache der Datenbank

Tabelle 5.1: Monitor Level des DB2 Snapshot Monitor

Um einen Snapshot zu erstellen, kann im Command Line Processor (CLP) ein entsprechender Befehl abgesetzt werden. **Listing 5.4** zeigt einen Auszug eines derartigen Datenbank-Schnappschusses („GET SNAPSHOT FOR DB ON SAMPLE“) auf einer Beispiel-Datenbank. Anhand der gelisteten Monitor-Elemente ist ersichtlich, dass zum Zeitpunkt der Aufnahme zwei Applikationen mit der Datenbank SAMPLE verbunden sind und

sieben Sperren halten. Daneben werden auch Informationen über die Bufferpool-Aktivitäten und die Zahl der SQL-Statements sowie Transaktionsabschlüsse (commits) bzw. -abbrüche (rollbacks) dargestellt.

```
      Database Snapshot

      Database name                               = SAMPLE
      Database status                             = Active

      Snapshot timestamp                          = 02/17/2010 12:19:11

      High water mark for connections             = 2
      Application connects                        = 2
      Applications connected currently            = 2
      Appls. executing in db manager currently    = 1
      Agents associated with applications         = 2
      Maximum agents associated with applications= 2
      Maximum coordinating agents                 = 2

      Locks held currently                        = 7
      Lock waits                                  = 1
      Time database waited on locks (ms)          = 26039
      Lock list memory in use (Bytes)             = 2304
      Deadlocks detected                          = 0
      Lock escalations                            = 0

      Buffer pool data logical reads              = 98
      Buffer pool data physical reads             = 27
      Buffer pool temporary data logical reads    = 0
      Buffer pool temporary data physical reads   = 0
      Asynchronous pool data page reads           = 0
      Buffer pool data writes                     = 2
      Asynchronous pool data page writes          = 0

      Commit statements attempted                 = 2
      Rollback statements attempted               = 0
      Dynamic statements attempted                = 8
      Static statements attempted                 = 2
      Failed statement operations                 = 0
      Select SQL statements executed              = 1
      Update/Insert/Delete statements executed    = 1
      DDL statements executed                     = 2

      ...
```

Listing 5.4: Auszug eines GET SNAPSHOT FOR DB ON SAMPLE

Alternativ zu dieser recht unflexiblen Bildschirmausgabe, können *Tabellenfunktionen*, *Views* oder Monitor-*API-Aufrufe* zur Gewinnung von Snapshot-Daten verwendet werden [IBM06i]. Die Monitor-Informationen, die mittels **Tabellenfunktionen** zurückgeliefert werden, sind besonders komfortabel. Jede Spalte repräsentiert ein Monitor-Element, während jede Zeile einem überwachten Objekt entspricht. Die Anweisung „SELECT * FROM TABLE(SNAPSHOT_DBM) AS T“ liefert bspw. eine temporäre Tabelle mit sämtlichen Informationen über den Datenbankmanager. Eine detaillierte Liste der verfügbaren Tabellenfunktionen ist in [IBM06i] sowie in [IBM10] enthalten.

Für die Monitor Level Database *Manager, Database, Application, Table, Lock, Tablespace, Bufferpool* und *Dynamic SQL* sind seit DB2 UDB v9 verschiedene **administrative Views**, die dieselben Informationen wie die entsprechenden Tabellenfunktionen liefern, verfügbar. Um beispielsweise die Anwendungs-Informationen zu einer Datenbank abzufragen, genügt die Anweisung „SELECT * FROM SYSIBMADM.SNAPAPPL“. Seit DB2 UDB v9.1 gibt es darüber hinaus zusätzlichen Views, die neben den thematisch gruppierten Werten einzelner Monitor-Elemente auch errechnete Werte (z.B. Bufferpool Hit Ratio) zurückliefern und damit mehr Potential für Analysen bieten. Dazu zählen neben der Bufferpool Hit

Ratio bspw. auch Informationen zu den am längsten aktiven SQL-Anweisungen, die zudem nach Anzahl der Ausführungen, durchschnittlicher Ausführungszeit oder Anzahl der Sortierungen geordnet dargestellt werden können.

Wie nützlich Snapshots für die Analyse sind, hängt in besonderem Maße von den Intervallen ab, in denen sie erzeugt werden. Sammelt man zu selten, können wichtigen Informationen verpasst und Problemsituationen mitunter nicht erkannt werden. Eine hoch-frequente Sammlung hingegen, bringt nicht immer einen Mehrwert und kann das operative System unnötig belasten. Im Gegensatz zu Events und deren Monitoring haben sie jedoch in der Regel den Vorteil, dass sie wenig Speicherplatz und Performance kosten.

5.3.1.3 Event Monitor (Ereignis-Monitor)

Ereignis-Monitore dienen der umfangreichen, **lückenlosen Protokollierung** von Informationen über eine Datenbank und daran angeschlossene Anwendungen, wenn bestimmte Ereignisse aufgetreten sind [IBM06i]. Dies erlaubt die Sammlung von Informationen über Zustand-Änderungen, die über eine Momentaufnahmen-Überwachung durch Snapshots nur sehr schwierig oder gar nicht erfasst werden können. Ein Ereignis-Monitor wird in einer bestimmten Datenbank erzeugt und überwacht dann nur Ereignisse in dieser Datenbank. Der Snapshot Monitor hingegen kann System-Aktivitäten über Datenbank-Grenzen hinweg überwachen.

Ereignisse repräsentieren dabei Datenbank-Aktivitäten, wie beispielsweise Verbindungs-Auf- und -Abbau, Deadlocks, Transaktions-Beginn und -Ende etc. So wartet zum Beispiel ein Deadlock Event Monitor solange, bis ein Deadlock auftritt und sammelt anschließend Informationen über die beteiligten Anwendungen und Sperren, die zu dieser Situation geführt haben. Mit einem Snapshot Monitor kann die Wahrscheinlichkeit hoch sein, dass der Deadlock übersehen wird oder schon aufgelöst sein könnte, bevor überhaupt ein Abbild des Systems gezogen wird.

Anstatt einen „Schnappschuss“ des Zustands des DBMS zu einem bestimmten Zeitpunkt zu liefern, beschafft ein Ereignis-Monitor eine kontinuierliche Folge von Berichten über Ereignisse so, wie diese auftreten. Jeder Ereignis-Monitor zielt darauf ab, einen bestimmten Ereignis-Typ zu überwachen, der für die Instanz, die den Monitor erzeugt hat, von Interesse ist. Die ermittelbaren Informationen der verschiedenen Ereignis-Typen entsprechen im Grunde denen des Snaphot Monitor. So können auch mittels eines Event Monitor umfangreiche Informationen zu den beteiligten Anwendungen, deren Anweisungen sowie zu Performance-Metriken auf Datenbank-, Bufferpool- oder auch Tablespace-Ebene gewonnen werden [IBM06i, Wie05].

Im Unterschied zum Snapshot Monitor muss ein Event Monitor, nachdem er (per „CREATE EVENT MONITOR“) erzeugt wurde, aktiviert werden (per SET EVENT MONITOR STATE) um kontinuierlich Performance-Daten im Falle des Eintretens der durch ihn beobachteten Events zu sammeln. Benutzer können zur Überwachung unterschiedlicher Typen von Ereignissen eine Reihe von Monitoren erzeugen und jeden davon unabhängig von dem anderen aktivieren oder deaktivieren.

Der Lebenszyklus eines Event Monitors erstreckt sich demnach von seiner Erzeugung bis zu seiner Löschung. Bei der Erzeugung wird entweder eine Datei, Tabelle oder eine Pipe zur Ausgabe der Daten angegeben. Danach erfolgt die Aktivierung des Event Monitors und die aktive Phase, in der Daten gesammelt werden. Nach einer Deaktivierung muss zur endgültigen Beseitigung noch die Löschung des Event Monitors mittels DROP EVENT

MONITOR erfolgen. Die erzeugten Tabellen oder Dateien sind von der Löschung des Event Monitors nicht betroffen und können im nachhinein ausgewertet werden.

Der **Event Analyzer** (db2eva) dient bspw. der Auswertung der Informationen, die der Event Monitor in *Tabellen* abgelegt hat. Damit können jedoch lediglich Informationen von nicht mehr aktiven Ereignis-Monitoren angezeigt werden. Um die Daten eines Event Monitors korrekt im Event Analyzer anzuzeigen, sollte der Event Monitor vorher deaktiviert werden. Für die Analyse von Informationen, die der Event Monitor in *Dateien oder eine Pipe* geschrieben hat, eignet sich das **Event Monitor Productivity Tool** (db2evmon). Die Informationen werden in formatierter Weise auf der Standardausgabe ausgegeben. Das Tool muss hierbei jedoch vor der Aktivierung des entsprechenden Event Monitor gestartet werden [IBM06m].

Bei dem Event-Monitoring besteht der Vorteil darin, dass alle Daten in dem entsprechenden „Intervall", welches durch den Benutzer durch Angabe des zu überwachenden Events festgelegt wurde, durchgehend aufgezeichnet werden und somit keine evtl. benötigten Daten fehlen. Der Overhead des Event Monitors ist jedoch aufgrund der kontinuierlichen Sammlung wesentlich größer als der des Snapshot Monitors. Als Analogie kann man sich einen Snapshot als Foto vorstellen, während der Event Monitor einen Film aufzeichnet.

Eine übersichtliche und mit anschaulichen Beispielen unterlegte Einführung in den Event Monitor ist [Ree05] zu entnehmen. Der Autor schildert darin auch intelligente Maßnahmen und Tricks, wie der durch den Event Monitor verursachte Overhead eingeschränkt werden kann. Es wird u.a. geraten, Dateien den Tabellen als Speichermedium vorzuziehen bzw. alternativ Tabellen in einem separaten Tablespace abzulegen, der mit einem ausreichend dimensionierten Bufferpool versehen ist. Es ist ebenso empfehlenswert, Gebrauch von der Filterung zu machen, um die zu sammelnden Ereignis-Daten auf die wesentlichen Informationen (z.B. bestimmte Anweisungstypen, Datenbanken, Anwendungen etc.) einzuschränken. Mit Hilfe dieser Maßnahmen kann man den vom Event Monitor verursachten Overhead auf 10% bis 20% einschränken.

5.3.1.4 Health Monitor und Health Center

Bei dem **Health Monitor** handelt es sich um DB2-internes Tool, das den Status einer Instanz und aktiver Datenbanken mittels Indikatoren fortlaufend überwacht und bei Auftreten Schwellwert-verletzender Ausnahme-Situationen Warnungen bzw. Alarme auslöst und/oder vordefinierte Aktionen einleitet. Die Warnmeldungen dienen dazu, Administratoren über den vermeintlich kritischen Systemzustand zu benachrichtigen und die Informationen in ein Protokoll zu verzeichnen. Die Aktionen können dazu beitragen, den („Gesundheits"-)Zustand des Systems zu verbessern.

Der Monitor überprüft den Zustand des Systems mit Hilfe sogenannter *Health Indicators*, um zu ermitteln, ob eine Warnung oder ein Alarm ausgegeben werden muss. Health Indicators gibt es auf Instanz-, Datenbank-, Tablespace- und Tablespace-Container-Ebene. Man unterscheidet nach [IBM06i] drei Arten:

- *Threshold-based:* Diese Indikatoren haben einen kontinuierlichen Wertebereich und besitzen zwei Schwellwerte (Warnung und Alarm), die diesen Wertebereich in drei Bereiche teilen. Je nachdem, in welchem dieser Bereiche sich der Indikator(wert) befindet, hat er den Zustand „Normal", „Warning" oder „Alarm".
- *State-based*: Diese Indikatoren repräsentieren eine endliche Menge von zwei oder mehr Zuständen einer Ressource und zeigen an, ob sie normal arbeitet oder nicht. Ein Zustand wird als „Normal", alle anderen werden als nicht normal („Attention") angenommen.

- *Collection state-based:* Diese (Gruppen-)Indikatoren überwachen auf Datenbank-Ebene eine Menge von Ressourcen, aus deren Einzel-Zuständen sich ein aggregierter Gesamt-Zustand ergibt. Befindet sich eine der Ressourcen im abnormalen Zustand „Attention", so gilt dies auch für den (Gruppen-)Indikator. Andernfalls nimmt der Indikator den Zustand „Normal" an.

Eine Warnung bzw. ein Alarm und/oder eine Aktion werden immer dann ausgelöst, wenn sich ein Indikator von einem normalen („Normal") in einen nicht normalen Zustand („Attention", „Warning", „Alarm") begibt [IBM06i]. Für jeden Indikator speichert der Health Monitor die letzten zehn Werte. Durch einen Health Snapshot kann auf die Werte der Indikatoren zugegriffen werden. Der Health Monitor ist dabei für eine Datenbank solange aktiv, wie die Datenbank aktiv ist.

Das **Health Center** ist ein graphisches Frontend zum Health Monitor. Sämtliche Einstellungen der Schwellwerte, abgesetzte Alarme und ausgeführte Aktionen können hier konfiguriert und verfolgt werden. Das Health Center wird darüber hinaus durch das **Web Health Center** um die Darstellung der Informationen über einen Web-Browser erweitert [IBM06i].

5.3.1.5 Explain Snapshots & Explain Tools

Der Optimizer von DB2 UDB erzeugt für eine SQL-Anweisung eine Menge von Zugriffsplänen, welche jeweils eine Kostenschätzung für die Ausführung dieses Plans enthalten. Die Kosten werden anhand von momentanen Statistiken, Einstellungen (Konfigurationsparameter), Bind-Optionen und der Optimierungs-Klasse geschätzt [IBM06c]. Nachfolgend wählt der Optimizer den voraussichtlich günstigsten Plan aus und bringt diesen zur Ausführung.

Die Analyse von Ausführungsplänen und Kostenschätzungen des DB2-Optimierers kann über die im DB2 Control Center und per Kommandozeile (db2expln) verfügbaren **Explain Snapshots** erfolgen. Ein derartiger Explain Snapshot kann insbesondere für die Beurteilung der Verwendung und Wirkungsweise von Zugriffspfaden der über die Anwendungen ausgeführten SQL-Statements wichtig sein. Explain-Informationen sind auch hilfreich, um die Performance von SQL-Anweisungen zu vergleichen bevor und nachdem Änderungen am System durchgeführt wurden. Diese Änderungen können beispielsweise das Hinzufügen oder Löschen von Indexen, Aktualisieren der Statistik-Informationen, Hinzufügen von Materialized Query Tables (MQT) oder Änderungen an der Anweisung selbst sein. Primär sind die Explain-Informationen für die Analyse von Zugriffsplänen gedacht, jedoch reichen die Analyse-Möglichkeiten weit darüber hinaus. In welcher Weise die darin enthaltenen Informationen für das Applikations- und System-Tuning hilfreich sein können, wird ausführlich in [Hen07] und [IBM06c] erläutert.

DB2 UDB bietet ein reichhaltiges Angebot an Werkzeugen (**Explain Tools**) zur Analyse von Zugriffsplänen, die der Optimizer ausgewählt hat. Informationen zu statischen und dynamischen Anweisungen werden in sog. Explain-Tabellen gesammelt und können mit Hilfe folgender Tools gewonnen bzw. analysiert und visualisiert werden [IBM06b]:

- *Visual Explain*: Die Visual Explain Facility ermöglicht die grafische Ansicht und Navigation von Zugriffsplänen für Anweisungen auf DB2-Datenbanken beliebiger Instanzen.
- *db2exfmt*: Dieses Werkzeug erzeugt aus den Explain-Informationen eine vorformatierte Ausgabe.

- *db2expln* und *dynexpln*: Das Kommando *db2expln* wird im Command Line Processor verwendet, um die aktuellen Zugriffspläne für statische SQL-Anweisungen abzurufen. Dabei werden keine Informationen des Optimizers angezeigt. Für dynamische SQL-Anweisungen ohne Parametermarker eignet sich das Werkzeug *dynexpln*, welches für das dynamische SQL ein Pseudo-Package erzeugt und für dieses mittels db2expln die Explain-Informationen abruft.

Alternativ zu den dargebotenen Möglichkeiten lassen sich jederzeit selbst SQL-Anfragen gegen die Explain-Tabellen absetzen. Detaillierte(re) Informationen über die Benutzung von Explain Snapshots und den Explain Tools sind [IBM06j] zu entnehmen.

5.3.1.6 Protokolldateien

Eine weitere Informationsquelle für die Diagnose sind **Protokolldateien**, welche Informationen aller Art aufzeichnen. Diese Dateien sind nicht nur auf DBMS-Schicht sondern auch auf den anderen Schichten eines Systems zu finden. Dazu zählen sowohl Speicherabbild-Dateien wie Core-, Dump- und Trap Files als auch gewöhnliche Log-Dateien (wie etwa das Transaction Log, DB2 Diagnostic Log und das Administration Log). Die Log-Dateien zeichnen dabei bestimmte Ereignisse auf, während die Speicherabbild-Dateien den Inhalt entsprechender Hauptspeicher-Bereiche von abgebrochenen Prozessen enthalten. Eine genaue Beschreibung der Dateien und deren potentiellen Analysen lassen sich in [IBM06k] nachlesen.

Für eine manuelle Problemdiagnose sammelt das Werkzeug **db2support** alle wichtigen, über die Standard-Mechanismen protokollierten Informationen zusammen, damit diese z.B. an den Support von DB2 UDB für eine weitere Analyse geschickt werden können. Zudem eignet sich das Werkzeug auch dazu, diese Informationen für die eigene Diagnose aus dem laufenden System zu extrahieren und offline zu analysieren.

5.3.1.7 Tracing

Um das Verhalten von Anwendungen zu analysieren, werden typischerweise **Traces** verwendet. Diese zeichnen beispielsweise von Anwendungen die Reihenfolge der Aufrufe und die dabei übergebenen Parameter auf. Im Rahmen einer Bottleneck-Analyse (siehe Abschnitt 2.1.3) lässt sich anhand der Zeitstempel der Aufrufe im Nachhinein feststellen, wo wie viel Zeit verbraucht wurde [IBM06k]. Allerdings ist dabei der damit und mit der rapide wachsenden Datenmenge verbundene Overhead nicht außer Acht zu lassen.

Anwendungen, die über SQLJ, JDBC, ODBC oder das CLI mit DB2 UDB kommunizieren, können das Tracing aktivieren. Erwähnenswert in diesem Zusammenhang ist der IBM CLI Trace Parser[32] zur Analyse der Trace-Dateien. Das Werkzeug bereitet die Trace-Informationen ähnlich einem Snapshot auf. Detailliertere Informationen und auch genügend Beispiele sind in [Gao04, IBM06l, Ree05, Smi04] zu finden.

5.3.1.8 Activity Monitor

Bei dem lediglich über einen Wizard konfigurierbaren **Activity Monitor** [IBM06i] handelt es sich um ein Werkzeug zur Analyse der Performance von Anwendungen, einzelnen Anweisungen, dem Sperrverhalten und der Ressourcen-Auslastung auf Ebene einer Datenbank bzw. Datenbank-Partition. Dazu liefert der Activity Monitor vordefinierte Berichte, die auf Snapshots basieren, um eine komfortable Analyse zu ermöglichen. Für einige der Berichte kann das Werkzeug sogar Empfehlungen für Aktionen aussprechen, die bei der

[32] ftp://ftp.software.ibm.com/ps/products/db2/tools/CLITraceParser.zip

Diagnose und dem Tuning unterstützen sollen und automatisch ausgeführt werden können [IBM06i].

Um das Ergebnis der Aktionen zu kontrollieren, werden nachfolgend erneute Informationen gesammelt und in einem Bericht zusammengefasst. Konnte das Problem nach dem Vergleich des alten mit dem neuen Bericht korrigiert werden, ist die Arbeit beendet. Andernfalls wird der Activity Monitor iterativ eingesetzt und basierend auf den momentanen Berichten fortgesetzt, oder im Falle einer Erschöpfung sämtlicher Möglichkeiten neu eingerichtet.

5.3.1.9 Memory Visualizer

Der Memory Visualizer ist ein Werkzeug zur Visualisierung des (Haupt-)Speicher-Verhaltens einer Instanz oder einzelner Datenbanken innerhalb einer Instanz. Dabei können einzelne Speicher-Komponenten ausgewählt und deren Verhalten bzw. Auswirkungen zum aktuellen Zeitpunkt oder über einen Zeitverlauf visualisiert werden [IBM06i].

Zu den darstellbaren Speicherkomponenten zählen u.a. der Database Global Memory, Database Manager Shared Memory, Application Global Memory, Application Shared Memory, Agent Shared Memory und Agent Private Memory.

Die Informationen unterscheiden sich nicht wesentlich von denen eines entsprechenden Snapshot Monitors. Dieses Werkzeug eignet sich jedoch für die Problem-Analyse dahingehend, als dass die Auswirkungen von Änderungen bzw. Aktionen auf Speicher-Ressourcen direkt aus internen DB2-Strukturen, ohne über den (teureren) Umweg der Snapshots bzw. Events, angezeigt werden können.

5.3.1.10 Problem Determination Tool

Ein sehr nützliches Werkzeug ist das **Problem Determination Tool** (db2pd), das seit der Version 8.2 in DB2 UDB enthalten ist [IBM06m, IBM06k]. Dieses liest Informationen direkt aus dem Speicher von DB2 UDB aus ohne dabei, wie etwa der System Monitor, Sperrmechanismen oder Ressourcen von DB2 UDB zu beanspruchen. Dadurch arbeitet es sehr performant, beansprucht kaum Ressourcen und kann ohne merklichen Overhead des öfteren aufgerufen werden, um Informationen zu gewinnen.

Mit dem Problem Determination Tool können die gängigsten der Snapshot-Informationen bei wesentlich geringerer Auslastung des überwachten Systems abgerufen werden. Allerdings sind die Ausgaben recht Weiterverarbeitungs-unfreundlich. Sie erfolgen in textueller, tabellarischer Form mit kryptischen, wenig sprechenden Spalten-Bezeichnern.

In [Ree05] beschreibt der Autor an konkreten Beispielen, wie sich dieses Tool gezielt zur Performance-Problem-Analyse einsetzen lässt. Auch in [Eat07a - Eat07f] finden sich umfangreich dargestellte Einsatz-Szenarien zum Monitoring von Buffer Pools, Tabellenzugriffen, dem Sperrverhalten und der Log-Auslastung.

5.3.1.11 Fazit

Wie schon eingangs angedeutet, lassen sich die hier vorgestellten Werkzeuge in zwei Kategorien einteilen. Wir unterscheiden Tools zum reinen Sammeln und jene zum Auswerten der gesammelten Informationen. Erstere sind im Einzelnen der Database System Monitor mit Snapshot Monitor und Event Monitor, der Health Monitor, der sowohl Daten sammelt als auch auswerten kann, die Explain Snapshots, Protokolldateien, das Tracing sowie das Problem Determination Tool. Zur zweiten Gruppe zählen demnach der Health Monitor, der Activity Monitor, der Memory Visualizer, der Event Analyzer und die Explain Tools.

Für die Gewinnung von Performance-Daten im autonomen Datenbank-Performance-Tuning sind diese DB2-internen Werkzeuge als Datenlieferanten wichtig. Die Werkzeuge sollten in eine gesamtheitliche Monitoring-Architektur integriert, ihre unterschiedlichen, gesammelten Ausgaben vereinheitlicht, zentral gespeichert und damit für spätere Analysen zugreifbar gemacht werden. Es sollte dabei allerdings beachtet werden, dass die Tools nicht durchgehend disjunkte Informationen sammeln und sich durchaus in ihren Anwendungsbereichen überlappen können.

Werkzeuge zum (reinen) Auswerten der Informationen eignen sich jedoch nicht wirklich bzw. nur begrenzt für das autonome Datenbank-Performance-Tuning. Primär haben diese zum Ziel, dem DBA die gesammelten Informationen zu präsentieren. Dafür werden die Inhalte vorverarbeitet und intuitiv, nutzerfreundlich, meist über ein graphisches User Interface dargestellt.

Die DB2 Advisor-Mechanismen (z.B. Configuration und Design Advisor) mit ihren umfangreichen Analyse- und parametrisierbaren Konfigurationsmöglichkeiten sind indes prädestiniert für die automatische Optimierung. Die Tools verfügen über (remote) Ansteuerung-Möglichkeiten via API bzw. Kommandozeile und sind dafür ausgelegt, im laufenden Betrieb möglichst störungsfrei Empfehlungen bzgl. der aktuellen Workload zu vermitteln und ggf. automatisch umzusetzen. In Abschnitt 8.3 demonstrieren wir bspw. den Einsatz des Design Advisors im Rahmen eines Tuning-Workflows zur Optimierung der Index-Ausnutzung.

5.3.2 DBMS-externe Tools

Aufgrund der immer komplexer werdenden Systeme und des damit verbundenen steigenden Anspruchs an das Performance-Management, versuchen diverse Hersteller die Administratoren mit Hilfe verschiedener Produkte und Technologien bei ihrer Arbeit zu unterstützen.

5.3.2.1 Performance Expert

Der DB2 Performance Expert (PE) ist ein Tool zur Performance-Überwachung, -Analyse und zum Performance-Tuning von DB2-Datenbanksystemen. Er ermöglicht einen umfassenden Überblick über Leistungs-Informationen eines DB2-Systems bzw. aller Subsysteme auf verschiedenen Plattformen. Detaillierte Beschreibungen zu den Daten, Funktionalitäten und möglichen Einsatzgebieten finden sich unter [CBM+06, IBM08, Wie05, Wei10].

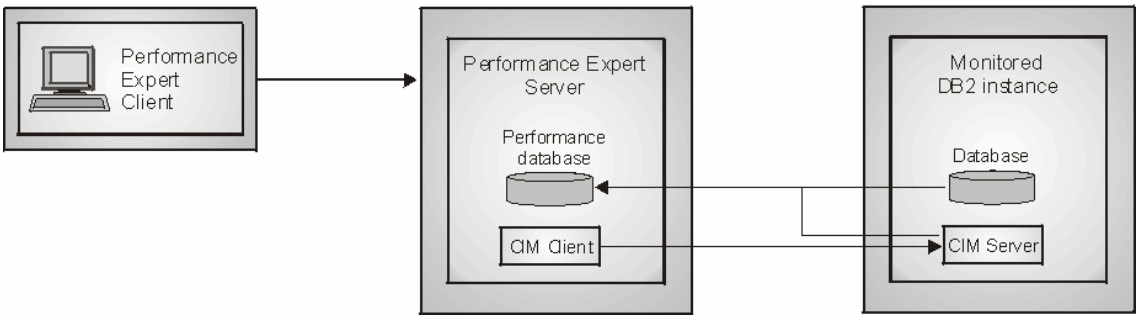


Abbildung 5.6: Grundlegende Architektur des PE [IBM08]

PE besteht aus einer Server- und einer Client-Komponente (siehe **Abbildung 5.6**). Der **Performance Expert Server** sammelt in Nutzer-definierbaren Intervallen Snapshot-Monitor- und teilweise Event-Monitor-, Konfigurationsparameter- und optional auch Betriebssytem-Daten. Ein PE Server kann dabei mehrere Systeme überwachen und speichert die Performance-Daten der überwachten DB2-Instanzen in seiner **Performance-**

Datenbank. Der PE unterscheidet zwischen *temporären*, *short-term-* und *long-term-Performance-Daten*, mit unterschiedlicher Lebensdauer und Detaillierungsgrad [CBM+06]. Daneben enthält die Performance-Datenbank auch Metadaten über die gespeicherten Daten, die Tabellen und ihre Spalten sowie Konfigurations-Informationen für den Performance Expert Server und die von ihm angebotenen Funktionalitäten. Mit dem **PE Client**, der eigentlichen Benutzer-Oberfläche, werden die vom Server angeforderten Daten graphisch angezeigt und analysiert. Über den Client können zudem auch das Verhalten des PE Server und seiner überwachten Daten-Quellen konfiguriert sowie spezifische Aktionen gestartet werden.

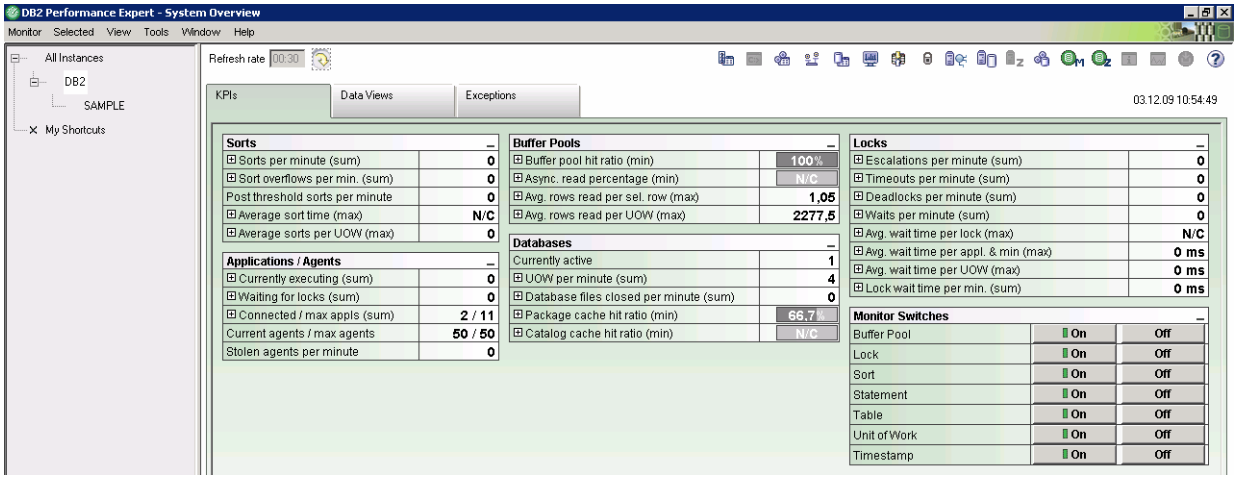


Abbildung 5.7: PE Client - System-Übersicht

Die Client-GUI (siehe **Abbildung 5.7**) bietet Auswerte-Möglichkeiten auf den Performance-Daten mittels folgender in [Wei10, Wie05] und [CBM+06, IBM08] ausführlich und anhand diverser Screenshots veranschaulichter Kategorien:

- *Statistic & Operating System Details*: Hierüber können Monitor-Elementen bzw. Statistiken über die Instanzen, Datenbanken, Tablespaces, Tabellen, Bufferpools und den dynamischen SQL Statement Cache visualisiert werden. Mit Hilfe dieser Daten und verschiedener Anzeigemodi lassen sich bestehende Performance-Engpässe online, reaktiv analysieren und u.U. beheben. Die Anzeige von dynamischen SQL Statements aus dem DB2 Package Cache hilft darüber hinaus, problematische Statements schnell zu ermitteln. Für jedes einzelne kann mittels Visual Explain der Zugriffspfad angezeigt werden. Da Performance-Probleme evtl. auf Engpässe im Betriebssystem zurückzuführen sind (z.B. Speicher, CPU, Plattenplatz), können über eine CIM-Schnittstelle (vgl. Abschnitt 5.2) optional auch Betriebssystem-Daten über die Zeit gesammelt und angezeigt werden.
- *System Parameters*: Diese Kategorie ermöglicht die Anzeige von Informationen über DB- und DBM-Konfigurationsparameter. Konfigurations-Daten werden über die Zeit gesammelt, um zu ermitteln, ob evtl. eine Veränderung in der DB2 Konfiguration Ursache für ein Performance Problem war.
- *System Health*: Im Rahmen der System Health bietet der PE Client eine graphische und individuell anpassbare Darstellung verschiedener Metrik-Werte-Verläufe in einer Gesamtübersicht (System Overview Window). Diese ermöglicht den Administ-

ratoren einen schnellen Blick auf die Gesundheit der überwachten Datenbanken und deren Aktivitäten in Echtzeit.

- *Application Summary/Details*: Die Application Summary gibt einen Einblick in die momentan aktiven Anwendungen, deren SQL Statements, Sperr-, Sortier- und Caching-Verhalten, die Bufferpool-Nutzung und die dafür benötigten (Lauf-)Zeiten. Über Sortierung oder Filterung kann der DBA schnell sehen, welche Applikationen z.B. viel CPU-Zeit verbrauchen, sortieren, Sperren halten oder lang-laufende SQL-Statements ausführen. Für jede einzelne Applikation lassen sich zudem detailliertere Informationen anzeigen, ein SQL Activity Trace (siehe weiter unten) basierend auf dem DB2 Statement Event Monitor starten oder ein „Force" zum sofortigen Abbruch spezifizierter Applikationen durchführen.
- *Applications in Lock Conflicts*: Diese Form der Analyse kann genutzt werden, um detaillierte Informationen über alle in Sperrkonflikten involvierten Applikationen und deren spezifische, verantwortliche Statements anzeigen zu lassen.

Der PE Server unterscheidet intern zwischen drei Ebenen des Performance Monitoring. Das (1) **Online-Monitoring** (near real-time) wird genutzt, um den Zustand des DB2-Systems und des darunter liegenden Betriebssystems sowie den der aktuellen Operationen zu einem bestimmten Zeitpunkt zu überwachen und auf akute Probleme (z.B. Deadlocks oder Lock Timeouts) zu untersuchen. Die Daten können dabei „auf Knopfdruck" oder in nutzerdefinierten Intervallen (in Sekunden- bzw. Minuten-Basis) in Form von Snapshots zurückgegeben werden. Sie werden für diesen Zweck lediglich einen kurzen Zeitraum temporär gespeichert. Zusätzlich zu den Snapshot-Daten in den weiter oben aufgeführten Kategorien ist das Anstoßen eines *SQL Activity Report*, sprich eines Event Monitor Trace für eine bestimmte Applikation möglich. Der Report enthält umfangreiche Informationen über die Ausführung dynamischer und statischer SQL-Statements innerhalb der zu überwachenden Applikation. Dazu zählen u.a. Start-/Stop-Zeiten, verbrauchte CPU, Rückgabe-Codes und weitere statistische Metriken.

Die Online-Monitoring-Snapshot- bzw. SQL-Tracing-Daten können individuell von jedem Client angefordert und müssen demnach auch unabhängig voneinander bearbeitet werden. Alle anderen Informationen des PE werden zentralisiert gesammelt sowie bereitgestellt und sind damit für alle PE Clients gleichermaßen verfügbar.

Das (2) **Short-Term-History-Monitoring** stellt den DBA Performance-Informationen zur Verfügung, die in einem kurzen Zeitintervall aufgetreten sind und dabei helfen sollen, bspw. Probleme der letzten Minuten bis hin zu wenigen Stunden zu analysieren und einen kurzfristigen Trend im Werteverlauf zu erkennen. Dafür müssen die Probleme nicht erst wie bei dem flüchtigen Online-Monitoring reproduziert werden. Die dafür notwendigen Daten werden in der Performance-Datenbank auf höchstem Detailgrad in einer Kurzzeit-Historie persistiert. Bei der Visualisierung der entsprechenden Monitor-Elemente über die PE Client GUI wird eine Vorverarbeitung in Form einer Delta-Bildung durchgeführt. Dem DBA sollen nicht die seit Beginn des Monitoring akkumulierten Counter-Werte dargestellt werden, sondern die Änderungen im letzten Monitoring-Intervall. Werden demnach die Statistiken minütlich durch ein entsprechend konfiguriertes Sammel-Intervall aktualisiert, dann beziehen sich die im Client angezeigten Metrik-Werte auf die Geschehnisse der letzten Minute.

Ziel des (3) **Long-Term-History-Monitoring** ist die langfristige Speicherung der Snapshot- und Event-Monitoring-Daten. Zu diesem Zweck werden die Daten des Short-Term-Schemas der Performance-Datenbank in konfigurierbaren, regelmäßigen Abständen verdichtend aggregiert und in dem sog. PE Performance Warehouse (PWH) abgelegt. Da-

durch eröffnen sich, basierend auf einer (Daten-) Historie, weitreichende Vergleichs- und Analysemöglichkeiten bis hin zur langfristigen Trenderkennung und Vorhersage.

Die Kurzzeit-Historie ist sehr feingranular und enthält typischerweise einen Aktualitäts- bzw. Detaillierungsgrad auf (Ein- bis Fünf-)Minuten-Ebene. Diese Daten werden nach einem definierten Zeitraum automatisch gelöscht, in der Regel nach einigen Wochen. Damit die Informationen über den Verlauf aber nicht gänzlich verlorengehen, können sie in die Langzeit-Historie, das PWH, übertragen werden. In dem **PWH** sind die Daten nicht mehr ganz so detailliert verfügbar, da es aggregierte Daten aus der Kurzzeit-Historie bzw. eines SQL Activity Report enthält. Für Metriken vom Typ Counter werden Delta-Werte über das spezifische Intervall berechnet, Gauge-Werte werden gemittelt und für die Aggregation von Water Marks wird jeweils der höchste bzw. geringste Wert des Intervalls bestimmt. Das kürzeste Aggregations-Interval beträgt 15 Minuten. Die gängige Praxis liegt bei dem Default von einer Stunde. Da die Daten im PWH nicht automatisch gelöscht werden, hat man die Möglichkeit, diese über mehrere Monate oder Jahre zu halten und für eine proaktive Performance-Analyse oder bei der Kapazitätsplanung einzusetzen. Im PWH stehen für diesen Zweck zusätzlich Analysefunktionen wie Reporting, Rules-of-Thumb, Queries, Trendanalyse und Trendvorhersage zur Verfügung [IBM08, Wei10].

Die Nomenklatur der PE-Langzeit-Historie deutet auf eine DWH-Struktur hin. Fakt ist jedoch, dass das PWH lediglich ein Schema mit strikt relationalen Objekt-Strukturen in der PE Performance-Datenbank darstellt. Daher kann es rein formell nicht wirklich als ein Warehouse gesehen werden. Als ein Read-Only-Repository zur Integration von Performance-Daten aus verschiedenen Quellen bietet es jedoch beim näheren Betrachten zahlreiche Warehouse-ähnliche Funktionalitäten:

- *Extract*: Hierzu zählen die regelmäßige Sammlung und Speicherung von DB2 System-Snapshots über DB2-Objekte (DB, Bufferpool, Tabelspace, Table etc.) sowie die automatische Extraktion von DBM- und DB-Konfigurationsparameterwerten bei Änderung in die Performance-Datenbank.
- *Transform*: Die DB2-Snapshot- und -Event-Monitor-Daten werden auf ein relationales Modell abgebildet. Neben dem reinen Schema-Mapping finden auch Vorschriften zur Aggregation bzw. zur Delta-Bildung Anwendung, je nach gewünschter, nutzerdefinierter Granularität.
- *Load*: Periodisch können die aggregierten DB2 Snapshot-Daten aus dem Short-Term-Bereich der Performance-Datenbank in die PWH-Strukturen überführt werden.

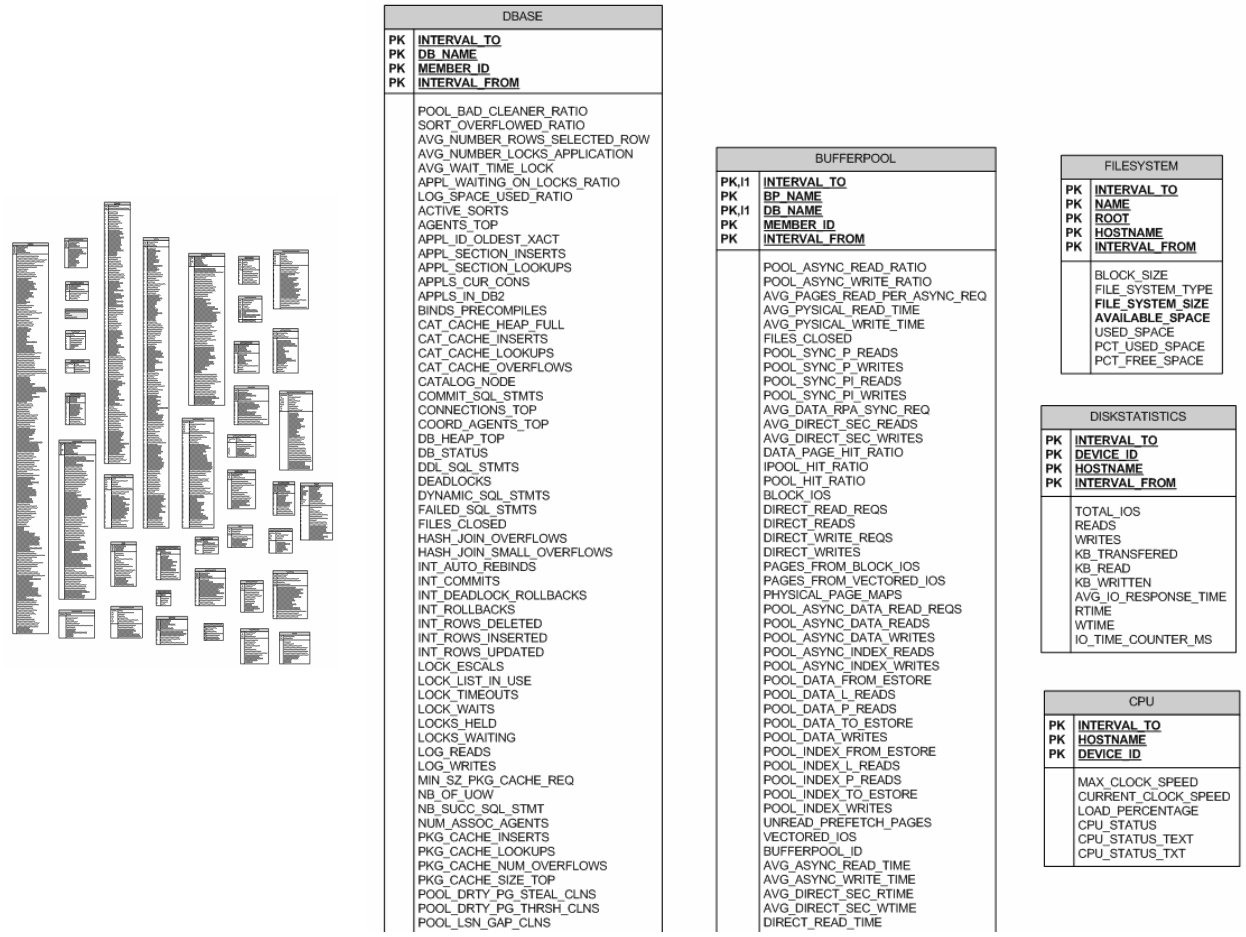
In Abschnitt 8.2 greifen wir diese grundsätzlichen Ideen im Rahmen der Vorstellung unserer Performance-Monitoring-Infrastruktur auf und erweitern sie um für das autonome Datenbank-Tuning-spezifische Aspekte.

Da DBAs i.d.R. wenig Zeit haben, um sich kontinuierlich einen Überblick der überwachten Ressourcen zu verschaffen, sind in den PE Funktionalitäten zur automatischen Ausnahmebehandlung (**Exception Processing**) integriert. Das heißt, sobald eine Veränderung in einem Bereich der Datenbank stattfindet, über die der DB-Administrator informiert werden will, schickt die Exception-Processing-Komponente eine (warnende bzw. alarmierende) Nachricht an den Administrator oder macht sich im PE-Client durch ein Popup, farbige Signale oder einen Sound bemerkbar. Auf Wunsch können auch eine E-Mail verschickt oder eine User-Exit-Routine zur Konvertierung und Weiterleitung dieses Ereignisses an externe Programme ausgeführt werden. Ein User Exit ist ein vordefinierter Austrittspunkt,

zu dem eine vom Anwender geschriebenen Routine die Kontrolle übernehmen kann. Damit ist es im Prinzip möglich, jede erdenkliche Aktion bzw. Aktionsfolge, wie bspw. ein Skript-Lauf oder das Starten eines Tools, durchzuführen.

PE unterscheidet zwischen zwei Arten der Ausnahme-Behandlung:

- *Periodic Exception Processing*: Eine Periodic Exception tritt auf, sobald die von Benutzern definierten Schwellwertgrenzen (thresholds) für bestimmte Performance-Metriken über- bzw. unterschritten werden. Diese Art der Ereignis-Erkennung basiert auf der automatischen Auswertung der im Rahmen des Short-Term-History-Monitoring gesammelten Snapshot-Daten. DBAs können die Exception-Processing-Funktionalität nutzen, um vordefinierte Alerts für OLTP- oder BI-Workloads bzw. auch selbst definierte Alerts auszulösen. An verschiedene Workload-Typen angepasste Schwellwerte sind notwendig., da z.B. in einer OLTP-Workload eine BPHR zwischen 90% und 100% liegen sollte, wobei in einer BI-Workload schon eine Buffer Pool Hit Ratio von 70% bis 80% ausreichend sein kann (siehe Abschnitt 7.3 und 8.3).
- *Event Exception Processing*: Dieser Modus basiert auf den Daten des Event Monitor. Derzeit können jedoch ausschließlich Deadlock Events überwacht werden. Somit treten jene Deadlock Event Exceptions immer dann auf, wenn ein Deadlock in der betrachteten Datenbank erkannt wird.



(a) Übersicht PE-DB-Schema (b) Auszug aus dem PE-DB-Schema

Abbildung 5.8: Schematischer Auszug aus der PE Performance-Datenbank

Abbildung 5.8 (a) soll abschließend schematisch den Umfang und die Breite der Tabellen (im Sinne der Zahl der erfassten Metriken) der Performance-Datenbank verdeutlichen. **Abbildung 5.8 (b)** hingegen, stellt exemplarisch einige der unverknüpften Performance-Daten-Tabellen dar. Für jedes der Monitor Level existiert im Wesentlichen eine eigene Tabelle, die sämtliche dem Snapshot Monitor Level zugeordnete Metriken als Spalten umfasst und darüber hinaus eigene zusammengesetzte Metriken (wie bspw. die BPHR) beinhaltet.

5.3.2.2 IBM Tivoli Monitoring (for Databases)

Mit **Tivoli** vertreibt IBM eines der umfangreichsten und am weitesten verbreiteten System-Management-Softwaresysteme[33] und bietet u.a. Unterstützung für die Bereiche der Verfügbarkeits-Überwachung, des Konfigurations-Management, Sicherheits-Management, Speicher-Management und dem Management der Mainframe-Betriebssysteme z/OS und OS/390.

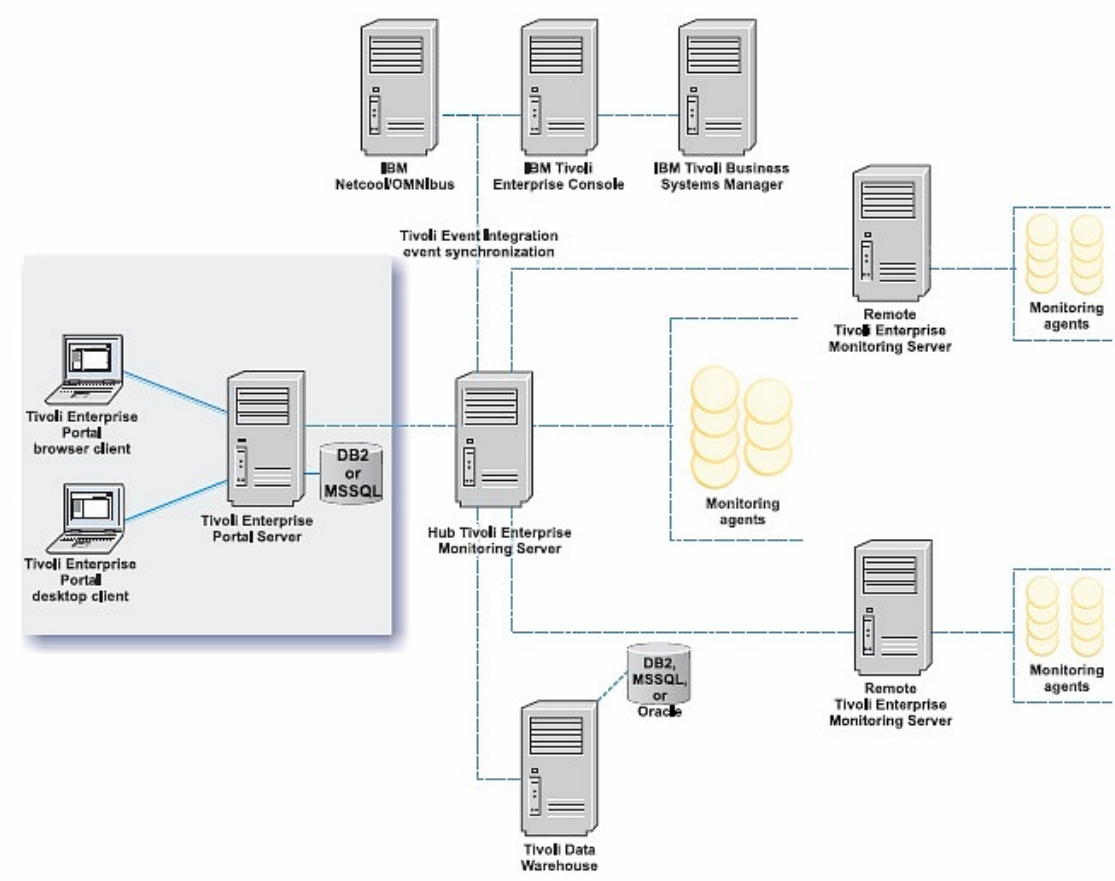


Abbildung 5.9: Tivoli Monitoring for Databases - Architektur [IBM08a]

Das Subsystem **IBM Tivoli Monitoring** (ITM) dient speziell der Überwachung und Kontrolle von kritischen Hardware- und Software-Komponenten. Hierzu gehören u. a. Betriebssysteme, Datenbanken und Anwendungen in verteilten Umgebungen [IBM08a, IBM08b, IBM09, Wei10]. IBM Tivoli Monitoring hilft bei der Verwaltung großer, heterogener Implementationen durch die ständige Überwachung wesentlicher Systemressourcen, die automatische Erkennung von Engpässen und potentiellen Problemen sowie der Reak-

[33] Neben Tivoli sind u.a. noch OpenView von HP und Patrol von BMC in diesem Bereich von Bedeutung.

tion auf Ereignisse. Idealerweise sollen Probleme bereits in einem frühen Stadium erkannt und dadurch schnell behoben werden können, noch bevor sie sich auf Endbenutzerseite negativ auf die Systemleistung auswirken. Als Zielplattformen werden die UNIX-Derivate AIX, HP-UX, Linux, OS/400 und Solaris sowie Windows NT und Windows 2000 unterstützt. Im Speziellen für unsere Zwecke interessant ist die Komponente **IBM Tivoli Monitoring for Databases** für DB2, Oracle, Microsoft SQL-Server und Sybase, als Lösung für die vereinfachte Überwachung und Verwaltung der Ressourcen einer Backend-Datenbank-Infrastruktur.

Im Gegensatz zu dem relativ kompakt aufgebauten PE besteht ITM im Wesentlichen aus den in **Abbildung 5.9** dargestellten Komponenten [IBM08a]. Im Mittelpunkt der Architektur steht der IBM **Tivoli Enterprise Portal Client**, ein anpassbarer Desktop- oder Browser-Client, der eine durchgängige Sicht und Überwachung der Systemleistung bietet. Über diesen zentralen Verwaltungspunkt wird die Sammlung von Überwachungsdaten zu Anwendungen und Ressourcen von (Sub-)Systemen aus Monitoring-Agenten und anderen Quellen gesteuert.

In **Abbildung 5.10** ist die System-Übersicht des Tivoli Enterprise Portal Client abgebildet. Die Performance-Daten und die aktuelle Situation werden über sog. Arbeitsbereiche (Workspaces) in Form von Diagrammen und Tabellen dargestellt. Es ist möglich zwischen den vorhandenen Arbeitsbereichen zu wechseln bzw. diese in gewissem Umfang zu bearbeiten, um sich beispielsweise mehr Informationen anzeigen zu lassen oder aber einfach andere Darstellungsformen zu wählen. Über diese Workspaces können die verschiedenen Überwachungs-Funktionen an die Anforderungen der DBA angepasst werden, ohne Programmcode schreiben zu müssen.

Der Tivoli Enterprise **Portal Server** ermöglicht die Bearbeitung und die Analyse der durch die Monitoring-Agenten erfassten Daten und befindet sich architekturell zwischen dem Client und dem Tivoli Enterprise Monitoring Server. Er dient sowohl als zentrale Kommunikations-Instanz für die Clients, als auch als Sammelstelle für die angelieferten Monitoring-Daten.

Die Tivoli Enterprise **Monitoring Agents** sammeln Daten von dem zu überwachenden System. Jeder Agent sendet Reports an den Monitoring-Server, von welchem er Instruktionen zum Sammeln von Performance- und Verfügbarkeits-Daten des zu überwachenden Systems bekommt. Die Agenten senden zudem in regelmäßigen Abständen ein Heartbeat-Signal an ihren verantwortlichen Monitoring Server, um ihre Verfügbarkeit und den Betriebs-Status mitzuteilen.

Die Monitoring-Umgebung kann sowohl aus mehreren Tivoli Enterprie Monitoring Agents, als auch aus vielen Tivoli Enterprise Monitoring Servern bestehen. Ein Tivoli Enterprise **Monitoring Server** steuert einen oder mehrere Monitoring-Agenten, der die Daten zur Erfassung, Filterung, Korrelation und Aufbereitung für die Ursachen-Analyse übergibt. Neben der Überwachung der Verfügbarkeit der Agenten und dem Erstellen von Reports, kann der Monitoring Server auch gewisse Systembedingungen auswerten und im Falle einer kritischen Situation mit der Ausführung (bzw. Weiterleitung) einer Problem-lösenden *Policy* reagieren. Die Informationen über das verwaltete System, kritische Situationen und Policies werden innerhalb einer dem Monitoring Server eigenen Datenbank, der **Enterprise Information Base**, verwaltet.

Analog zu PE gibt es bei Tivoli unterschiedliche Monitoring Level bzw. Ebenen der Datenhaltung. Man differenziert auch hier zwischen den Online-, Short-Term und Long-Term-Monitoring-Daten. Das **Tivoli Enterprise Data Warehouse (TEDW)** ist verantwortlich für das langfristige Speichern von Performance-Daten, die von Agenten in ihrer

Umgebung erfasst wurden. Die Monitoring Agents und Server senden hierzu periodisch ihre Kollektionen an Short-Term-Daten an den **Warehouse Proxy Agent**, damit diese in das Tivoli Data Warehouse geschrieben und dort für spätere Analysen langfristig vorgehalten werden können. Der **Warehouse Summarization und Pruning Agent** ist ein spezieller Server-Prozess, der die Größe der Warehouse-Datenbank durch regelmäßige Aggregationen und Löschung nicht mehr benötigter Daten handhabbar macht. Das Tivoli Enterprise Data Warehouse dient schließlich der Analyse von Trends bzw. zur Generierung von Warehouse Reports für die historischen Daten. Alternativ kann auch Reporting Software von Drittherstellern zur Erzeugung von Langzeit-Reports auf den Warehouse-Daten verwendet werden.

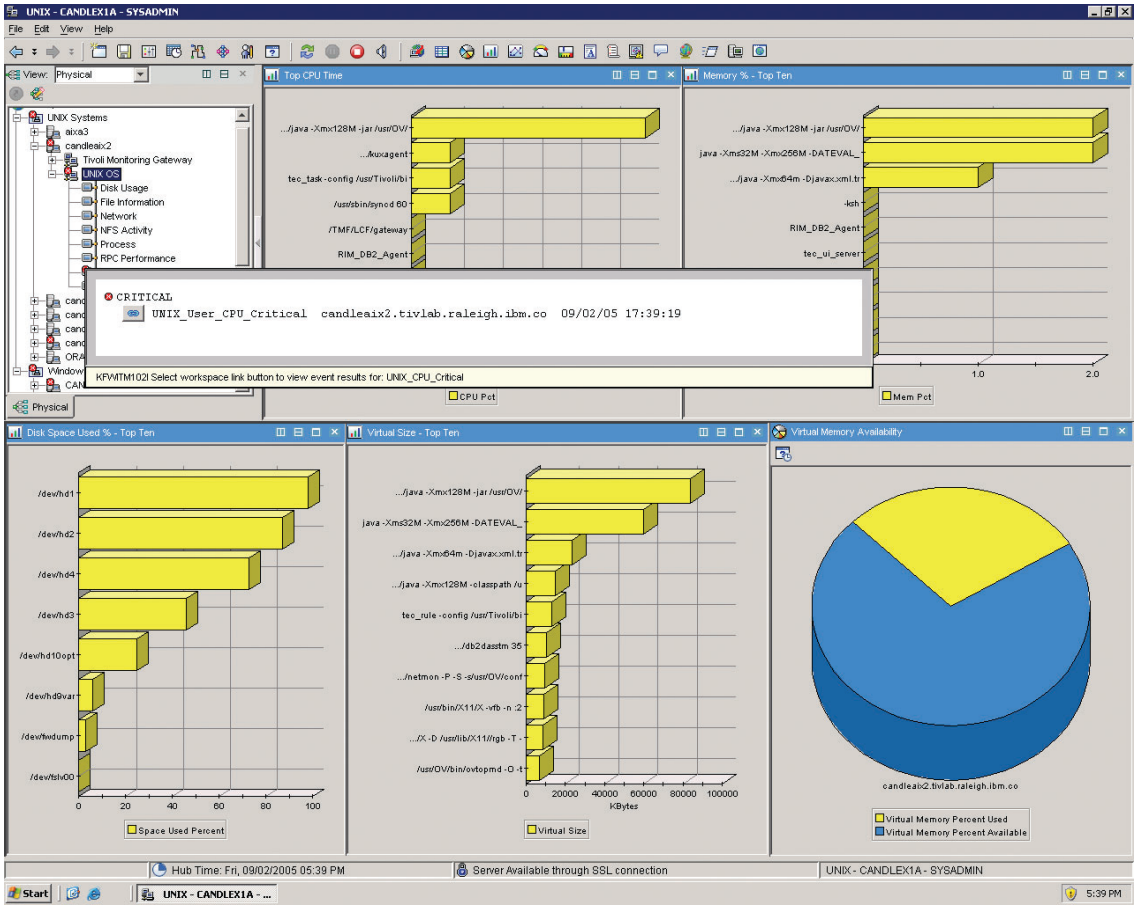


Abbildung 5.10: ITM Systemübersicht einschließlich eines Situation-Flyover [IBM09]

Genau wie PE kann Tivoli eine oder mehrere Datenbanken überwachen und verwalten. Auch Tivoli greift hierfür auf die von DB2 gelieferte Möglichkeit, Snapshots und Events zu erzeugen, zurück. Alternativ zum Einsatz des Warehouse bietet sich die Speicherung der Performance-Historie in Binärdateien an, die man bei Analysen auswerten und bei Bedarf auch selbst löschen muss. Auch bei Tivoli kann man auswählen, wie oft die Short-Term-Daten (periodisch) gesammelt werden sollen. Hierbei sind 1 Minute, 5 Minuten, 15 Minuten, 30 Minuten, jede Stunde oder jeder Tag möglich. Es besteht die Wahl, die Daten entweder bei dem Tivoli Enterprise Monitoring Agent oder dem Tivoli Enterprise Monitoring Server zu speichern. Im Unterschied zum PE ist es bei Tivoli möglich, die Datensammlung feingranularer zu konfigurieren und für die einzelnen Attributgruppen unterschiedliche Sammel-Intervalle einzustellen. Während PE für die verschiedenen überwachten Datenbanken einer Instanz eigene Schemata in der PE Performance Database

anlegt, werden bei Tivoli die Performance-Daten aller Datenbanken entsprechend vollqualifiziert in einer Tabelle bzw. in einer Datei gespeichert.

Auch die Bereiche der Datensammlung unterscheiden sich nicht wesentlich voneinander. Bei dem Tivoli-Produkt lassen sich ebenfalls detaillierte Informationen und Statistiken über die Applikationen, Bufferpool-Aktivitäten sowie den Status der Datenbank und ihrer Ressourcen gewinnen. Ferner bietet sich, ähnlich den PE Reports, die Option einer aufbereiteten Zusammenfassung/Konsolidierung der Daten über die Kapazitäten, Leistungsvermögen und Verfügbarkeit in Form von stündlicher, täglicher oder auch wöchentlicher Betrachtungszeiträume.

Mit dem Anwachsen der Daten-Mengen werden auch Tools zur vernünftigen Verwaltung erforderlich. Durch die Löschung bzw. Aggregation/Komprimierung der Daten-Mengen können zum einen die Performance von Queries gegen das Warehouse, aber auch der benötigte Speicherplatz effizienter gestaltet werden. Der Summarization Agent bestimmt abhängig von der Konfiguration, wann die Short-Term-Daten mittels des Warehouse Proxy in das Warehouse kommen. Die Short-Term-History-Daten werden bei Tivoli nach dem Upload auf das Long-Term-Repository, das TE Data Warehouse, automatisch gelöscht. Über den Summarization-Parameter kann konfiguriert werden, in welchen Zeitabständen aggregiert wird. Das Warehouse-Intervall gibt an, wie oft die Daten in das Warehouse transferiert werden sollen. Im Gegensatz zu PE bietet Tivoli über den Pruning Agent und den Pruning-Interval-Parameter die Möglichkeit, die aggregierten Daten im Warehouse automatisch zu löschen. PE hingegen löscht die Daten aus seinem Long-Term-Repository (PWH) nicht selbstständig. Eine Trennung der Aggregation von dem Zuführen in das Warehouse ist beim PE zudem auch nicht möglich. Die Daten werden hierbei unmittelbar vor dem Laden aggregiert.

Die Datengewinnung von ITM basiert vollständig auf **CIM** (vgl. Abschnitt 5.2). Demnach kann alles, was über eine CIM-Instrumentierung verfügbar ist, auch überwacht werden. Als CIMON bedient sich ITM unter Windows des integrierten WMI, um zusätzliche Parameter und Metriken vom Betriebssystem zu erhalten. Die verfügbaren Klassen werden dabei über das WMI-Repository gelesen. Die Erstellung von zusätzlichen (CIM-)Datenquellen bzw. Überwachungs-Entitäten ist dabei mit der Distributed Monitoring Workbench und dem Windows-MOF-Compiler zum Laden externer Klassen möglich [IBM08a]. Da die Informationen durch die Verwendung eines CIM-Modells systemübergreifend standardisiert sind, können alle Ressourcen von einer einzelnen Workstation aus überwacht werden.

Der funktionale Umfang von der Tivoli-Monitoring-Ereignisbehandlung ist vielversprechend. Ressourcen werden mittels sogenannter **Resource Models** überwacht, die sich zusammensetzen aus den folgenden Eigenschaften und Funktionalitäten [IBM08a, MCG+02]:

- der zu überwachenden logischen bzw. physischen Ressource selbst
- der Protokollierung der Messwerte zur Erkennung von Langzeit-Tendenzen
- auf Schwellwerten basierende Symptome, die Probleme (**Situationen**) beschreiben (z.B. zu hoher Plattenfüllstand, zu hohe CPU-Auslastung, zu viele Prozesse etc.)
- einer optionalen Korrelationsanalyse. Mehrere Symptome können zu einem Fehler zusammengefasst werden
- optionalen bewährten Maßnahmen (**Policies**), die versuchen, auf Symptome zu reagieren und das Problem bzw. den Fehler zu beheben

Auf solche Situations-bedingte Ereignisse kann durch akustische Signale oder entsprechende graphische Hinweise im Client hingewiesen werden. Wird ein Problem festgestellt, kann es optional auch an die Tivoli **Enterprise Console** (TEC) weitergereicht und dort in Zusammenhang mit anderen Ereignissen gebracht und verarbeitet werden.

Jeder Monitoring-Agent wird mit einer Reihe vordefinierter Situationen geliefert, um unmittelbar beim Start des Tivoli Enterprise Portal mit der Überwachung zu beginnen. Die Schwellwerte dieser Situationen können bearbeiten werden, um sie an die aktuelle Umgebung anzupassen. Weiterhin kann man auch eigene Situationen erstellen.

Die, durch das Tivoli Ressourcen-Modell eingebundenen, bewährten (best-practices-) Maßnahmen für die automatische Erkennung und Lösung von Infrastruktur-Problemen sind denen des PE sehr ähnlich, bieten aber vor allem bei der automatischen Auflösung eine bei weitem größere Mächtigkeit, bspw. zur zeitlich geplanten Ausführung von Aktionen, zum Terminieren von Arbeitsabläufen für Benutzer oder zum Automatisieren manueller Tasks. Zum Entwerfen und Anpassen von Policy-Richtlinien als Reaktion auf erkannte Problem-Situationen wird ein Tivoli-interner Workflow-Editor verwendet.

Neben den Automatisierungs- bietet ITM zahlreiche Hilfe-Funktionen. Indem der Nutzer die Maus über den entsprechenden Alert bewegt (bspw. in dem Flyover aus Abbildung 5.10), kann er sich ausführliche Erläuterungen zu Ereignissen und Problemen sowie zu entsprechenden Lösungsvorschlägen abrufen.

5.3.2.3 Weitere Daten-Quellen

Abgesehen von umfangreichen Performance-Überwachungs- und Datensammeltools[34] für sämtliche Hardware- und Software-Komponenten einer IT-Infrastruktur, liefert auch das Betriebssystem einige Funktionalitäten, die bei der Problem-Analyse behilflich sein können.

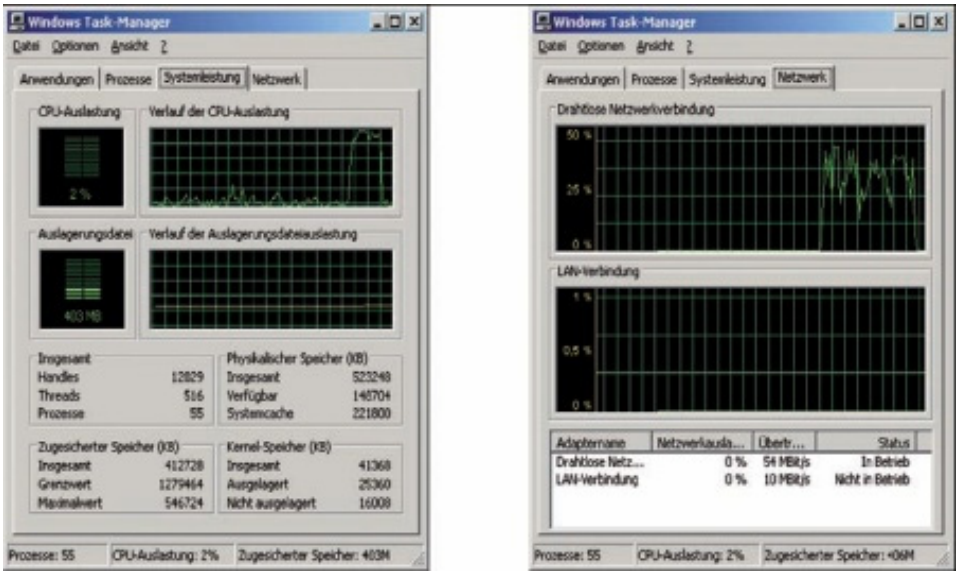


Abbildung 5.11: Der Windows Task Manager

Unter Windows sei der *Windows Task Manager* (**Abbildung 5.11**) genannt, der auf einen Blick die aktuelle Prozessor-, Speicher- und Netzwerkauslastung der jeweils letzten ca. 120

[34] Eine nahezu vollständige Auflistung sämtlicher Performance-Monitoring-Tools im DB2-Umfeld findet sich unter http://www.monitortools.com/

Sekunden anzeigt. Dies reicht möglicherweise aus, um eine kurzzeitige außergewöhnliche Belastung festzustellen, die als Ursache für einen Engpass gelten kann.

Der *Windows Performance Monitor* (**Abbildung 5.12**) ist dagegen deutlich vielseitiger und komfortabler. Auch er ist in der Lage festzustellen, ob die Server-Hardware zeitweise (oder vielleicht auch generell) mit der Datenbank und ihren Anforderungen bzw. Anfragen überfordert ist. In der Grundeinstellung stellt das Tool drei wesentliche Performane-Indikatoren zur Beurteilung der CPU-, Speicher- und der I/O-Performance grafisch dar. Diese Visualisierung lässt sich jedoch um zusätzliche (auch DB-spezifische) Indikatoren erweitern. Darüber hinaus besteht die Möglichkeit, die gemessenen Werte, wahlweise in einer Textdatei, Binärdatei oder auch (MS-SQL-Server-)Datenbank zu protokollieren, damit auch über einen längeren Zeitraum zu erfassen und später auszuwerten.

Darüber hinaus sind die Tools *netstat* bzw. *iostat* zur Überwachung des Netzwerk- bzw. der I/O-Verkehrs erwähnenswert. Als UNIX-Äquivalent sei hier bspw. der *System Activity Reporter* genannt.

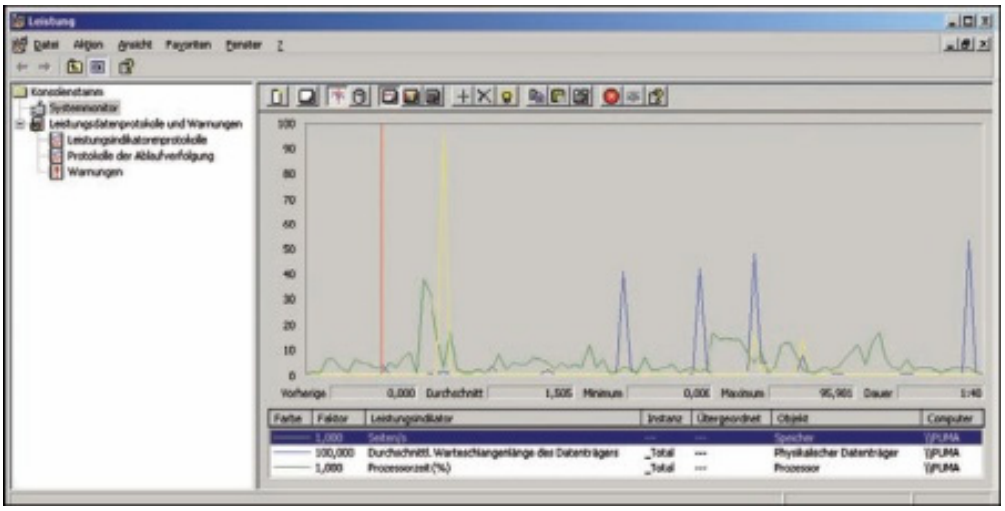


Abbildung 5.12: Der Windows Performance Monitor

5.4 Fazit

Um Performance-Probleme reaktiv bzw. proaktiv zu erkennen, deren Ursachen zu diagnostizieren und die gesammelten Informationen im Tuning-Prozess bzw. im Sinne „globaler" Entscheidungen anzuwenden, müssen die entsprechenden Komponenten bzw. deren Zustand und Verhalten **systematisch** überwacht bzw. gemessen werden. Durch die Bildung von auf gesammelten Performance-Daten basierenden Trends, können potentielle Probleme und Ziel-Abweichungen auch schon frühzeitig, vor dem eigentlichen Auftreten, erkannt werden.

Wie wir bereits in Abschnitt 2.1 motiviert haben, gibt es mehrere **Strategien** für das Monitoring. Wir unterscheiden u.a. die planmäßige, **regelmäßige** Informations-Sammlung (Routine Monitoring) über die Belastung des Systems während Normal- und Spitzenaktivitäten. Dabei sollte man auch Applikationen und einzelne Statements unter Beobachtung stellen, um frühzeitig Ressourcen-intensive Workload zu identifizieren.

Neben der kontinuierlichen Erfassung und Auswertung von Performance-Metriken, ist die Analyse von **Ereignissen** (Events), die wesentliche System-Aktivitäten und Problem-Situationen widerspiegeln, eine gängige Methode, um möglichst zeitnah zur Laufzeit Probleme und Fehlfunktionen erkennen und automatisch reagieren zu können.

5.4.1 Zentralisierte Verwaltung vielseitiger Monitoring-Daten

Zum Erfassen, Speichern, Aggregieren und Bewerten von Informationen über die Workload und die Performance von Ressourcen bieten sich dutzende auf den verschiedenen Ebenen und Komponenten wirkende „Experten-Monitore“ als Datenquellen an. Die vorgestellten Mechanismen und Tools bieten zudem erste gute Ansätze zur Automatisierung von Abläufen, wie der Problem-Erkennung und -Analyse. Klassische Monitoring-Mechanismen und -Tools überwachen bestenfalls eine Menge von Ressourcen innerhalb einer Schicht in einem Software-Stack. Angesichts der Vielzahl von Komponenten, die an einer Lösung beteiligt sein können und miteinander in Beziehung stehen oder kommunizieren, kann es sein, dass jede Komponente für sich korrekt arbeitet, das Gesamtergebnis dem Endanwender beispielsweise aufgrund fehlerhafter Kommunikation dennoch nicht performant oder korrekt zur Verfügung steht. Daher darf sich nicht darauf beschränkt werden, einzelne Hardware- und Software-Komponenten zu betrachten. Vielmehr sollte der Daten- und Informationsfluss über die Komponenten hinweg überwacht und optimiert werden.

Die meisten der Tools können Daten entweder gar nicht oder nicht dauerhaft speichern. Die Ablage der Daten in einem standardisierten Format hat ebenfalls noch Seltenheitswert. Aus unserer Sicht kann daher eine für gesamtheitliche Betrachtungen notwendige konsolidierte Speicherung und anschließende integrierte Auswertung der Daten nicht durch lediglich eines der Tools gewährleistet werden. Dieser Umstand motiviert umso mehr unsere im Rahmen der vorliegenden Niederschrift erarbeitete Produkt-unabhängige Monitoring-Architektur, die sich das Potential der Tools zur Sammlung, initialen Aufbereitung und ggf. automatischen Weiterleitung der Performance-Daten zu Nutze macht und damit auf nahezu beliebige Datenquellen „aufsetzen“ kann. Die Überwachungs-Daten aller Komponenten sollen dabei an zentraler Stelle erfasst, konsolidiert und langfristig gespeichert werden.

5.4.2 Adaptives Monitoring

Der Gesamtumfang an Performance-Daten wird maßgeblich durch die Frequenz der Sammlung und die Konfigurations-Möglichkeiten entsprechender Tools bestimmt. Insbesondere weil die Überwachung von Ressourcen bzw. deren Ereignisse einen kleinen, aber kumulativen und nicht zu verachtenden Effekt auf die Leistung des operativen Systems hat, sollte ein Kompromiss zwischen der Daten-Qualität bzw. Aktualität und dem akzeptierten Overhead zur Sammlung der Daten gefunden werden.

Idealerweise sollte das Monitoring aus diesem Grund nicht ständig betrieben, sondern *gezielt* in Umfang und Dauer an den aktuellen Kontext angepasst und vor allem im intensiven Maße zur Lösung spezieller Probleme eingesetzt werden. Daher ist es grundsätzlich eine durchaus sinnvolle Strategie, den Overhead der Sammlung auf dem zu überwachenden System so gering wie möglich zu halten und zu jeder Zeit nur soviel Daten zu sammeln wie für aktuelle oder spätere Zwecke unbedingt notwendig. Eine adaptive Monitoring-Strategie (siehe auch Abschnitt 4.1.2.1) ist ratsam, die den Detailgrad der Datensammlung maßgeblich abhängig macht von der aktuellen Performance, der anliegenden Workload und dem relevanten (Umgebungs-)Kontext. Adaptivität soll nicht nur den *Umfang* bzw. den Detailgrad, sondern auch die Art und Menge der gesammelten Daten beeinflussen. Weitere zentrale Aspekte in diesem Zusammenhang sind *Dauer* und *Art* des Monitoring. Wie wir gesehen haben, könnte man beispielsweise unterscheiden zwischen reaktiver und proaktiver bzw. auch zwischen Poll- und Push-basierter Überwachung.

Insbesondere durch *Policies* spezifizierbare Nutzer- bzw. Applikations-spezifische Anforderungen in Form von Typ, Lebensdauer, Genauigkeit oder Zeithorizont der zu gewinnenden Daten können und sollten berücksichtigt werden. Es ist vorstellbar, die zu betrachtenden Metriken bzw. auch die Symptome und kritische Wertekombinationen (und damit auch die relevanten Ressourcen) aus den in der Policy angegebenen Zielen bzw. dem durch eine Menge von in der Policy definierten Metriken anvisierten Zielzustand automatisch „abzuleiten". Zusammen mit einem Ressourcen- bzw. Abhängigkeits-Modell können auch auf den ersten Blick nicht offensichtliche Abhängigkeiten bzw. Seiten-Effekte zwischen den Metriken bei dem Monitoring in Betracht gezogen werden und es ermöglichen, nur die tatsächlich relevanten Performance-Daten für die nachfolgende Analyse zu sammeln.

Ein anderer Teil der zu observierenden Metriken ergibt sich durch das *Tuning* und dem damit verbundenen vermeintlich positiven Einfluss auf die Performance einzelner Ressourcen. Die Auswirkungen der vorgenommenen Änderungen müssen kontinuierlich überwacht und in den adaptiven Monitoring-Prozess eingebunden werden.

Im Mittelpunkt der Betrachtungen zur Adaptivität steht vor allem die Wahl der *Frequenz* des Monitoring, sprich wann und wodurch die Datensammlung angestoßen oder in ihrem Umfang angepasst wird. Das einfachste, sub-optimale Szenario ist das andauernde vollständige und kontinuierliche Sammeln von Performance-Daten in Echtzeit. Darüber hinaus kann die Überwachung bei besonders akuten Problemen und damit verbundenen hohen Anforderungen an die Aktualität der Daten sofort und ohne Berücksichtigung des assoziierten Overhead oder auch gemäß in Policies durch Nutzer definierte bzw. automatisch bestimmter Zeitfenster in (mehr oder weniger) regelmäßigen Abständen erfolgen.

Durch Lernen des (Normal-)Zustandes, dem Erkennen von (Problem-)Mustern und der darauf basierenden Vorhersage bekannter, ähnlicher Situationen in naher Zukunft, können vorbeugende Monitoring-Maßnahmen (etwa zur Reduzierung der Last oder zur Erhöhung des Detailgrades in spezifischen Problem-Bereichen) eingeleitet werden. Aber auch eine Anfrage- und Regel- bzw. Ereignis-gesteuerte Datensammlung (bzw. Anpassung) ist vorstellbar. Beispielsweise kann man bei Workload-Änderungen oder in Reaktion auf erkannte bzw. bevorstehende Performance-Probleme andere Datenbereiche in einer anderen Frequenz überwachen. In Zeiten besonders intensiver Problem-lösender Tuning-Maßnahmen kann des Weiteren eine dem DB2 angelehnten Mechanismus zur automatischen Drosselung des Monitoring zur Schonung der Ressourcen erfolgen.

Die fortwährende Überwachung und Analyse der Tuning- bzw. der Monitoring-Qualität, das Beobachten und Lernen des Lastverhaltens sowie die flexible Anpassung an sich ändernde Rahmenbedingungen (z.B. neue Policy-Anforderungen bzw. -Ziele) dürfen nicht vernachlässigt und mit wachsender Erfahrung sollen auch die Regeln der Adaptivität über die Zeit kontextsensitiv und reflektiv angepasst werden.

Kapitel 6

Verwaltung von Performance-Daten

Im Rahmen der manuellen bzw. automatischen Erkennung und -Diagnose von Performance-Problemen in IT-Systemen sollte sowohl auf eine adäquate, den Anforderungen angepasste Monitoring-Strategie als auch auf eine zuverlässige, konsistente, flexible und individuell adaptierbare Datenbasis zur maßgeblichen Entscheidungsfindung Wert gelegt werden.

Performance-relevante Daten kommen in der Praxis meistens nicht aus einem zentralen System, sondern sind auf verschiedene Subsysteme, oftmals redundant, verteilt. Die Abfrage nach bestimmten Informationen ist dadurch schwierig realisierbar und fehlerbehaftet, da verschiedene Daten-Quellen unterschiedliche Ergebnisse liefern können. Die aufgrund ihrer Komplexität und des Umfangs die Nutzer überfordernden Performance-Daten-Quellen sind traditionell zumeist relationaler oder Datei-basierter Struktur. Der Großteil an eher starren Gebilden betrachtet die Daten darüber hinaus lediglich *temporär* von einem *festen* und vor allem *lokalen* Standpunkt aus, so dass eine oft hilfreiche dynamische Änderung der Betrachtungsweise nur mit deutlichem Aufwand möglich ist.

Die relevanten Daten aus verschiedenen, möglicherweise heterogenen Quellen müssen also anfänglich sinnvoll zusammengestellt, bereinigt und geordnet werden, um eine Verknüpfung von verteilten Informationen und eine weitergehende Analyse zu erlauben. Daher bietet sich die Untersuchung und Übertragung von Techniken und Konzepten des **Data Warehousing** (siehe Abschnitt 2.3) auf die Verwaltung von Performance-relevanten Daten an.

Wesentlicher Bestandteil der vorliegenden Arbeit ist die Konzipierung eines generischen Repository zur Speicherung, Aufbereitung und nachgelagerten Analyse von Performance-Daten aus verschiedenen Quellen. Nachdem im vorangegangenen Kapitel bereits die umfangreichen Möglichkeiten zur Gewinnung von Performance-Daten aus den verschiedenen Quellsystemen aufgezeigt wurden, soll nun dargelegt werden, wo und vor allem wie die Informationen adäquat zu verwalten und bereitzustellen sind.

In dem von uns verfolgten Ansatz, dem **Performance Warehouse** (PWH), werden alle Informationen zentral und vereinheitlicht hinterlegt, um die zugrunde liegenden Ressourcen vollständig, d.h. nach allen relevanten Perspektiven in ihrem Kontext, überwachen und analysieren zu können. Dafür werden kontinuierlich alle durch die verschiedenen Monitoring-Tools gesammelten Werte von Performance-Indikatoren bzw. aufgetretenen Events in die gemeinsame Datenbasis integriert. Die Integration umfasst u.a. die Transformation, Berechnung und Aggregation Performance- und Kontext-relevanter Kennzahlen. Mittels entsprechender Werkzeuge können die Performance-Daten schließlich flexibel navigierend analysiert, den Bedürfnissen zur Laufzeit angepasst und zur Entscheidungs-Unterstützung verwendet werden.

Zunächst sollen in **Abschnitt 6.1** die aus unserer Sicht essentiellen Anforderungen an derartiges Performance-Repository aufgestellt und anschließend eine mögliche Umsetzung aufgezeigt werden. Das Kapitel deckt diesbezüglich alle zur Gestaltung und Nutzung unseres PWH relevanten Aspekte ab. Da die weiteren Abschnitte zur Konzipierung stark auf den potentiellen Inhalten des PWH basieren, widmet sich **Abschnitt 6.2** der Zusammenfassung der zu verwaltenden Daten. Das PWH besteht im Kern aus einem generischen und einem multidimensionalen Datenmodell. Wir stellen in **Abschnitt 6.3** vor allem den strukturellen Aufbau der multidimensionalen Speicherungs-Strukturen sowie eine mögliche relationale Abbildung konzeptionell, aber auch anhand einiger exemplarischer Beispiele dar. Nach der Einführung in Art und Aufbau erfolgen zudem eine kurze Diskussion der Vor- und Nachteile einer solchen multidimensionalen Persistierung sowie eine Bewertung der Dynamik und Änderbarkeit der Daten im PWH. **Abschnitt 6.4** zeigt zudem kurz auf, wie die vorgestellten Strukturen mittels bekannter ETL-Mechanismen ausgehend von beliebigen Performance-Daten-Quellen erzeugt und mit Performance-relevanten Informationen gefüllt werden können. In **Abschnitt 6.5** schließlich legen wir in einem Vorgehensmodell, dem Cube Advisor, zu Grunde, wie auf dem PWH aufsetzende Cubes zur Lösung eines bestimmten Problems zur Laufzeit durch den Administrator bzw. automatisch selektiert, erstellt und gefüllt werden können und einen für die Analyse entsprechend aufbereiteten Ausschnitt der Performance-Daten enthalten. Insbesondere die mit der Größe des Datenbestandes wachsende Komplexität langfristiger Trend-Auswertungen und tiefgreifender Analysen weckt den Bedarf für zusätzliche Optimierungs-Techniken. Dieses Thema wird im letzten **Abschnitt 6.6** im Überblick aufgegriffen.

6.1 Anforderungen an die Performance-Daten-Haltung

Die Art der Strukturierung, Speicherung und Verwaltung von Performance-Daten bestimmt maßgeblich die darauf aufbauenden Analysen zur Problem-Erkennung und Entscheidungsfindung. Entsprechend viel Bedeutung sollte man daher der Konzipierung eines Monitoring-Repository schenken. Zu den essentiellen Anforderungen an ein System zur Daten-Haltung und -Verwaltung zählen aus unserer Sicht die nachfolgenden Aspekte:

- **Analyse-orientierte Speicherung** der Daten: Die Modellierung und Speicherung der Performance-Daten soll sich an Art und Typ der nachfolgenden Analysen und Auswertungen orientieren. Die Struktur des Repository definiert sich daher vor allem durch die an es gestellten Analyse-Anforderungen.
- **Historische bis hoch-aktuelle Datenverwaltung**: Um auch langfristige Trends[35] in den Performance-Daten zu erkennen, Vergleiche zu vergangenen Status ziehen zu können und Ursachen für Probleme zu ergründen, müssen neben den aktuellen auch historische Daten verfügbar sein. Gerade die Anforderung der langfristigen Datenvorhaltung macht eine möglichst effiziente Speicherung zur Reduzierung des Speicherplatzbedarfs erstrebenswert. Der Großteil der Performance-Daten soll über einen *langen* Zeitraum hinweg verfügbar sein. Da ältere Daten zunehmend an Bedeutung verlieren, ist u.U. eine (detaillierte) Datenaufbewahrung nur über ein bestimmtes Zeitintervall sinnvoll. Über die Zeit und mit sinkender Priorität bzw. Bedeutung für Performance-Analysen kann der Detaillierungsgrad der Daten und damit der Speicheraufwand abnehmen. Für einen (anderen) Teil der Daten ist auch

[35] In der Regel geht man bei Trend-Berechnungen oder Vorhersagen davon aus, dass sich Muster aus der Vergangenheit wiederholen. Dies macht eine Erfassung und permanente Speicherung der Historie unabdingbar.

eine temporäre, *flüchtige* Speicherung ratsam, wenn der Nutzen langfristiger Aufbewahrung nicht zu erkennen ist.

- **Integrierte Auswertbarkeit**: Durch ein entsprechendes Datenmodell bzw. geeignete Verknüpfungsmöglichkeiten soll gewährleistet werden, dass Performance-Daten über verschiedene Ressourcen hinweg korreliert betrachtet werden können. Darüber hinaus soll die Verknüpfung zwischen dynamischen Performance-Daten (Metriken und Events), semi-dynamischen Struktur- und Topologiedaten und den Ablaufdaten des Tuning-Systems ermöglicht werden.
- **Darstellung** der Daten **auf verschiedenen Abstraktions-Ebenen**: Im Rahmen einer effizienten (langfristigen) Speicherung großer Daten-Mengen sowie der vereinfachten, intuitiven Analyse bzw. Navigation ist es stark von Vorteil, dass die gesammelten Performance-Daten auf verschiedenen Abstraktions- bzw. Aggregations-Ebenen mit unterschiedlichen Detaillierungs-Graden verfügbar sind. Durch die Verdichtung der Daten auf verschiedene Ebenen (z.B. durch Durchschnitts- oder auch Minimum-/Maximum-Bildung) kann eine schnelle und zuverlässige Entscheidungs-Unterstützung mittels fundierter, schrittweiser, der Arbeitsweise der DBAs angelehnten Top-Down-Analysen gewährleistet werden. Basierend auf einer Verdichtung/Aggregation der Daten kann mittels spezieller Operationen zur Laufzeit von einem höheren Aggregations-Grad auf einen kleineren gewechselt werden und umgekehrt. Data Warehousing stellt üblicherweise solche (Analyse-)Möglichkeiten bereit.
- **Vereinheitlichte** und **generische** Speicherung von Daten aus verschiedenen heterogenen Quellen unterschiedlichsten Typs: Performance-Daten können, wie oben erwähnt, einer Vielzahl an Quellen mit unterschiedlichen Formaten und voneinander differenzierenden Informationen entstammen. Um stabil gegenüber (strukturellen) Änderungen an den Quellen zu sein sowie um eine einheitliche, übersichtliche und einfach(er) zu verwaltende, gemeinsame Speicher-Struktur zu schaffen, sollten diese Daten integriert, vereinheitlicht und generisch gespeichert werden. Eine (Daten-)Modell-Anpassung zur Laufzeit entfällt genau dann, wenn Struktur-Änderungen auf operativer Ebene auf Instantiierungen bzw. Daten-Änderungen auf Repository-Ebene abgebildet werden können.
- **Erweiterbarkeit** bzgl. neuer Datenquellen: Da moderne IT-Systeme im Allgemeinen bzgl. ihrer Struktur (sowohl in Hard- als auch Software) nicht konstant bleiben, muss sich das Repository auf Struktur-Änderungen dynamisch und automatisch anpassen können bzw. durch eine generische (Aus-)Gestaltung von ihnen in der eigenen Struktur unberührt bleiben. Die generische Speicherung kann daher als Voraussetzung für eine effiziente Erweiterbarkeit gesehen werden.
- **Interoperabilität** und Werkzeug-Unterstützung: Das Repository zur Speicherung der Daten sollte zudem über eine (Programmier-)Schnittstelle für lesenden und schreibenden Zugriff sowie Import- und Export-Schnittstellen für einen standardisierten Zugriff und die Anbindung von am Markt etablierten (Sammlungs-, Transformations- und Analyse-)Tools verfügen.
- Möglichkeiten zur **Transformation** der Daten: Die Integration der Daten aus den verschiedenen Quellen (und damit Datenformaten bzw. Schemata) und deren langfristige effiziente Speicherung machen in den meisten Fällen zusätzliche Transformationen in Form von Konvertierungen, Bereinigungen, Berechnungen und Aggregationen der Performance-Kennzahlen notwendig. Diese Anforderung

ist nur von Bedeutung, wenn die Daten nicht bereits vorher in einer Staging Area bzw. im Rahmen der Datengewinnung transformiert werden.

Wie man anhand der Ausführungen in Abschnitt 2.3 sowie den hier aufgezeigten Parallelen erkennen kann und auch im weiteren Verlauf der Arbeit sehen wrid, erfüllen Techniken des Data Warehousing eine Vielzahl der Forderungen.

Unser **Performance Warehouse** (PWH) soll diesem Prinzip folgen und alle Performance-relevanten Informationen über die zugrunde liegenden Ressourcen und Aktivitäten auf dem Software-Stack zusammen mit ihren Kontext zentralisiert, integriert, konsistent und in flexibler Form bereitstellen. Damit kann das zugrunde liegende operative System vollständig, d.h. nach allen relevanten Perspektiven (Dimensionen), überwacht und analysiert werden und je nach Benutzer- und Analyse-Anforderungen verschiedene Formen der Aufbereitung und Darstellung unterstützen. Die Daten sollen in Folge einem Administrator oder den autonomen, System-verwaltenden Komponenten für weitere Analysen (z.B. Ursachenanalyse, Lernen des Normalzustands, Trenderkennung etc.) zur Wissens- und Erfahrungs-gestützten Planung und Entscheidungsfindung langfristig zur Verfügung stehen. Durch eine Langzeit-Speicherung der gesammelten Performance-/Meta-Daten ist es außerdem möglich, in der Historie nach Präzedenzfällen zu suchen und durch Korrelation zu aktuellen Situationen mögliche Lösungen, die früher zum Erfolg führten, in Erwägung zu ziehen. Außerdem lassen sich länger in der Vergangenheit zurückliegende Ursachen für aktuelle Probleme zurückverfolgen.

Das Performance Warehouse kann dabei als spezifische Ausprägung bzw. Anwendung eines DWH-Systems angesehen werden. Hierbei ist demnach auch das dem Data Warehousing gemeine, in Abschnitt 2.3 geschilderte, Vorgehensmodell für die Gestaltung und Nutzung eines derartigen PWH anwendbar. Wir legen im Folgenden das Vorhandensein von Performance-Überwachungs-Tools bzw. Sensor-Schnittstellen, die während der Laufzeit alle erforderlichen Performance-Daten aufzeichnen bzw. zur Verfügung stellen und an das PWH weitergeben können, zu Grunde. Diese Rohdaten aus den verschiedenen Quellen können zur Laufzeit automatisiert *extrahiert*, verdichtet, miteinander verknüpft und nach einer *Transformation* in die Analyse-orientierten Datenstrukturen des PWH *geladen* werden.

Dem DWH-Gedanken entsprechend, werden diese Daten im Performance Warehouse zunächst in Anwendungs-neutraler, wiederverwendbarer, aber vor allem generischer Form vorgehalten, um so, je nach Benutzer- und Analyse-Anforderungen verschiedene Formen der Aufbereitung und Darstellung zu unterstützen (z.B. in Form von Data Marts). Die Sammlung der Daten an zentraler Stelle kann dabei relational oder multidimensional erfolgen. Einzig das Datenmodell sollte *generisch* genug sein, um nicht bei Daten- oder Struktur-Änderungen auf der Quell-Ebene mit strukturellen Änderungen im zentralen Repository reagieren zu müssen. Eine automatische und zeitnahe Anpassung der Instantiierung hingegen ist gewünscht.

Insbesondere zur manuellen (Top-Down-) Analyse der Daten im PWH eignet sich anschließend der Einsatz von OLAP-Technologien, die helfen jene konsolidierten Informationen intuitiv und nach allen relevanten Perspektiven zu betrachten und damit eine Vielzahl von Möglichkeiten zur Vereinfachung sowie Beschleunigung von Entscheidungsprozessen bieten. Die Betrachtungsdimensionen orientieren sich dabei an der Natur und dem Kontext der Performance-Daten und bieten eine intuitive, einfach begreifliche Möglichkeit der Organisierung und Auswahl der Daten zur Bereitstellung und nachfolgenden Analyse.

Ein **zentrales Repository** bietet die Möglichkeit sämtliche Performance-Daten an einer zentralen physischen Lokation konsolidiert und historisiert zu verwalten. Dem Aufwand zur Bereitstellung und Pflege der (i.w.) zusätzlich redundant gehaltenen Daten steht demnach die Möglichkeit einer einzelnen, Analyse-freundlichen Datenbasis gegenüber. Ein derartiges globales Repository für sämtliche Performance-Daten kann jedoch schnell zum Flaschenhals der gesamten Monitoring-Infrastruktur (von der Gewinnung bis hin zur Analyse der Daten) werden und damit organisatorische und technische Schwierigkeiten verursachen.

Denkbar ist im Gegensatz dazu auch eine **verteilte Variante**, die mittels mehrerer Repositories für einzelne Schichten bzw. Ressourcen im Software-Stack, Datenbereiche, Werkzeuge oder Organisations-Einheiten den höchsten Grad an Unabhängigkeit und wohl einen schnellen Zugriff auf die lokalen Performance-Daten ermöglicht. Der Ansatz erfordert indes eine umfassende Synchronisation zur übergreifenden, globalen Analyse der Daten. Der Daten- und Informationsaustausch kann nur über spezifische Austauschformate und Schnittstellen erfolgen. Hierbei können Techniken verteilter Datenbanksysteme, insbesondere zur Synchronisation und Kommunikation zwischen den einzelnen Teilsystemen angewandt werden. Da gerade die gesamtheitliche Analyse der Daten im Vordergrund unserer Betrachtungen steht, ist diese Variante jedoch eher ungeeignet.

Mit der Organisationsform eines **virtuellen Performance Warehouse** hingegen würde man, ähnlich wie bei Datenbank-Sichten, auf eine separate redundante Datenhaltung verzichten. Zum Zeitpunkt der Analyse werden die Daten unter direktem Zugriff auf die operativen, heterogenen Systeme gewonnen, transformiert und ggf. aggregiert. Die Lösung ließe sich architekturell im Grunde „einfach" realisieren, würde jedoch die operativen Systeme zu stark belasten. Die Ad-hoc-Auswertung der Daten kann in diesem Fall gerade bei umfangreicheren Datenbetrachtungen (wie sie sehr häufig zu erwarten sind) unbefriedigend viel Zeit in Anspruch nehmen. Zudem ist eine umfassende Konsolidierung und Historisierung der Daten auf Basis der operativen Systeme, wenn überhaupt, nur begrenzt möglich.

Um die vermeintlichen Nachteilte eines zentralen Datenbestands zu relativieren, bietet sich der Einsatz von **Performance Data Marts** an. Darunter verstehen wir, wie in Abschnitt 2.3 bereits aufgeführt, Nutzer-, Analyse- bzw. Themen-spezifische Ausschnitte des Gesamt-Datenbestandes des zentralen Performance Warehouse. Dies bedeutet meist, dass dort sehr spezielle Daten für typische Aufgaben in u.U. anderer Organisationsform abgelegt sind. Ziel ist, dem Nutzer nur die Daten zur Analyse und Ursachenfindung aufbereitet zur Verfügung zu stellen, die er wirklich benötigt. Diese Reduktion der Komplexität ermöglicht schnellere und effektivere Analyse-Erfolge und zudem eine Entlastung des durch Einfüge-Operationen ohnehin stark beanspruchten Überwachungsdaten-Hauptbestands im PWH.

Im Falle der Performance-Daten-Speicherung ist es sinnvoll, die feingranularen Daten im PWH zu speichern, aggregierte Werte aber (teilweise) auf Data Marts auszulagern. Dadurch wird das PWH zu einem großen Teil von den Aggregations-Operationen entlastet, welche dann auf den jeweiligen Data Marts ausgeführt werden. Außerdem können Anfragen an das System möglicherweise ohne Zugriff auf das PWH bearbeitet werden. Dies ist genau dann der Fall, wenn nur auf aggregierte Werte zugegriffen wird.

Die verringerte Last (Rechenzeit) erkauft man sich jedoch, abhängig von der Anzahl und Umfang der durch die Data Marts materialisierten Abstraktionsstufen, mit einer Erhöhung des Speicherplatzbedarfs. Daher empfiehlt sich für die Aktualisierung der Data Marts,

anstatt einer kompletten Neuberechnung, eine inkrementelle Wartung, welche nur geänderte Daten des PWH und damit auch der Quellen berücksichtigt. Bei großen Daten-Mengen ist es sonst sehr wahrscheinlich, dass die Last auf dem PWH und den Data Marts inakzeptabel hoch ist und andere Prozesse behindert.

Auf ein mögliches Zusammenspiel zwischen den Quellsystemen, den verschiedenen Ebenen eines PWH und den Data Marts im Rahmen einer Performance-Monitoring-Infrastruktur wird ausführlich im Abschnitt 8.2 hingewiesen.

6.2 Inhalte des Performance Warehouse

Ein (Performance) Data Warehouse kann nur die von uns aufgestellten Anforderungen zur Daten-Verwaltung erfüllen, wenn zuvor ein adäquates Ziel-Datenmodell entworfen und im Betrieb Einsatz finden wird. Die Struktur des PWH-Datenmodells ist hingegen zum großen Anteil von den potentiellen Inhalten der für die Analysen relevanten Daten-Quellen abhängig.

Grundsätzliches Ziel des PWH ist die Abbildung und langfristige Speicherung von Performance-Daten sowie Kontext- und System-Informationen über den Zustand sowie das Verhalten des Systems mit seinen Ressourcen über die Zeit. Im Mittelpunkt des PWH stehen daher **Performance-Metriken**, die an den jeweiligen zugeordneten Ressourcen entstehen und mittels Sensoren bzw. Tools ausgelesen werden können.

Um die Performance-Werte als Funktion über die Zeit betrachten und damit sowohl reaktiv als auch proaktiv (über) bestimmte Zeiträume auf einfache Art und Weise analysieren zu können, müssen Informationen über die **Zeit** der Entstehung/ Erkennung/ Änderung vorliegen.

Damit alle Performance-Informationen über die Ressourcen des Systems eine benutzer-definierbare Zeit, zumeist kurz- bis mittelfristig verfügbar sind, müssen alle gesammelten Daten in ihrer kleinsten verfügbaren **Granularität** (sprich auf höchster Detailstufe) gespeichert werden. Durch Aggregation und Konsolidierung dieser Informationen können auch höhere Granularitäten evaluiert werden, ohne dass ein Informationsverlust vorliegt. Wenn die Daten nach einer gewissen Zeit an Bedeutung verlieren, können sie entweder archiviert oder durch eine dauerhafte materialisierte Aggregation verdichtet werden.

Für eine effektive, spätere Analyse sollen zusätzlich auch bestimmte System-Ereignisse in Form von **Events** gespeichert werden. Atomare und komplexe Ereignisse können durch entsprechende Tools bzw. die Quellen selbst erkannt und vor Ort ausgewertet und u.U. weiterverarbeitet, aber vielmehr im PWH für späteres Data Mining und zur Problem-Findung gespeichert werden. Hierbei ist es denkbar, die umfangreichen Möglichkeiten zur Beschreibung und Korrelation auf relationale Strukturen abzubilden oder einfach in standardisiertem CBE-Format (siehe Abschnitt 5.2) in einem „Large Object"- bzw. XML-Attribut zu externalisieren. Bei der späteren, retrospektiven Auswertung können dann XQuery- und SQL-Anfragen kombiniert werden.

Eine Vielzahl von Events basiert auf einer Korrelation von Metriken und deren Werte zur Laufzeit (z.B. Unterschreitung eines Schwellwertes zur Bufferpool Hit Ratio). Es kann aber auch Events geben, die nicht auf Metriken basieren, z.B. Log-Messages oder bestimmte System-Übergänge. Die Speicherung der Metrik-Daten kann, muss aber keineswegs zusammen mit den Event-Daten erfolgen. Wichtig ist die Möglichkeit zur integrierten Auswertbarkeit. Das bedeutet vor allem, dass man anhand des Datenbestands bestimmen kann, auf welchen Metriken bestimmte Events basieren bzw. welche anderen

(unmittelbaren) Metriken bzw. Ressourcen durch das Auslösen eines Events auch betroffen sind.

Jede Metrik lässt sich einer Ressource des zu verwaltenden Systems zuordnen. Diese Ressourcen können wiederum in Beziehungen (Sub-/Superkomponenten, qualitative und quantitative Abhängigkeiten) zueinander stehen. Unter solchen Beziehungen verstehen wir bspw. das Enthaltensein (also etwa die Dekomposition einer Metrik in additive Elemente), die gegenseitige Beeinflussung oder auch die Benutzung von Ressourcen (siehe Abschnitt 7.5). Um jene Performance-Daten eindeutig einer Ressource zuordnen zu können und die Abhängigkeiten zu erfassen, muss zusätzlich die Systemstruktur vorliegen. Vor allem aber um die Performance-Indikatoren auf verschiedenen Abstraktions-Ebenen betrachten zu können, ist die Abbildung **topologischer Strukturinformationen** des Systems im Datenmodell des PWH nötig.

Neben der Erfassung von Sensor-Daten, sind auch die Änderungen über die **Effektor-Schnittstellen**, wie etwa Konfigurationsparameter oder Adaptionen des physischen Designs, zu verzeichnen. Durch die Aufnahme dieser Änderungen in die (Performance-) Historie bietet sich die Möglichkeit etwaige Korrelationen zum Systemverhalten nachvollziehen zu können.

Darüber hinaus erscheint es im Sinne künftiger (Ursachen-)Analysen und eines stetigen Lernprozesses zweckmäßig, die Performance- und Struktur-Daten mit Beobachtungs- bzw. **Laufzeit-Daten des Tuning-Systems** zusammenzuführen, um somit eine integrierte Auswertbarkeit zu ermöglichen. Zu den Laufzeit-Daten des Tuning-Systems gehören bspw. die erkannten Probleme bzw. Events sowie daraufhin ausgeführte Problem-lösende Aktionen oder langfristige Analysen. Hierdurch ergibt sich die Möglichkeit zur Evaluation der Wirkung bzw. Effizienz, aber vor allem der (nicht ohne Weiteres im Voraus absehbaren) Nebenwirkungen von Tuning-Plan-Ausführungen.

Die Korrelation von Tuning-Maßnahmen und deren Auswirkungen erfordert, dass die bereits durch die Tuning-Pläne bzw. Effektoren vorgenommenen Änderungen protokolliert werden. Anhand der vorgenommenen Änderungen und dem Systemzustand (repräsentiert durch Kombination der Sensor-Werte des Systems zu einem bestimmten Zeitpunkt) kann die Effizienz der Änderungen eingeschätzt werden. Die Korrelationen von Tuning-Maßnahmen und Auswirkungen lassen sich allerdings nur schwer einschätzen, da eine Performance-Änderung nicht nur von den Änderungen der Effektoren abhängig sein muss, sondern auch weiteren Einflüssen, wie bspw. der Systemlast oder „naturbedingten" Verzögerungen, unterliegt.

Neben den zeitlichen und topologischen Aspekten, wollen wir für spätere Analysen ebenfalls Wissen über die **Workload** integrieren, gesondert speichern bzw. hervorheben und somit stets auffindbar zu machen. Hierunter fallen bspw. konkrete Transaktionen und SQL-Queries durch die Anwendungs-Programme bzw. Nutzer zur Datengewinnung oder auch im Rahmen des Tunings ausgeführte Optimierungs-Aktionen.

Es ist zudem vorstellbar, neben den aktuellen Ist-Daten auch die für einen bestimmten Zeitraum gültigen bzw. zu erreichenden Soll-(**Policy**-)Daten mit im PWH zu hinterlegen. Mit Hilfe dieser Informationen lassen sich Analysen über Abweichungen und etwaigen Degradationen der Performance sowie Ursachenfindung automatisieren und problemlos retrospektiv zu späteren Zeitpunkten durchführen.

6.3 Struktur des Performance Warehouse

Grundlage für die Datenhaltung ist eine geeignete Form der Persistierung aller relevanten Daten. Die bisherigen Ausführungen zur Vielzahl der potentiellen Daten-Quellen haben bereits zur Genüge die Notwendigkeit der Integration bzw. Vereinheitlichung benötigter Daten in einem zentralen Repository motiviert.

Zugunsten der Simplizität, Interoperabilität und vereinfachten (integrierten) Auswertbarkeit wollen wir asynchronen Event-Daten nach der Extraktion aus den zu überwachenden Systemen in das gängige **CBE-Format** (siehe Abschnitt 5.2) überführen. Durch das einheitliche Format kann ein einfacherer Umgang mit ihnen und eine unmittelbare Event-Korrelation bzw. Erkennung durch Tools gewährleistet werden. Im Anschluss sollen die Event-Daten zusammen mit den i.d.R. synchron erfassten Performance-Daten gespeichert werden.

Mit Kenntnis des Aufbaus des zu überwachenden Systems und seiner Ressourcen (Topologie), lässt sich eine homogene und vor allem **generische Speicherstruktur** für das PWH finden, die unabhängig von strukturellen Änderungen seitens der Quellen sämtliche Kontext- und Performance-Daten (Metriken und Events) beinhaltet.

Über den Zeit- und Kontext-Bezug können dann jederzeit Verbindungen zu den in eigenen Repositories abgelegten Daten über den Tuning-Verlauf hergestellt und damit vor allem Evaluierungs-Analysen durchgeführt werden (siehe Abschnitt 8.3 und 8.4).

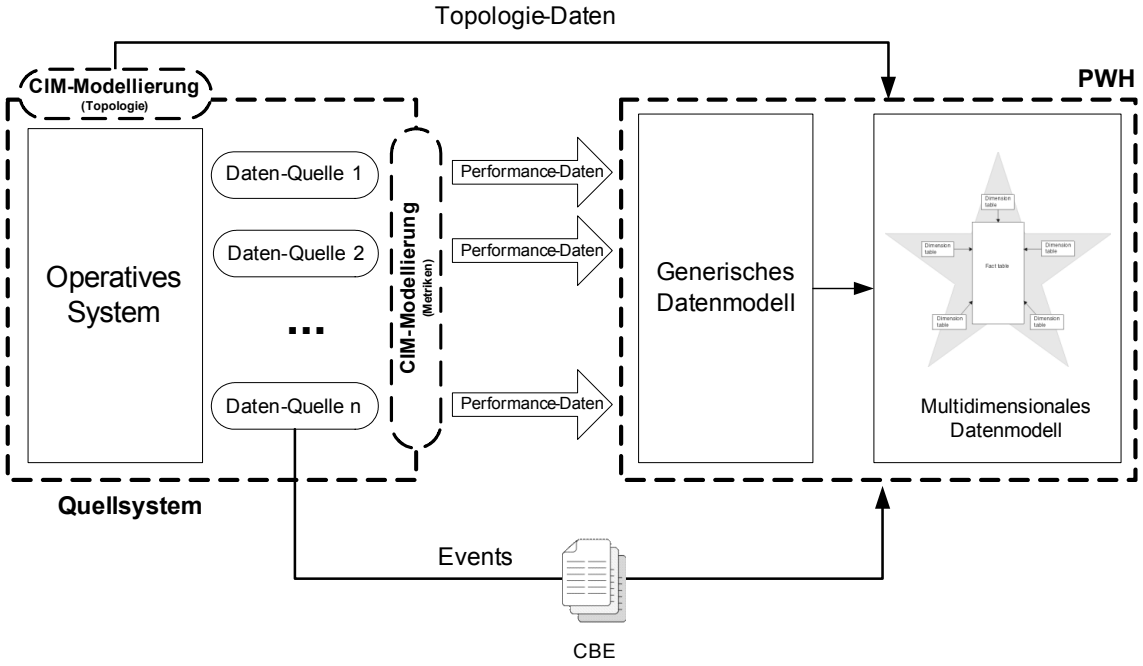


Abbildung 6.1: Potentielle Datenhaltungs-Ebenen im PWH-Umfeld

Die Datenhaltung kann prinzipiell in mehrere Betrachtungs- und Analyse-Ebenen unterteilt werden, die in der Übersicht in **Abbildung 6.1** dargestellt sind. Die unterste Ebene (links im Bild) stellt die verteilten, heterogenen, operationalen Daten(-Quellen) dar. Sowohl Inhalt als auch topologische Struktur können über ein CIM-Datenmodell (siehe Abschnitt 5.2.1 sowie 5.2.2) beschrieben und o.B.d.A. auf Quellseite vorausgesetzt werden. Eine vernünftige und integrierte Analyse dieser Daten ist erst durch eine Zusammenführung in ein globales, idealerweise änderungsresistentes, generisches Datenmodell im PWH (rechte Seite im Bild) möglich. Das generische Datenmodell definiert sich hauptsächlich durch die

topologische Struktur der Quellen und soll von Änderungen auf jener Seite unberührt bleiben. Um die DBAs bei der Problem- und vor allem der Ursachen-Findung noch weiter zu unterstützen und intuitive Analysen zu ermöglichen, bietet sich eine weitere, restrukturierte und den Inhalt aufbereitende, multidimensionale Ebene an. Aus Performance-Gründen können die Performance-Daten aus den Quellen auch direkt in eine generische, multidimensionale Struktur transformiert werden. Das generische, von Struktur-Änderungen auf Quellseite unabhängige, Datenmodell soll dabei als strukturelle und inhaltliche Grundlage für unsere Anwendungs- und Aufgaben-spezifischen, multidimensionalen Performance Cubes dienen.

Wir verstehen das PWH daher rein konzeptionell als ein Repository mit einem zentralen *generischen Ressourcen-Modell* (siehe Abschnitt 6.3.1.) zur Integration sämtlicher Performance- und Kontext-Daten sowie der Möglichkeit zur Erzeugung und Analyse von darauf aufbauenden multidimensionalen Datenstrukturen, wie bspw. *Performance Cubes* (siehe Abschnitt 6.3.2). Wir möchten an dieser Stelle keine Beschränkung einer möglichen Implementierung vorgeben, aber gezielt auf die potentielle Zweiteilung sowie die Möglichkeit eines einzelnen, rein multidimensionalen PWH-Modells hinweisen.

6.3.1 Das generische Datenmodell

Ziel ist es, die Daten an einer zentralen Stelle konsistent und langfristig abzuspeichern. Durch die Art und Struktur der Quellen, die Performance- bzw. Metadaten liefern, ergeben sich bereits Anforderungen an die Form der Speicherung und Strukturierung der Daten in dem zentralen Repository.

Da ein System im Laufe der Zeit immer dynamischen Veränderungen unterliegt, muss auch das Repository an diese angeglichen werden bzw. mit diesen umzugehen wissen. Sie beinhalten zum einen Änderungen im System-Aufbau, wie das Hinzufügen oder Entfernen von System-Elementen. Zum anderen zählt dazu aber auch ein effizienter Umgang mit großen Datenvolumina, welche über einen langen Zeitraum den Zustand verschiedenster Elemente widerspiegeln, die zudem in Beziehung zueinander stehen können.

Änderungen auf systemstruktureller Ebene sollen ganz einfach auf Änderungen in der Instantiierung im generischen Modell abgebildet werden können. Dazu bietet sich eine höchst generalisierte Struktur nach dem EAV-Modell [LiPl10] an. Es werden nur die grundsätzlichen allen Ressourcen gemeinsamen Eigenschaften (Attribute) abgebildet. Die individuellen Besonderheiten werden separat gespeichert und den entsprechenden Attributen bzw. Ressourcen zugeordnet.

In [Ell10] beschreiben wir ein im Rahmen der vorliegenden Arbeit entwickeltes, universelles, CIM-orientiertes Datenmodell für die Modellierung der Struktur und der Performance-Daten (Metriken, Events) eines Datenbank-basierten Software-Stack sowie Möglichkeiten der Translation (Überführung) aus anderen CIM-Ressourcen-Modellen. Es wird zudem deutlich gemacht, wie Performance-Metriken aus dem generischen CIM-Modell ausgelesen und in bereinigter und einheitlicher Form in eine multidimensionale Speicherstruktur eingefügt und wie CBEs mit der abgebildeten Systemstruktur verknüpft werden können. Es wird auch demonstriert, wie eigene Event-Beschreibungen auf den integrierten Metriken definiert, deren Auftreten automatisch detektiert und abgespeichert werden können.

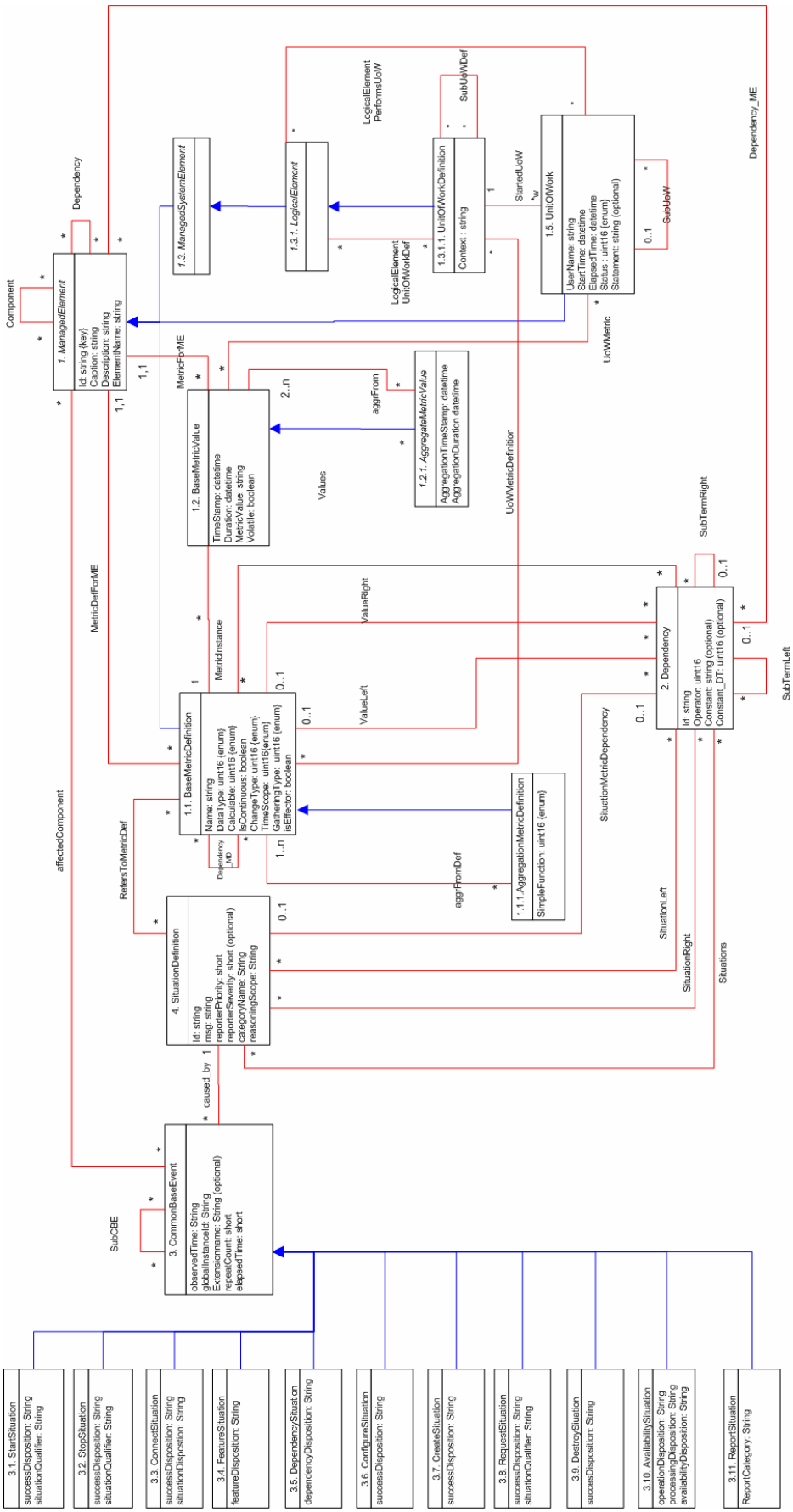


Abbildung 6.2: Ein generisches Performance- und Ressourcen-Datenmodell (nach [Ell10])

Durch das in **Abbildung 6.2** auf Basis von UML [36]visualisierte Modell, auf das im Folgenden in Kürze eingegangen wird, ist es möglich, sowohl die den Ressourcen zugehörigen Performance-Metriken, deren aufgetretenen Events, die Effektoren[37] sowie den Kontext in Form von Zeit, Topologie und Workload konsistent an einer zentralen Stelle für die spätere Auswertung zu hinterlegen. Dabei werden sowohl Quellen mit bestehenden CIM-Modellen als auch Quellen ohne eine standardisierte Schnittstelle bzw. ein gemeinsames Datenmodell unterstützt. Das Modell ist erweiterbar um spezifische, von, der eine beliebige Ressource im Software-Stack darstellenden Klasse, *ManagedSystemElement* abgeleitete Klassen, gemäß CIM-Standard. Das obige Modell beschreibt die Struktur der Ressourcen auf Typ-Ebene. Eine konkrete Instantiierung auf Objekt-Ebene wird nachfolgend exemplarisch aufgezeigt.

Aus etwaigen bestehenden CIM-Modellen werden nur ausgewählte strukturelle und inhaltliche Informationen übernommen, die mit dem generischen Ziel-Datenmodell konform sind und die Generizität unbeeinflusst lassen. Prinzipiell soll dem PWH dadurch eine Anbindung an beliebige CIM-Quellen gewährleistet werden. In [Ell10] zeigen wir daher eine Vorgehensweise für ein Mapping auf unser eigens definiertes, generisches CIM-Datenmodell. Sofern die Daten der Quellen über ein oder mehrere CIM-Modelle dargestellt und zugegriffen werden können, ist eine Automatisierung der Abbildung, der Transformation und des Ladevorgangs in ein semantisch gleichwertiges Modell möglich. In Abschnitt 5.2.6 haben wir bereits festgestellt, dass die Abbildung zwischen verschiedenen syntaktischen, aber auf der gleichen Semantik beruhenden Ausprägungen eines Ressourcen-Modells als relativ unkritisch angesehen wird.

In [Ell10] werden ausführlich die Voraussetzungen an bestehende CIM-Modelle des zu verwaltenden Systems, die Möglichkeiten des vorliegenden generischen Modells sowie die konkreten einzelnen Objekte, Attribute, Assoziationen und Aggregationen beschrieben. Daneben finden sich für die Laufzeitdaten im Speziellen auch Detaillierungen zu CIM-Metrics und Common Base Events, auf deren Grundlage dieses Modell erarbeitet wurde.

Für eine (semi-)automatische Abbildung beliebiger CIM-Modelle bzw. Teile davon in unser generisches Datenmodell sind vor allem Aspekte wie die Ressourcen, deren (Aggregations- bzw. Kompositions-)Hierarchien, die zugeordneten Metriken und Effektoren sowie die Abhängigkeiten und Assoziationen untereinander geeignet. Die meisten Attribute der in dem Modell dargestellten Klassen können demnach direkt aus einem bestehenden CIM-Modell übernommen werden. Andere Klassen müssen zunächst aus einem bestehenden CIM-Modell transformiert werden. In diesem Modell nicht vorhandene Attribute und Assoziationen der entsprechenden Objekte werden dabei ignoriert. Die Darstellung möchte keinen Anspruch auf Vollständigkeit erheben, da möglicherweise eigene Erweiterungen des CIM-Modells weitere Möglichkeiten zur Adaption bieten. Andere Klassen und Attribute wiederum müssen manuell definiert und hinzugefügt werden, falls sie Verwendung finden sollen. Für eine ausführliche Auflistung sei auf [Ell10] verwiesen.

CBEs sind nicht Teil des DMTF-Standard, können aber als zusätzliche Informationsquelle dienen und optional ohne weiteres an die CIM-Beschreibung angebunden und mit den zugrundeliegenden Metriken und Werten verknüpft werden. Zur Repräsentation der Common Base Events verwenden wir eine etwas modifizierte, komprimierte Version des

---

[36] Auf die Beschreibung der verwendeten UML-Notation soll an dieser Stelle kein Wert gelegt werden. Details können bspw. den Ressourcen unter http://www.uml.org/ entnommen werden.

[37] In unserem generischen Modell stellen Effektoren spezielle änderbare Metriken dar und werden gesondert gekennzeichnet.

ursprünglichen Modells. Die meisten Attribute können direkt aus der „Common Base Event"-Spezifikation extrahiert und übernommen werden. Aus ihrer Kombination wird, falls nicht schon vorhanden, eine Situations-Definition mit automatisch generierter ID geschaffen, die mit dem CBE-Konstrukt über eine entsprechende Assoziation verknüpft wird. Um ein CBE schließlich einer CIM-Komponente zuordnen zu können, müssen entsprechende Informationen über die Ressource im Rahmen des *extendedDataElement* inkludiert sein (siehe Abschnitt 5.2.4).

Nachdem die Daten in das neue Datenmodell überführt wurden, können weitere Situationen modelliert werden. Für diesen Zweck existiert die *Dependency*-Klasse. Die damit definierbaren Abhängigkeiten zwischen Metriken lassen sich mit entsprechend aufgetretenen Ereignissen assoziieren. Für konkrete Beispiel-Modellierungen sei wiederum auf unsere Ausführungen in [Ell10] referenziert. Die sich daraus ergebende rekursive Darstellung im Datenmodell findet sich in Abbildung 6.2 rund um die Klasse *Dependency*. Es lassen sich mit Objekten dieser Klasse Zusammenhänge von Metriken darstellen, die für bestimmte Situationen von Bedeutung sind. Eine assoziierte *SituationDefinition* (sprich das Zustandekommen eines Events) liegt genau dann vor, wenn die (Un-)Gleichung erfüllt ist. Somit ist das Datenmodell nicht ausschließlich auf die von Datenquellen gelieferten Events angewiesen, sondern erlaubt nach der Extraktion der Daten ebenso die Modellierung/ Definition neuer Situationen und die automatische Dokumentation ihres Auftretens. Ein umfangreiches Beispiel hierfür findet sich in [Ell10].

In [Ell10] legen wir anhand einer beispielhaften Überführung von Performance-Daten aus einem CIM-Modell in unser Repository die Tauglichkeit und Funktionalität des hier vorgestellten Konzepts dar. Die Beispielbeschreibung in Form eines Ausschnittes auf Instanz-Ebene des in Abbildung 6.2 visualisierten Datenmodells auf Typ-Ebene mit konkreten Werten ist in **Abbildung 6.3** dargestellt.

Zu sehen sind zwei konkrete Bufferpools. Auf *Bufferpool_B* wird eine Unit of Work ausgeführt, welche mit ihrer Definition assoziiert ist. Die instantiierten Klassen *Bufferpool_A* und *Bufferpool_B* besitzen dabei mehrere deskriptive Attribute, welche ihren Status (z.B. *operationalStatus* und *StatusDescriptions*) näher beschreiben, sowie zusätzlich damit assoziierte, quantitative Metriken inklusive deren Definitionen.

Alle drei hier dargestellten Metrik-Werte stellen Bufferpool Hit Ratios zu bestimmten Zeitpunkten dar. Dies bedeutet insbesondere, dass über jenes dargestellte Zeitintervall Messungen erhoben und Werte ermittelt wurden. Die in diesem Beispiel dargestellten Werte müssen nicht real vorkommenden Werten entsprechen. Die Werte gehören zu dem System-Objekt, welches jeweils über die Assoziations-Instanz von *DB2_MetricValueForBPHR* verbunden ist. Da die Attribute größtenteils selbsterklärend sind, soll hier nur kurz auf die Unterschiede und deren Bedeutung der verschiedenen Metrik-Werte eingegangen werden:

- BPHR_A_1
  - Gültigkeit: von 2007-03-07 15:13:43 bis 2007-03-07 15:13:64
  - Wert: 60 %
  - zugehöriges Systemelement: DB2_Bufferpool_A
- BPHR_A_2
  - gültig von 2007-03-07 15:13:00 bis 2007-03-07 15:13:43
  - Wert: 30 %
  - zugehöriges Systemelement: DB2_Bufferpool_A

- BPHR_B_1
    - gültig von 2007-03-07 15:13:39 bis 2007-03-07 15:13:54
    - Wert: 20 %
    - zugehöriges Systemelement: DB2_Bufferpool_B

Wie ersichtlich ist, steigt die BPHR von Bufferpool_A um 15:13:43 Uhr um 30% auf 60%. Innerhalb des Zeitintervalls von [15:13:39 Uhr, 15:13:54 Uhr] am 07.03.2007 liegt der durchschnittliche Wert über beide hier dargestellten Bufferpools bei 25%.

In diesem Beispiel ist ebenfalls eine Instanz eines Common Base Events vorhanden. Sie besteht fast nur aus laut Spezifikation notwendigen Elementen und wurde am 07.03.2007 um 15:13:00 Uhr ausgelöst, sprich zu Beginn der Gültigkeit des Metrik-Wertes BPHR_B. Der Attributwert der Instanz von *SituationData* enthält eine Warnung über eine niedrige BPHR. Weiterhin ist innerhalb des *extendedDataElements* als Wert des Attributes *Values* die Identifikation von *DB2_Bufferpool_A* angegeben. Die über die Assoziation *affectedComponent* verknüpften Elemente bieten Informationen über die „Adresse" der betroffenen Komponente.

Abbildung 6.4 stellt die Überführung der Objekte des Quellschemas auf die Objektebene unseres generischen Datenmodells dar. Schwarz Eingefärbtes kann direkt und grün Eingefärbtes indirekt, im Anschluss an automatische Anpassungen, übernommen werden. Alles rot Gefärbte kann nur durch manuellen Eingriff eingetragen werden, da dem System im Allgemeinen die nötigen Informationen für ein Setzen der Werte nicht vorliegen.

Zusätzlich zum Überführen der Daten in das neue Datenmodell können weitere, originär nicht in Form von CBEs vorliegende Situationen modelliert und später erkannt werden. Hierfür kann auf das Dependency-Objekt zurückgegriffen werden, um derartige Abhängigkeiten zwischen bestimmten Metriken darzustellen. Im konkreten Fall soll die in *SituationDefinition_B* definierte Situation vorliegen, sobald ((((Metrik-Wert von *BPHR-Definition_A*) + (Metrik-Wert von *BPHR-Definition_B*)) / 2) <= (50)). Diese Situation tritt in diesem Beispiel in dem Zeitraum von 2007-03-07 15:13:39 bis 2007-03-07 15:13:43 auf. Aus diesem Grund ist ein CBE entstanden. Die Erkennung kann automatisiert erfolgen, sobald Situation und Abhängigkeiten der Metriken erst einmal definiert sind.

An dieser Stelle soll nicht näher auf die Besonderheiten der Modellierung der Quell- und Ziel-Datenmodelle sowie mögliche automatisierte Überführungen eingegangen werden. Weitergehende Konzipierungs- und Modellierungs-Details sind unseren in [Ell10] aufgeführten Schilderungen zu entnehmen.

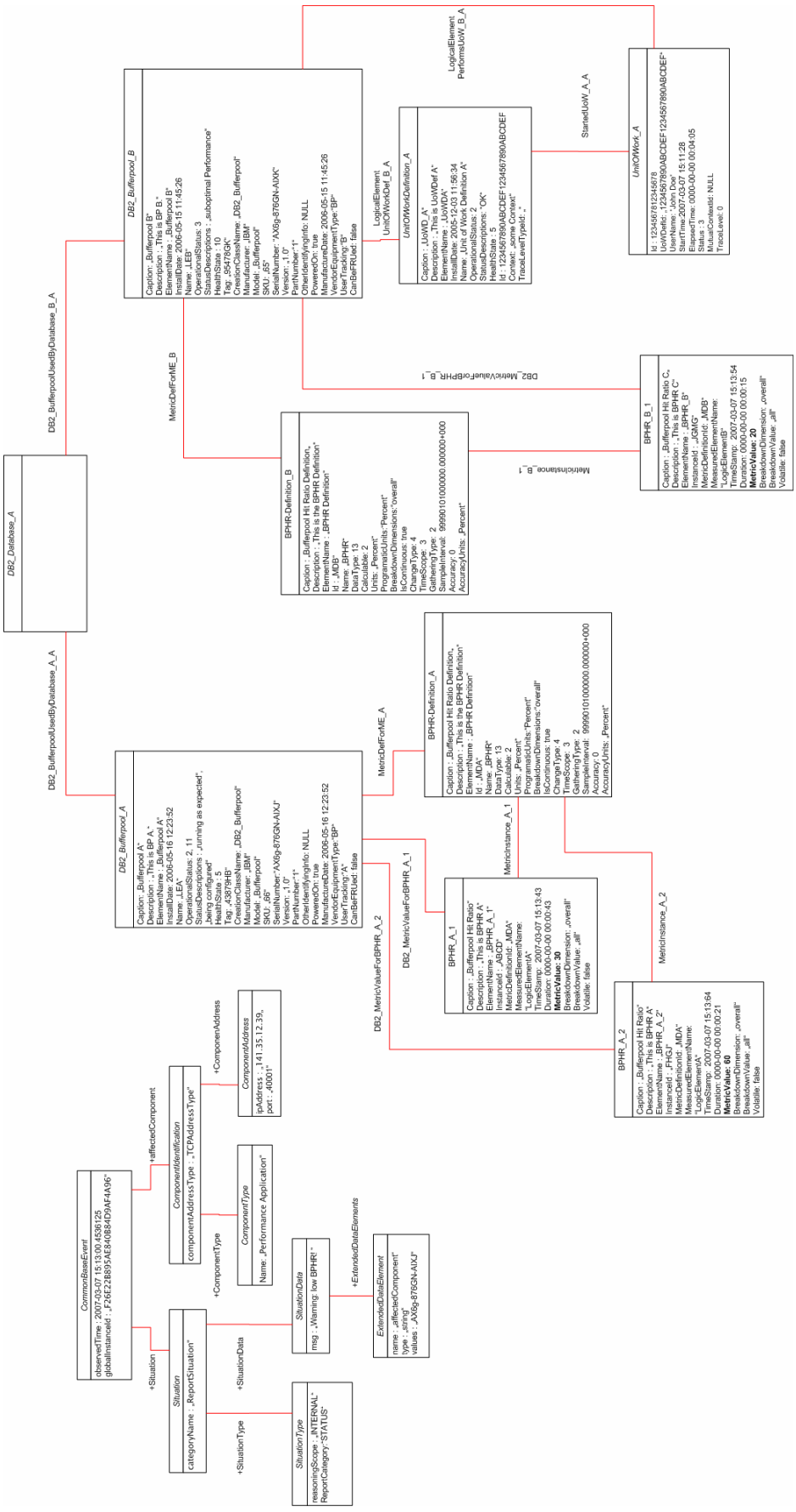


Abbildung 6.3: Eine Quellen-Beschreibung auf CIM-Instanzen-Ebene (nach [Ell10])

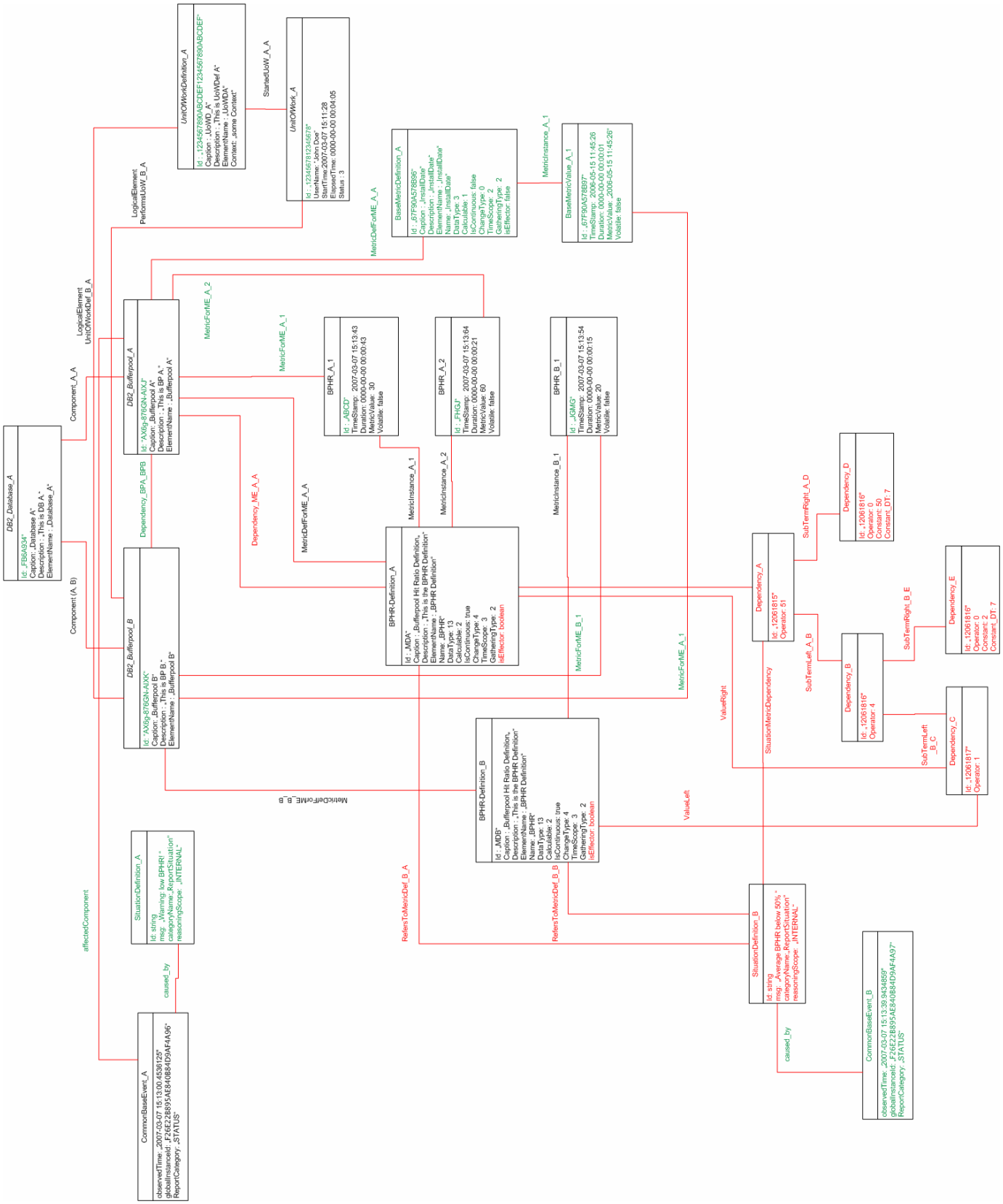


Abbildung 6.4: Abbildung eines CIM-Modells in unser generisches Datenmodell (nach [Ell10])

6.3.2 Das multidimensionale Datenmodell (Performance Cubes)

Gängige Datenstrukturen traditioneller Monitoring-Tools konfrontieren bzw. überfordern den Nutzer bei der Analyse mit einer Fülle von Daten und nicht erkennbaren Zusammenhängen. Der ineffektive Umgang mit den großen Daten-Mengen und den Korrelationen zwischen Topologie- und Performance-Daten lässt den menschlichen Analyse- und Entscheidungs-Prozess unnötig verlangsamen. Mit herkömmlichem SQL geraten die Nutzer sehr schnell an die Grenzen des machbaren. Das Herausfiltern relevanter Fakten stellt selbst für erfahrene Administratoren eine Herausforderung dar und wird zunehmend zeitaufwendig und fehleranfällig. Die flachen Performance-Reports sind zumeist starr, inflexibel und in Struktur stark von der ursprünglichen Fragestellung geprägt. Es gestaltet sich als schwierig, zeitaufwendig bis gar unmöglich, die Performance-Daten aus anderen Blickwinkeln zu betrachten.

Im PWH sollen u.a. Performance-Daten für die kurzfristige und die Langzeit-Analyse gespeichert werden. Im Hinblick auf die Entscheidungs-Unterstützung ist die grundlegende Idee daher die Nutzung einer OLAP-Komponente zur fortgeschrittenen, intuitiven Analyse dieser Daten durch die Administratoren [Wie05]. Zu diesem Zweck müssen die erfassten Performance- sowie Architektur- und Topologie-Daten zunächst aus ihrer recht inflexiblen Form in eine andere, Analyse-orientierte und idealerweise multidimensionale Struktur mit einer sinnvollen Dimensionierung und Darstellung von Daten auf verschiedenen Abstraktions-Ebenen überführt werden. Durch eine Aufteilung des Datenraums in für Nutzer verständliche Betrachtungsdimensionen und der flexiblen Erhöhung/Senkung des Detailgrades lassen sich gezielt Art und Umfang der zu betrachtenden Daten regulieren. Die multidimensional aufbereiteten Daten können zudem als Input für eine Reihe von Data-Mining-Algorithmen zur Trend-Analyse, -Vorhersage und sogar -Vermeidung dienen (siehe Abschnitt 7.3 - 7.5).

Wie wir bereits einführend angedeutet haben, ist eine derartige Konsolidierung und Restrukturierung des Inhalts *anstatt* des generischen Modells oder auch auf diesem *aufbauend* denkbar. Eine der ersten Entscheidungen, die im letzteren und naheliegenderen Fall für das Design derartiger **Performance Data Marts** (oder auch Performance Cubes) getroffen werden muss, ist die Auswahl der Ziel-Datenbank und die Struktur, in der die Performance-Daten sowie der Kontext gespeichert werden sollen. Traditionell verwendet man zur Persistierung und Visualisierung von (Performance-)Daten u.a. Spreadsheets/ Dashboards, Dateien, XML-Dokumente oder auch relationale Datenbanken [Wie05]. Der Idee des DWH-Gedanken folgend und im Hinblick auf die Analyse-Orientierung sowie die Fähigkeit zur integrierten Auswertbarkeit von Performance-Daten aus verschiedensten Quellen, erscheint eine relationale Datenbank-Lösung (Stichwort ROLAP, siehe Abschnitt 2.3) am ehesten geeignet.

Zu dem Thema sind bereits zahlreiche Vorarbeiten erfolgt. Die in diesem Abschnitt aufgeführten Detaillierungen basieren größtenteils auf den Veröffentlichungen [Wie05, Wie06, Wie09] in diesem Kontext. In [Ell10] umreißen wir zudem in Kürze, wie mit Hilfe der strukturellen und semantischen Kontext-Informationen des generischen Datenmodells eine automatische Ableitung multidimensionaler Konstrukte erfolgen kann.

6.3.2.1 Terminologie

Die Terminologie unseres in **Abbildung 6.5** schematisch skizzierten, multidimensionalen Meta-Modells gleicht im Wesentlichen den Definitionen des Abschnitts 2.3.3. Dennoch erscheinen uns einige der feinen Unterschiede erwähnenswert. Im Folgenden stellen wir daher die wichtigen in [Wie05] ausführlich beschriebenen Konzepte zusammenfassend dar.

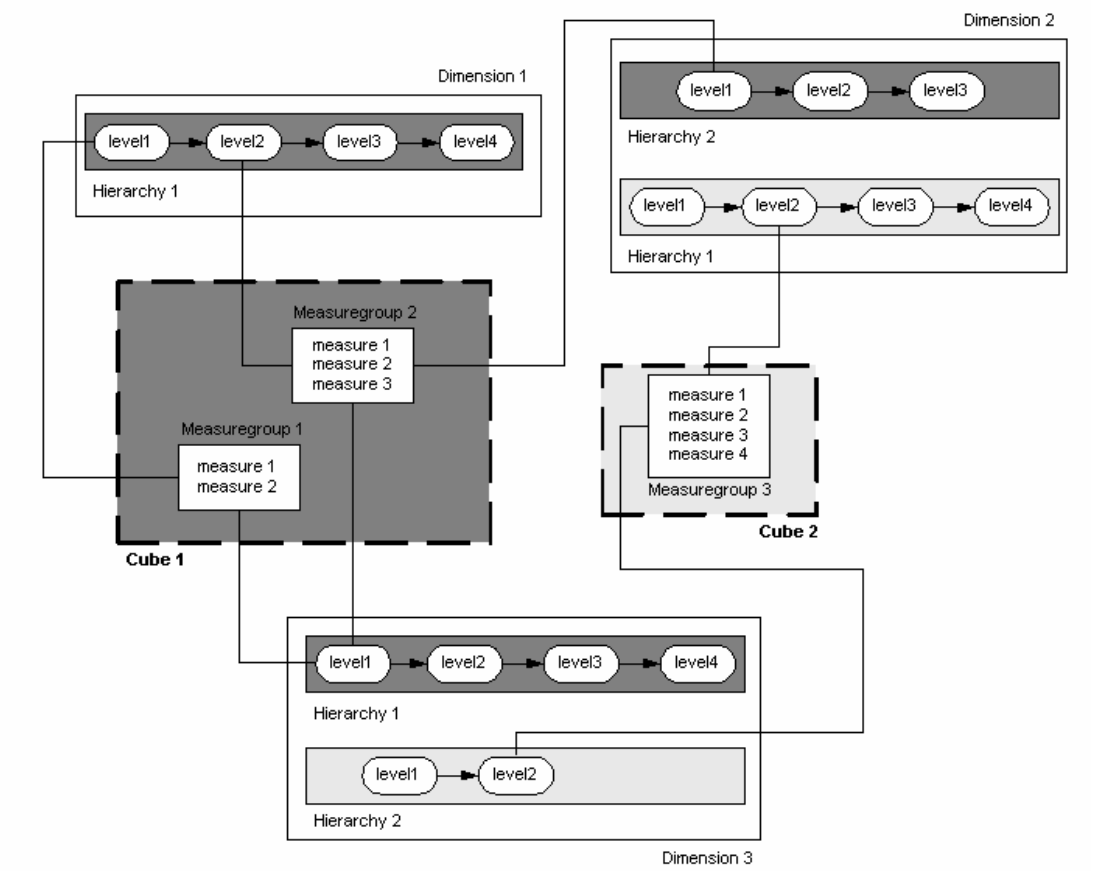


Abbildung 6.5: Ein multidimensionales Meta-Modell (nach [Wie05])

Ein multidimensionaler **Performance (Hyper)Cube** repräsentiert einen spezifischen, an individuelle Analyse-Bedürfnisse angepassten, Ausschnitt von Performance-Daten über einen bestimmten Zeitraum. Jede Instanz eines derartigen Cube ist ein abstraktes Abbild eines logischen Star oder Snowflake Schema (siehe Abschnitt 2.2.3) und gruppiert relevante, charakterisierende **Dimensionen** bzw. konkrete **Hierarchien** um eine oder mehrere zentrale Gruppierungen von Performance-relevanten Fakten (Measures), den sogenannten **Measuregroups**. Für eine weitergehende semantische Strukturierung, etwa nach Ressourcen, Problemfall oder Anwendungs-Domäne können Cubes auch in **Cube Schemata** eingeteilt werden.

Jede Dimension beschreibt mit ihren (Dimensions-)Attributen einen Aspekt einer Measuregroup bzw. eines Measures. Eine Dimension kann aus mehreren parallelen **Hierarchien** bestehen, die eine Möglichkeit darstellen, sich navigierend und explorativ durch die Dimension zu bewegen und die Daten auf unterschiedlichen Abstraktions-Ebenen zusammenzufassen. Jedem Cube wird der Einfachheit und Eindeutigkeit halber zu einer Zeit nur eine von mehreren parallel möglichen Hierarchien zugeordnet. Ein Cube setzt sich demzufolge aus einer Menge von mindestens 2 disjunkten Hierarchien zusammen. Die Dimensionen dienen dabei lediglich als eine Art Container zur Gruppierung von zusammengehörigen Hierarchien.

Eine Hierarchie besteht aus verschiedenen **Ebenen** (Levels) und den pro Ebene existierenden (Hierarchy-Level-)Attributen, welche Informationen darüber beinhalten, wie die Ebenen in der Hierarchie strukturiert sind und miteinander in Beziehung stehen. Ein Level kann in vielen verschiedenen Dimensionen (wieder-)verwendet werden.

Mathematisch kann die Beziehung der Ebenen (L) einer Hierarchie als partielle Ordnung (Halbordnung) über L abgebildet werden, so dass $L_1 < L_i < L_n$ für alle $1 \leq i \leq n\text{-}1$. Die Halbordnung „<“ definiert hierarchische *Roll-Up-Pfade*, welche die Ableitung (Aggregation) von Werten für Measures auf einer Ebene in der Hierarchie von der darunter liegenden erlauben.

Abhängigkeiten zwischen den Attributen existieren für gewöhnlich anhand funktionaler Abhängigkeiten. Hieraus lassen sich zwei Klassen von Hierarchy-Level-Attributen unterscheiden. Zum einen identifizieren die Werte primärer Attribute eindeutig die verschiedenen Instanzen einer Ebene. Zum anderen liefern deskriptive und funktional von den primären abhängige Attribute, zusätzliche Informationen über diese Instanzen.

Die **Measuregroups** stehen quasi im Zentrum der Cubes und gruppieren eine Menge von Performance-Kennzahlen (Fakten/Measures) semantisch nach Themenbereich. Die Struktur ist ähnlich der eines Array. Wie die Dimensionen eines Array, dient die Kombination der Measure(group)-Dimensionen als Index zur Identifizierung individueller Zellen und damit auch der Bestimmung der Situation bzw. des Kontext dieser Maßzahlen. Zur Ermittlung eines konkreten Wertes in diesem „Array“ muss demnach ein Wert für jede der Dimensionen spezifiziert werden. Bspw. ist die Aussage über eine BPHR von 13% ohne Zusatzinformation relativ nutzlos. Angereichert mit (semantischen, Kontext-spezifischen) Angaben über die Zeit und den Namen bzw. den hierarchischen Namens-Pfad des Bufferpools wird die Bedeutung und Aussagekraft deutlich.

Jede Measuregroup ist mit einer **Measuregroup Association List** (MGAL) versehen, die spezifiziert, mit welchen konkreten Hierarchie-Ebenen welcher Dimensionen die Maßzahlen dieser Gruppe verknüpft sind. Eine mit einer bestimmten Ebene verknüpfte Measuregroup gehört automatisch zu allen über die Roll-Up-Hierarchie definierten Vater-Ebenen dieser Hierarchie, sofern die enthaltenen Measures aggregier- bzw. berechenbar sind. Mit allen darunter liegenden Ebenen wird jedoch keine Beziehung eingegangen.

Darüber hinaus kann es auch leere Measuregroups geben, die semantische Informationen über den Kontext bzw. die Topologie der Quelldaten ausdrücken, bspw. Informationen darüber welche Datenbank durch welchen Nutzer bzw. Applikation zu welcher Zeit am häufigsten zugegriffen wurde, welche Operationen zu welchem Statement gehören oder auch welche Operationen auf welchen Ressourcen aktiv waren.

Für spezifische Besonderheiten und Einschränkungen des vorgestellten Modells und seiner Konzepte sei gesondert auf [Wie05] ver**wiese**n.

6.3.2.2 Relationale Repräsentation

Zur Speicherung der Performance-Daten in DWH-orientierten, multidimensionalen Strukturen eignen sich aus unserer Sicht am ehesten relationale Datenbank-Management-Systeme. Alle großen relationalen DBMS unterstützen mittlerweile komplexe OLAP-Anfragen und -Analyse-Funktionen sowie spezielle multidimensionale (Zusatz-)Strukturen auf der physischen Ebene. Sie können demnach sowohl mit operationalen als auch mit Data-Warehouse-Daten bzw. Anwendungen umgehen. Ein weiterer Vorteil der dieser Systeme ist, dass sie mit standardisierten Datenbank-Anfragen (SQL-Schnittstellen) arbeiten.

In unserer Arbeit zum Thema [Wie05] beschreiben wir im Detail die Umsetzung der aufgeführten Konstrukte des multidimensionalen (Meta-)Modells von der Konzipierung, über die E/R-Modellierung, bis hin zur Speicherung auf relationaler Ebene. **Abbildung 6.6** zeigt die dabei entstandene relationale Repräsentation. Sowohl Entity-Typen als auch n:m-Beziehungstypen werden in der darauf basierenden relationalen Abbildung durch eigene

Tabellen repräsentiert. Die allen Entity-Typen gemeinen Audit-Informationen dienen der Verfolgung des Ursprungs von Änderungen an den Instanzen. Alle anderen in Form von Tabellen dargestellten Entity- und Beziehungs-Typen ergeben sich naturgemäß aus Abbildung 6.5 und den obigen Beschreibungen.

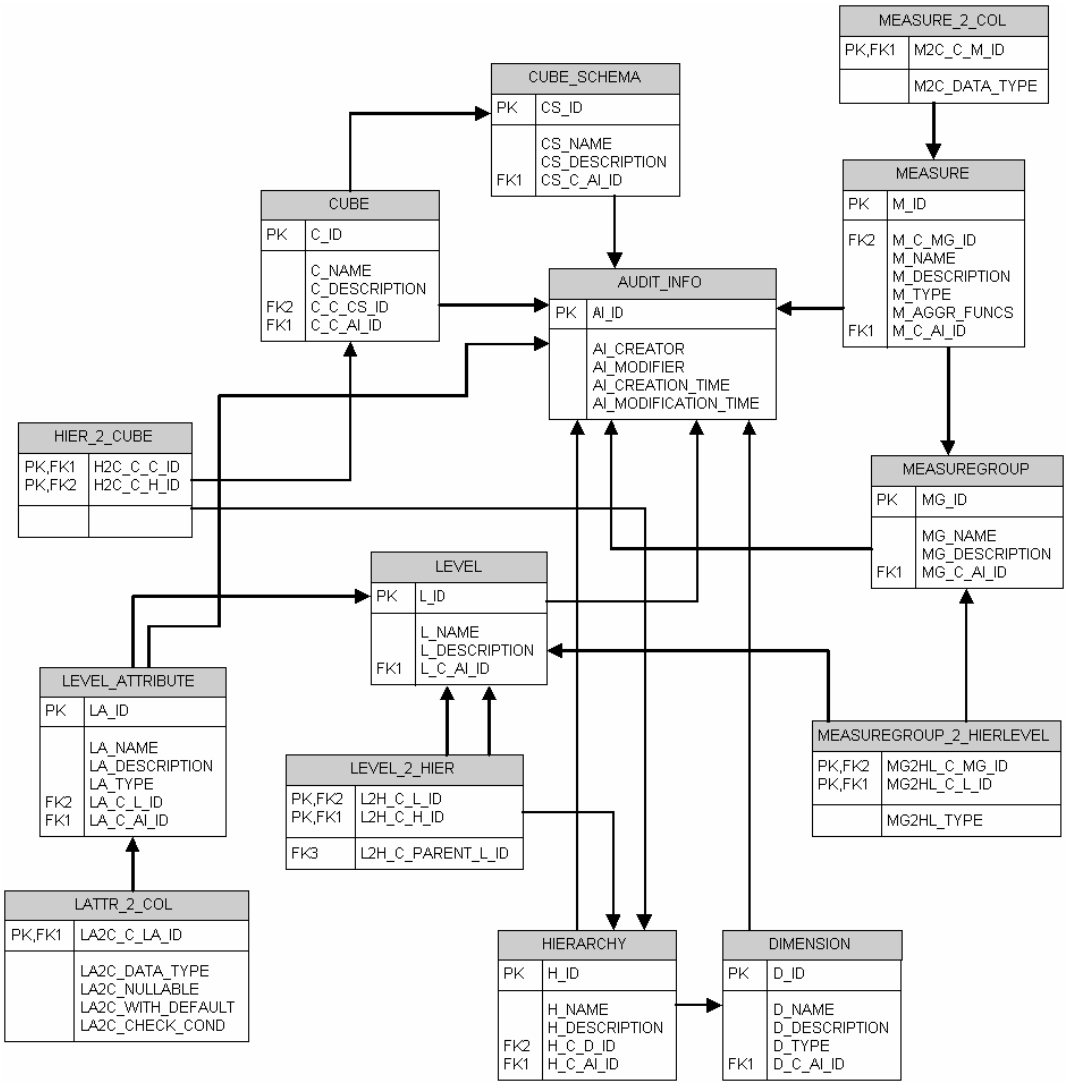


Abbildung 6.6: Relationale Repräsentation des multidimensionalen Meta-Modells [Wie05]

Die Instanzen dieses Schemas stellen letztlich konkrete Anwendungs-spezifische multidimensionale (Star-Schema-)Modelle dar. Diese Metadaten können einerseits dazu genutzt werden, um automatisch die verkörperten relationalen Star-Schemata zu erzeugen, andererseits aber auch um Administratoren bzw. Analyse-Tools bei der Navigation und der Erfragung von strukturellen Hintergrund-Informationen über die verfügbaren Daten und deren Verknüpfungen zu unterstützen. Eine ausführliche Beschreibung zur Umsetzung, Namensgebung und den Besonderheiten lässt sich unter [Wie05] nachschlagen.

Idealerweise werden die Administratoren bei der Instantiierung der Tabellen und der damit möglichen Cubes durch ein graphisches Interface bzw. Tools zur Wartung der multidimensionalen Strukturen unterstützt. Dennoch wird es mit wachsender Zahl der Dimensionen und damit auch der Komplexität erheblich schwerer die konkreten

Ausprägungen und das zugrundeliegende Modell für Menschen zu veranschaulichen. Daher sollte die Zahl der Dimensionen überschaubar und klar angeordnet sein.

6.3.2.3 Exemplarische multidimensionale Strukturen für die Performance-Analyse

Die Daten der Performance Data Marts im PWH sind in einem klassischen Star Schema mit zentralen Fakten, die hinsichtlich mehrerer Dimensionen ausgewertet werden können, organisiert. Dieses multidimensionale Datenmodell unterstützt die flexible und schnelle Ausführung detaillierter und aggregierter Anfragen sowie vielfältige interaktive Analysen.

Im Folgenden wollen wir einige der bereits angedeuteten Informationen, welche innerhalb eines multidimensionalen PWH als Kontext-beschreibende Dimensionen und Performance-charakterisierende Fakten dargestellt werden können bzw. sollten, zusammenfassend auflisten:

- Fakten
  - Sensor-Werte: Darunter fallen die über die Sensor-Schnittstellen gesammelten Metriken der entsprechenden Ressourcen.
  - Effektor-Werte: Sollten im Rahmen des Tuning Konfigurationsparameter des zu tunenden Systems (oder auch andere Effektor-Elemente) geändert werden, so muss diese Änderung in die Historie mit aufgenommen werden, um mögliche Korrelationen zum Systemverhalten nachvollziehen zu können.
  - Events: Die erkannten System-Ereignisse (inkl. deren Typen) sollten zusätzlich gespeichert werden, um eine spätere (korrelierte) Analyse einfach gewährleisten zu können.
  - Ausführungspläne: Jede Anfrage wird vom Optimizer nach einer bestimmten Operations-Abfolge ausgeführt. Der Anfrage-Ausführungsplan (query execution plan, QEP) ist abhängig von dem Zustand der Datenbank und ihrer Ressourcen. Es kann daher nützlich sein, den Systemzustand mit dem vom Optimizer gewählten QEP zu korrelieren und zu ermitteln welche konkreten Operationen mit welchen Wartezeiten ausgeführt wurden.
- Dimensionen
  - System-Struktur: Jede Metrik lässt sich einer Ressource des zu verwaltenden Systems zuordnen. Diese Elemente stehen wiederum in Beziehung zueinander (z.B. Sub-/Superkomponente). Um die Fakten auf verschiedenen Abstraktions-Ebenen betrachten und eindeutig einem System-Element zuordnen zu können, muss die topologische System-Struktur als Dimension vorliegen.
  - Zeit: Jeder Wert einer Metrik bzw. eines Effektors ist lediglich zu einem bestimmten Zeitpunkt bzw. Zeitraum gültig. Um die verschiedenen Metriken über bestimmte Zeiträume auf einfache Weise protokollieren und später analysieren zu können, muss die Zeit als Dimension vorliegen.
  - Workload: Es sollte zudem protokolliert werden, welche (SQL-)Aktionen sowie Statement-Operationen das System bzw. die Nutzer durch ihre Applikationen zu den entsprechenden Zeitpunkten der Metrik-Sammlung ausgeführt haben.

Im Rückblick auf das generische Datenmodell (Abschnitt 6.3.1) und basierend auf den Analysen in [Ell10] können demnach die Klassen *BaseMetricValue*, *BaseMetricDefinition*,

CommonBaseEvent und *SituationDefinition* als Fakten aufgefasst werden. Die Dimensionen bilden in dem Fall die Klasse *ManagedSystemElement* (inkl. *UnitOfWork* und *UnitOfWork-Definition*) und auch die Zeit, zu welcher die Werte gültig sind. Das *ManagedElement* taucht in dieser Auflistung nicht auf, da es sich um eine abstrakte Klasse handelt. Die *Dependency*-Klasse sollte dabei innerhalb der Hierarchien in den Dimensionen Eingang finden. Es ist jedoch auch vorstellbar, die Abhängigkeiten separiert von der multidimensionalen Struktur, aber innerhalb des gleichen Repository, zu hinterlegen. Anhand der Beziehungen zwischen den System-Elementen kann ein automatischer Roll-Up bzw. Drill-Down stattfinden. Über- bzw. untergeordnete Elemente lassen sich hierbei an der Assoziation *Component* identifizieren.

Zu den Fakten bzw. Measures der Cubes ließen sich auch die **Ausführungsdaten über die Tuning-Pläne** hinzuzählen und unter Verwendung einer Zeit- sowie einer Tuning-Plan-Dimension eine Historie verwirklichen. Darüber hinaus ist eine Verknüpfung mit den durch die Tuning-Pläne ausgelesenen bzw. (indirekt) angefassten und veränderten Effektor- bzw. Sensor-Elementen vorstellbar. Wie wir noch in Abschnitt 8.3 aufgreifen werden, erscheint aber auch eine nicht-multidimensionale Speicherung jener Laufzeitdaten und eine übergreifende Verknüpfung zur Laufzeit als nicht abwegig und wird von uns bevorzugt.

Die DB2-Architektur beinhaltet bereits eine umfassende Menge von Objekt-Hierarchien, die in **Abbildung 6.7** an einem Ausschnitt verdeutlicht werden und in Form in Kürze aufgezeigten Dimensions-Hierarchien Verwendung finden können. Die dargestellten Beziehungen repräsentieren im Wesentlichen das Enthaltensein und werden in Form von Kanten zwischen den mittels Rechtecken dargestellten DB2-Objekten und den entsprechenden (n,m)-Kardinalitäten visualisiert.

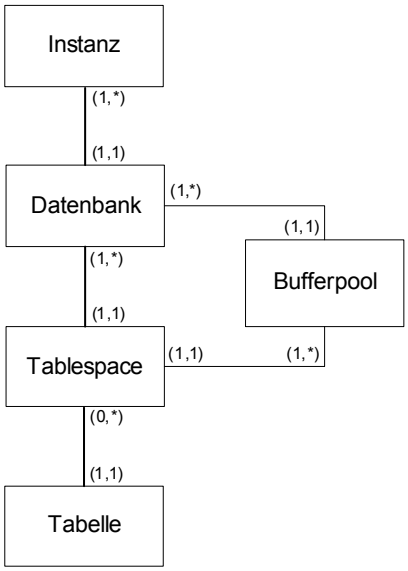


Abbildung 6.7: DB2-Objekt-Hierarchien

Tabelle 6.1 präsentiert einige konkrete und in unserem Szenario anwendbare Dimensionen sowie parallele Hierarchien. Die Hierarchie-Pfade sind von links nach rechts gesehen, von Kind in Richtung Vater (Roll-Up) dargestellt und beinhalten DB2-Ressourcen[38]. Abhängig von dem Ausgangsschema der Quellen und den Organisationsformen können auch

[38] Hierbei stehen *DBPG* für *Database Partition Group* und *TA* für *Transaction*.

Dimensionen wie jene in den letzten 3 Zeilen der Tabelle von Nutzen sein. Weitere Dimensionen und die umfassende Auflistung der DML-Statements zur Bereitstellung bzw. Füllung des multidimensionalen Meta-Modells finden sich in [Wie05, Wie09].

Dimension	Name der Hierarchie	Hierarchie-Pfade
DB_OBJECTS	DB_OBJECTS_LOGICAL	TABLE - SCHEMA - DB - INSTANCE
	DB_OBJECTS_PHYSICAL	TABLE - TBS - BP - DB - INSTANCE
	DB_OBJECTS_LOGICAL_EEE	TABLE - SCHEMA - DBPG - DB - INSTANCE
	DB_OBJECTS_PHYSICAL_EEE	TABLE - TBS - BP - DBPG - DB - INSTANCE
TIME	TIME	MICROSEC - SEC - MIN - HOUR - DAYTIME -
		DAY - MONTH - YEAR
		DAY - MONTH - QUARTER - YEAR
		DAY - WEEK - YEAR
WORKLOAD	WORKLOAD_NORMAL	STMT - APPL
	WORKLOAD_DETAIL	STMT_OP - STMT - TA - CONNECTION - APPL
USER	USER	USER - OS_GROUP
PARTITION	PARTITION	PARTITION
ORGANIZATIONAL_STRUCTURE		USER - DEPARTMENT
GEOGRAPHICAL_STRUCTURE		USER - DEPARTMENT - CITY - REGION - COUNTRY
HARDWARE_STRUCTURE		CONTAINER - FILE/DIRECTORY - FILE SYSTEM
		DISK - DISK_ARRAY
		CPU - NODE - CLUSTER

Tabelle 6.1: Exemplarische Dimensionen und Hierarchie-Pfade

In **Tabelle 6.2** sind einige der implementierten auf Performance-Metriken basierenden Measuregroups und Measuregroup Association Lists (MGAL) zusammengefasst. Die Klammern in der MGAL-Spalte sind ein Indikator für optionale Hierarchien. Das heißt auch, dass sich eine Measuregroup aus allen (nicht geklammerten) Hierarchien zusammensetzen und von der Granularität auf dem spezifizierten Hierarchy-Level beginnen muss. Die Liste ist beliebig erweiterbar und sofern die Quellen per CIM beschrieben werden können, im Grunde auch automatisch generierbar.

Measuregroup	Measuregroup Association List (MGAL)
BP_IO_INFO[1]	MICROSEC (x TBS) (x STMT_OP x USER) (x PARTITION)
STMT_COUNT	(APPL x USER x) MICROSEC (x DB) (x PARTITION)
APPL_STATUS	APPL x DB x MICROSEC
STMT_INFO	(DB x) MICROSEC x STMT
STMT_OP_INFO	(DB x) MICROSEC x STMT_OP
UOW_INFO	MICROSEC x TA (x DB)
TABLE_ACTIVITY	TABLE x MICROSEC (x PARTITION)
INTERNAL_COUNTS	(STMT_OP x USER x) MICROSEC (x DB) (x PARTITION)
ROW_COUNTS	(STMT_OP x USER x) (TABLE x) MICROSEC (x PARTITION)
AGENTS_N_CONNS	INSTANCE x MICROSEC (x PARTITION)
LOCKS_N_DEADLOCKS	DB x MICROSEC (x PARTITION)
INFORMATIONAL_DBCFG	DB x MICROSEC (x PARTITION)
MODIFIABLE_DBCFG	DB x MICROSEC (x PARTITION)
DBMCFG	INSTANCE x MICROSEC

Tabelle 6.2: Exemplarische Measuregroups

Die identifizierten Dimensionen und Measuregroups können auf unterschiedliche Weise kombiniert werden und ergeben u.a. die in **Tabelle 6.3** dargestellten Cubes.

Cube	Cube Dimension Hierarchies	Measuregroups (MG(CubeNr))
1	TIME x WORKLOAD_DETAIL	BP_IO_INFO, INTERNAL_COUNTS, UOW_INFO
2	TIME x DBOBJECTS_LOGICAL	BP_IO_INFO, STMT_COUNT, TABLE_ACTIVITY, INTERNAL_COUNTS, ROW_COUNTS, AGENTS, LOCKS_N_DEADLOCKS, INFORMATIONAL_DBCFG, MODIFIABLE_DBCFG, DBM_CFG
3	TIME x DBASE x USER x WORKLOAD_DETAIL	BP_IO_INFO, STMT_COUNT, APPL_STATUS, INTERNAL_COUNTS, STMT_INFO, STMT_OP_INFO, UOW_INFO
4	TIME x DBOBJECTS_PHYSICAL	MG(2)
5	TIME x DBASE x WORKLOAD_NORMAL	STMT_COUNT, APPL_STATUS
6	TIME x DBOBJECTS_LOGICAL_EEE x PARTITION	MG(1) \ {DBM_CFG}
7	TIME x DBOBJECTS_PHYSICAL_EEE x PARTITION	MG(1) \ {DBM_CFG}

Tabelle 6.3: Exemplarische Cubes mit zugehörigen Hierarchien und Measuregroups

Abbildung 6.8 veranschaulicht vier der genannten Measuregroups (in zwei Cubes) zusammen mit deren assoziierten Dimensionen. Die schematische relationale Struktur eines resultierenden Cubes entspricht der in Abbildung 2.12 dargestellten. Die künstlichen Schlüssel der im Hinblick auf die Performance denormalisierten Dimensions-Tabellen dienen in der zentralen Faktentabelle als jeweilige Fremdschlüssel und bilden einen gemeinsamen, zusammengesetzten Primärschlüssel.

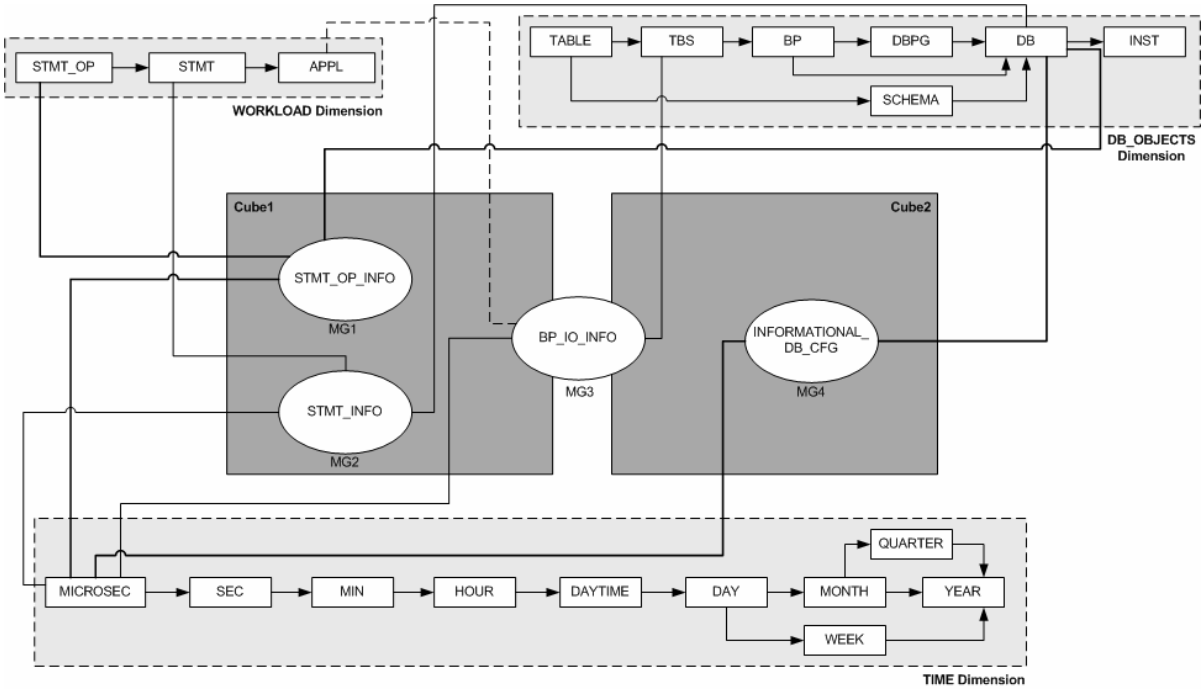


Abbildung 6.8: Visualisierung zweier exemplarischer Cubes [Wie05]

6.3.2.4 Bewertung des multidimensionalen Ansatzes

Dass ein integriertes Repository wie unser PWH keinen multidimensionalen Strukturen unterliegen muss, haben wir bereits gezeigt. Im Hinblick auf die Unterstützung der Analyse (durch den Menschen) sehen wir jedoch einige bedeutende Vorteile. Diese und auch die mit dem Paradigma verbundenen Nachteile werden in **Tabelle 6.4** kurz zusammengefasst und anschließend bewertet.

Unabhängig von den offensichtlichen Vorteilen der Konsolidierung, Vereinheitlichung und zentralen Speicherung von Performance-Daten verschiedenster Quellen und Formate, sehen wir die wesentliche Motivation im Einsatz von BI-Strukturen und -Techniken in der für die spätere Analyse entscheidenden Trennung von deskriptiven, eher statischen Stammdaten (Dimensionen bzw. Kontext wie Zeit, Workload, Ressourcen) und quantitativen, dynamischen Bewegungsdaten (Fakten bzw. Metriken wie die BPHR, der Durchsatz etc.).

Vorteile	Nachteile
Intuitivere, explorative Analysen	Zeitintensives Füllen der Cubes
Flexibilität	Visualisierbarkeit und Vermittelbarkeit bezüglich der n-Dimensionalität begrenzt
Schnelle, komplexe Auswertungen auf großen Mengen von Daten	Zusätzlicher Berechnungsaufwand (für Aggregationen) zur Laufzeit
Eine eigene, individuelle Sicht auf die gleichen Warehouse-Daten für jeden Nutzer	i.W. redundante Aufbereitung (Restrukturierung) bereits vorhandener Daten
Vereinheitlichung der heterogenen Datenbestände	Zusätzlicher Implementierungs-Aufwand und -Overhead (DWH Design Process)

Tabelle 6.4: Einige Vor- und Nachteile eines multidimensionalen Paradigmas

Die natürliche Unterscheidung in Fakten, Dimensionen und Hierarchien macht die Daten für die Nutzer leichter verständlich und „anfassbarer". Der Einstieg in eine Top-Down-orientierte Problem-Diagnose wird vereinfacht, die Komplexität wird reduziert. Intuitivere, assoziative, navigierende, aufeinander aufbauende Anfragen gegen die Daten zur schrittweisen Problem-Diagnose sind die Folge. Eine erhöhte Anzahl an Dimensionen und Hierarchien kann dem Nutzer jedoch auch mit graphischen Tools nicht mehr verständlich vermittelt werden.

Nutzer sind viel flexibler als mit bisherigen statischen Performance-Reports bzw. auch als mit spezifischen relationalen Strukturen. Sie können die Cubes in vielfältiger Weise manipulieren und die Betrachtungsweise anpassen (siehe dazu die typischen OLAP-Operatoren in Abschnitt 2.3.3). Dies ist weitaus effektiver und uneingeschränkter möglich, als bspw. mit einem herkömmlichen Ansatz und der Definition von Sichten. Die Vorteile bei der Diagnose von Performance-Problemen sowie der Ursachenanalyse sollten insbesondere im Abschnitt 7.2 deutlich werden, in welchem wir Anwendungs-Szenarien erörtern und Navigations-Beispiele aufführen.

Data Warehousing hat bereits in vielen Bereichen an Bedeutung gewonnen. Dementsprechend umfangreich ist das Angebot an vielseitig einsetzbaren DWH-, OLAP- und Data-Mining-Tools, die zur Definition und Ausführung des ETL, zur Visualisierung und zum Reporting auf bestehende Strukturen aufgesetzt werden können.

Es muss jedoch darauf hingewiesen werden, dass bestehende Quellen nicht ohne Weiteres multidimensionale Eigenschaften aufweisen können. Der Paradigmenwechsel ist verbun-

den mit einem umfangreichen DWH-Design-Prozess, von der Anforderungs-Analyse, über das konzeptuelle und logische, bis hin zum physischen Design. Entitäten und Beziehungen zwischen ihnen, Dimensionen, Hierarchien und Metriken müssen identifiziert, zusätzliche Strukturen erzeugt, Daten untereinander abgebildet, transformiert und schließlich übertragen werden.

Zu den offensichtlichen Nachteilen zählt vor allem die Performance-Verschlechterung bei zunehmender Größe und wachsendem Umfang der Daten. Mit steigender Zahl der Quellen und erhöhtem Datenvolumen wird nicht nur die Zeit zum ETL-basierten Füllen der Cubes (build-time), sondern auch je nach physischer Umsetzung die Zeit für Berechnungen in Form von Aggregationen entlang der Hierarchien während der Analyse (run-time) anwachsen. Kritisch betrachtet, kann in diesem Fall ein solches zentrales Repository zur Speicherung der Daten, welches größtenteils mit, den seltenen Lese-Operationen zahlenmäßig überlegenen, Einfüge-Operationen konfrontiert wird, einen Flaschenhals bei der Analyse darstellen und sich gegenläufig zu den langwierigen Query-Analysen verhalten (INSERT- vs. SELECT-Tradeoff). Im schlimmsten Fall muss die Lade-Zeit zur Analyse-Zeit hinzu- und mit Verzögerungen gerechnet werden. Im Normalfall jedoch werden Daten inkrementell und kontinuierlich in das PWH übertragen, so dass die Verzögerungen überschaubar und im Rahmen bleiben sollten. Bei wirklich zeitkritischen Analysen ist zudem auch eine direkte Auswertung auf den entsprechenden Quellen und der temporären Verknüpfung der Daten im Speicher denkbar. Wie wir aber bereits mehrfach angedeutet haben, sehen wir ebenso in der Verwendung von Data Marts eine ausreichende Entlastung. Analysen auf den langfristigen, historischen PWH-Daten hingegen, erfolgen zumeist unter weniger Zeit- und „Erfolgsdruck“.

Auch wenn Performance-Daten-konsolidierende Strukturen im PWH nicht zwangsweise multidimensional sein müssen, sehen wir im Hinblick auf die Unterstützung der Analyse durch den Menschen, durch auf einem generischen Datenmodell basierenden Performance Cubes, eine Vielzahl an Vorteilen.

6.3.3 Dynamik und Änderbarkeit

Neben der vom Anwendungs- und Problemkontext abhängigen Form der Datenbereitstellung bzw. Datenverarbeitung sollte vor allem die Datenpflege und -Aktualisierung als zentraler Prozess etabliert werden. Die Quellen der Performance-Daten sind im Verlauf der Zeit diversen Änderungen unterworfen, die idealerweise in dem PWH in Form einer nahezu lückenlosen Historie aufgezeichnet werden sollen.

Das generische Datenmodell und die multidimensionalen Entitäten sind so gewählt, dass strukturelle Änderungen und auch Änderungen der Instantiierung auf der operationalen Ebene lediglich die Instantiierung der PWH-Ebene betreffen (also keine Struktur- bzw. Schema-Änderungen erfordern). Neu hinzugefügte oder auch gelöschte Ressourcen werden demnach ebenso unproblematisch erfasst wie Änderungen an der Workload. Erweiterungen auf Ebene der Datenquellen werden zu einem geringen Problem, wenn das Repository aktualisierte Struktur-Informationen mit neuen (Dimensions-)Werten auf Instanz-Ebene umsetzt, mit deren Hilfe Daten aus den neuen Quellen eineindeutig identifizierbar sind.

Man kann demnach anhand der Fakten und der Zuordnung zu konkreten Dimensions-Level-Attribut-Werten herauslesen, in welchem Zeitraum Ressourcen gültig waren und welchen Einfluss sie auf das System hatten. Da jedoch Ressourcen noch immer existieren können, durch eine geänderte Workload jedoch nicht immer angesprochen werden und da

die Datensammlung nicht zwangsläufig kontinuierlich sein muss, bietet es sich an mittels Events über die Erzeugung, Änderung und Löschung von Ressourcen die Dynamik über die Zeit zu protokollieren.

Problematischer wird es, wenn sich z.B. der Name einer Ressource oder Eigenschaften ändern, die nicht in Form von Metriken (Measures) dargestellt werden und etwa zur Identifizierung der Dimension dienen. Laut [KiRo02] gibt es drei Methoden, durch welche derartige Änderungen (an Dimensions-Tabellen) erfasst und dokumentiert werden können. Sie sind unter dem Sammelbegriff *Slowly Changing Dimensions* bekannt:

- **Aktualisieren**
  Auf eine Historisierung wird hierbei komplett verzichtet. Es wird überprüft, ob ein älterer Datensatz existiert und im positiven Falle überschrieben. Andernfalls wird ein neuer angelegt.
- **Einfügen**
  Bei diesem Verfahren lässt man eine Historisierung von Dimensions-Tabellen oder einzelner Attribute zu. Dabei werden den Datensätzen Intervalle über die Zeit zugeteilt, in welchen sie gültig sind. Ist der Datensatz noch nicht vorhanden, so wird er einfach eingefügt. Falls ein älterer Datensatz vorhanden ist, so wird dieser nicht überschrieben und der neue mit entsprechendem Gültigkeitsintervall eingefügt. Dies entspricht der Methode, welche für ein PWH, wie es hier benötigt wird, am besten geeignet ist.
- **Einfügen mit neuer Spalte**
  Diese Methode lässt ebenfalls eine Historisierung zu. Dafür macht man jedoch Gebrauch von zusätzlichen Spalten. Ist ein älterer Datensatz noch nicht vorhanden, wird er neu angelegt. Falls er vorhanden sein sollte, wird der beinhaltenden Tabelle eine neue Spalte hinzugefügt und der neue Wert dort eingetragen. Diese Methode empfiehlt sich nur selten, da sie sowohl DDL- als auch DML-Anweisungen benötigt und unserer Forderung nach struktureller Stabilität des PWH zuwider läuft.

Auch in [Wie05] haben wir bereits andiskutiert, wie dem „Problem“ der Dynamik mittels Surrogat-Schlüsseln entgegengewirkt werden kann. Statt über ein Gültigkeits-Intervall zu wachen, sollten in unserem Fall alle sowohl statischen als auch dynamischen Eigenschaften und Beschreibungen einer Ressource in Form von Metriken (Measures) hinterlegt werden. Die Identifikation der entsprechenden Dimension erfolgt dann ganz einfach über künstliche Schlüssel (sogenannte Surrogate). Die Surrogat-Schlüssel brauchen in der Regel weniger Speicherplatz als natürliche Schlüssel und können oftmals auch automatisch mittels Identitätsspalten oder Sequenzen vom DBMS generiert werden. Auf diese Weise können inhaltliche oder auch strukturelle Änderungen an den natürlichen Primärschlüsseln ohne nennenswerte Aufwände und Folge-Änderungen vorgenommen werden.

6.4 Erzeugung und Befüllung multidimensionaler Strukturen

Um die notwendigen Prozesse zur Erzeugung multidimensionaler Ziel-Strukturen sowie deren relationale Repräsentationen und zum Transfer der Daten aus den Quell- in die Zielschemata unter Berücksichtigung vorab spezifizierter Abbildungs- und Transformationsvorschriften (Mapping) zu unterstützen, lassen sich eine Reihe von Tools einsetzen. Eine nahezu vollständige und fortwährend aktualisierte Liste mit Werkzeugen und Links zu den Hersteller-Homepages kann unter www.tdwi.org/marketplace eingesehen werden.

Zur Modellierung, Beschreibung und zum Austausch von Metadaten bzw. des benötigten Wissens in Data-Warehouse-Systemen eignen sich aber auch Standards, wie das Common Warehouse Model[39] der OMG. Dadurch lassen sich bspw. die Interoperabilität zwischen verschiedenen DWH-Werkzeugen steigern oder auch die im ETL-Prozess verwendeten Datenschemata von Quell- und Zieldatenbanken sowie die zwischen diesen stattfindenden Transformationen beschreiben. Es erlaubt zudem die Definition von Abbildungsvorschriften zwischen dem physischen Modell eines Data Warehouses und darauf aufsetzenden logischen Modellen, wie etwa des eines OLAP-Werkzeuges.

Die gängigsten Tools bieten zumeist jedoch umfangreiche Lösungen für spezialisierte Anwendungs-Szenarien zu einem hohen Preis. In [Wie05] demonstrieren wir daher anhand einer leichtgewichtigen, Plattform-unabhängigen, generischen Lösung, dem **Extensible Data Mart Framework (XDMF),** wie multidimensionale Strukturen auf einfache Art und Weise *erzeugt* und aus beliebigen Quellen, den Regeln einer eigenen Mapping-Algebra folgend, *befüllt* werden können.

Unser Framework besteht im Kern aus einer Meta-Struktur zur Repräsentation des *multidimensionalen Modells* und einem Meta-Modell zur Repräsentation des *Mapping* (und damit auch des ETL). Das multidimensionale Modell haben wir bereits (in Abschnitt 6.3.2) diskutiert. Ein (Schema-)**Mapping** beschreibt im Generellen, welche Elemente des Quell-Schemas mit welchen Elementen des Ziel-Schemas in Beziehung stehen. Ebenso wie ein Schema aus zwei verschiedenen, hierarchischen Typen von Elementen besteht (Entities und Attribute), so unterscheiden wir auch beim Mapping diese beiden Ebenen [Wie05]:

- Das *Entity Mapping* beschreibt welche Quell-Entitäten in Form von Tabellen bzw. Sichten auf welche Entitäten im Ziel-Schema abgebildet werden.
- Das *Attribute Mapping* spezifiziert welche Quell-Attribute auf welche Ziel-Attribute abgebildet werden.

Die Instantiierung dieser Strukturen soll ohne große Vorkenntnis auf einer abstrakten Ebene erfolgen können und dient jeweils einem (ROLAP Target) Generator zur Erzeugung des Ziel-(Star-)Schemas sowie einem (Query) Generator zur Erzeugung adäquater DML-Mapping-Anweisungen auf den Quellen für die Übertragung des Inhalts als Input. Basierend auf den materialisierten und instanziierten, high-level Mapping-Konstrukten kann der Query Generator im Anschluss an die Erzeugung der multidimensionalen Ziel-Strukturen eine low-level SQL-Query-Repräsentation generieren, die bestimmt, wie die Daten extrahiert, transformiert und letztlich in die Ziele geladen werden sollen. Dies erspart den Nutzern dem Umgang und die Wartung fehleranfälliger komplexer, selbst-geschriebener SQL-Statements.

Themenspezifische Data Marts können zusammen mit den komplexen SQL-basierten Daten-Transformations-Mappings auf einfache Weise, on-demand, zur Laufzeit erstellt und für die nachfolgenden Analysen verwendet werden. Die in den Quell-Systemen vorhandenen oder im Rahmen des generischen PWH-Datenmodells materialisierten Daten werden den Vorschriften folgend automatisch re-arrangiert und aggregiert in die multidimensionalen Zielstrukturen überführt. Die mittels des ROLAP Target Generator automatisch erstellten Cubes können somit per Load und den Anweisungen des Query Generator automatisch mit den transformierten Quelldaten befüllt und unmittelbar im Anschluss mittels SQL-OLAP-Erweiterungen oder auch BI-Tools analysiert werden.

[39] http://www.omg.org/spec/CWM/1.1/

Dabei werden das multidimensionale und das Mapping-Modell mitsamt Mapping-Operatoren mittels SQL in einfachen relationaler Strukturen abgelegt. Konsequentermaßen sind beide Generatoren dabei in Form von in DB2 hinterlegten Stored Procedures realisiert. Die ausführlichen Details und Besonderheiten der Modelle sowie der Quellcode lassen sich in [Wie05] nachschlagen.

Das XDM-Framework kann jedoch keine vollständige Automatisierung bieten. Es unterstützt lediglich bei der Erzeugung und Befüllung multidimensionaler Strukturen. Die Inhalte, sprich woher welche Daten wann kommen, müssen die Administratoren selbst definieren. Die Ziel-Schemata hängen in ihrer Definition von der Existenz und Struktur der Quellen ab. Auch der Mapping-Prozess kann daher ohne zusätzliches Wissen nicht vollständig automatisiert werden, da die reine syntaktische Repräsentation der Schemata sowie die eigentlichen Daten nicht in vollem Umfang deren Semantiken wiedergeben. Liegen jedoch für alle Quellen CIM-Modelle sowie entsprechende Provider vor, können Struktur und anschließend auch Daten bis auf wenige Besonderheiten vollständig automatisiert auf das generische Ressourcen-Modell und von dort einmalig auf multidimensionale Cube-Strukturen abgebildet werden [Ell10]. Andernfalls ist stets ein verlässlicher Dritter dafür verantwortlich zu definieren, wie verschiedene Schemata korrespondieren.

Das vorgestellte Framework kann ohne nennenswerten Aufwand um Abhängigkeiten, Nutzer-Interaktionen, Automatisierungs-Richtlinien, Standards und um graphischen Support erweitert bzw. angereichert werden, um auch speziellen Anforderungen zu genügen. Auch die Bibliothek an vorhandenen atomaren und komplexen Mapping-Operatoren kann um neue bzw. kombinierte Konstrukte erweitert werden. Neben den Kontext-unabhängigen sind auch Anwendungs-spezifische Operatoren denkbar. Eine zusätzliche Staging Area kann hierfür das Potential zur Erzeugung temporärer Objekte für weitaus komplexere Transformationen bereitstellen (siehe Abschnitt 8.2).

Das manuelle Füllen der Struktur-beschreibenden Tabellen kann mühsam und fehleranfällig sein und sollte in Anwendungen der „realen Welt“ vermieden bzw. durch graphische Front-Ends unterstützt werden. Das intuitive Erzeugen multidimensionaler Cube-Strukturen durch Maus-Eingaben und visuelle Unterstützung reduziert die initiale Entwurfsphase sowie die folgenden Wartungs-Aufwände. Wir werden im folgenden Abschnitt zudem einen Mechanismus zur Automatisierung der Erzeugung, Befüllung und Selektion von Performance Cubes kurz vorstellen und in Abschnitt 8.4 in einer konkreten Architektur eingebettet vertiefen.

6.5 Der Cube Advisor

Eine zusätzliche Ebene zur multidimensionalen Restrukturierung des Inhalts der Performance-Daten-Quellen ergibt nur dann Sinn, wenn weder die Performance in hohem Maße beeinträchtigt, noch die Analysen dadurch bedeutend verlangsamt werden. Es sollten daher eine Reihe von Automatismen ins Spiel kommen, um eine derartige Konstellation zu rechtfertigen. Vorstellbar sind u.a. die Automatisierung der Erzeugung bzw. das automatische Auffinden sinnvoller und möglicher Cube-Strukturen, die Automatisierung des Mapping und des inkrementellen Ladevorgangs, ein automatisiertes Vorschlagen von Cubes für bestimmte erkannte Problemstellungen und auch eine Automatisierung der Diagnose, sprich des Navigierens durch die Cubes.

Wir haben in [Wie05] mit der „On Demand Performance Expert Cubes”-Architektur als konkretes Anwendungs-Szenario des XDMF bereits zeigen können, dass Performance- und Problem-Analysen über einen langfristigen Zeitraum durch den Einsatz des

multidimensionalen und OLAP-Paradigmas profitieren. Aufbauend auf das Schema zur langfristigen Speicherung von Performance-Daten des IBM Performance Expert werden zur Build-Time Cube-Strukturen (Dimensionen, Hierarchien, Fakten) und entsprechende Mappings durch die Administratoren initial erzeugt. Zur Run-Time können diese globalen Definitionen an den Kontext angepasst und verfeinert werden.

Eine der Schwierigkeiten besteht jedoch in der Auswahl passender Cube-Strukturen für die Performance-Problem-Analyse. Dies kann u.U. als hochgradig nicht-trivialer, möglicherweise iterativer Trial-and-Error-Prozess angesehen werden, der ausschließlich durch erfahrene DBA ausgeführt werden sollte. Der Nutzer sollte eine Idee haben, welche Measures (Fakten) bzw. Cubes für die anstehende Analyse-Problematik am geeignetsten bzw. als Einstiegspunkt zu betrachten sind.

In [Wie05] beschreiben wir daher auch die Idee einer autonomen Performance**-Cube-Advisor**-Komponente zur *(problemgesteuerten) Auswahl***,** *Erzeugung und Befüllung* geeigneter Cubes. In dem darauf aufbauendem Patent der IBM [RWH06] wurden die Konzepte erweitert, die Verfahrensweisen ausführlich beschrieben und die allgemeine Architektur näher gebracht.

Zur Entwurfs- und auch zur Laufzeit sind die Administratoren mit der Erzeugung und Wartung sämtlicher anwendungsspezifischer Cube-Strukturen konfrontiert. Unser Cube Advisor soll basierend auf der durch den Nutzer übermittelten Problem- und Situations-Charakteristik bei der Auswahl und der Erzeugung geeigneter Cubes unterstützen. In Abschnitt 8.4 gehen wir näher auf die Architektur ein und demonstrieren, wie der Cube Advisor in ein bestehendes Performance-Monitoring- und -Tuning-Framework eingebettet und sowohl die Analyse als auch die Planung unterstützend eingesetzt werden kann.

Die endgültige Auswahl aus der Liste an potentiellen Cube-Kandidaten obliegt letztlich noch immer dem DBA. Die Navigation durch einen vorab selektierten Cube kann unzureichend sein und es notwendig machen, weitere Cubes in die Auswahl und damit in die Analyse einzubeziehen. Einen Ausblick auf die Möglichkeiten zur Automatisierung der Analyse (automatischer Drill-Down) ist in Abschnitt 7.2.4 gegeben.

6.6 Optimierungspotentiale

Da sich im Laufe der Betriebszeit des PWH sehr große Daten-Mengen ansammeln, kann eine Auswertung dieser möglicherweise sehr große Last auf dem PWH verursachen. Ein effizienter Umgang mit der Datenflut an hochfrequent aktualisierten und zahlreichen Performance-Daten, beispielsweise durch eine adäquate Aufbewahrungszeit und Techniken der Aggregation und Komprimierung vergangener Daten ist unerlässlich, um zeitnahe, performante Analysen auf dem zentralen Datenbestand zu ermöglichen.

Die Eigenschaft zur intuitiven und interaktiven Auswertung der Daten (mittels OLAP) erfordert eine zügige Gewinnung der Daten durch kurze Antwortzeiten. Letztere hängen jedoch in starkem Maße von der Zahl der zusätzlichen Berechnungen ab. Neben den durch die Komplexität der Queries bedingten langen Anfragezeiten weckt vor allem die Größe des Datenbestandes den Bedarf für zusätzliche Optimierungs-Techniken (zur Senkung der Anfragezeiten). Der Query Generator unseres XDMF kann beispielsweise von diesen und anderen Optimierungs-Techniken Gebrauch machen, um die Informationsgewinnung zur Laufzeit zu beschleunigen.

Kurze Antwortzeiten werden in der Regel nur für die geläufigen und zeitkritischen Anfragen gefordert, die hierfür auf einem optimierten physischen Schema beruhen.

Abgesehen von den traditionellen Indexierungs-Mechanismen ist im DWH-Umfeld die Vor-Aggregation und Speicherung häufig angefasster Daten in materialisierten Sichten sehr sinnvoll, um den Berechnungsaufwand zur Laufzeit zu reduzieren [Leh03]. In Abschnitt 2.3.3 haben wir bereits einige in der Literatur verbreitete Optimierungstechniken, insbesondere im Hinblick auf die Vorab-Materialisierung, kennengelernt, die ein Inbetrachtziehen der komplementären Anforderungen an Aktualität, Zugriffsgeschwindigkeit und Speicherplatzausnutzung notwendig machen. Techniken wurden entwickelt, um zu entscheiden, welche Teilmengen der Daten eines Cube im Voraus zu aggregieren sind, um zu bestimmen, wie groß die multidimensionalen Aggregate werden und auch um die Sichten zu indexieren. Die Ansätze zur Materialisierung von Aggregationen beeinflussen sowohl die Größe der Datenbank, als auch die Antwortzeiten von Anfragen. Je mehr Werte vorberechnet und gespeichert werden, desto wahrscheinlicher ist, Werte anzufragen, die bereits berechnet wurden. Daher sinkt die Zeit für die Berechnung der Antwort. Würden hingegen sämtliche Werte vorberechnet werden, so stiegen sowohl die Zeit zur eigentlichen Aggregation als auch die Größe der Datenbank ins Unermessliche.

Der Entwurf effektiver materialisierter Sichten erfordert eine adäquate Vorabplanung. Der Designer muss die Anfragen und das Nutzerverhalten teilweise vorhersehen, um Muster des Zugriffs auf die Tabellen und die kürzlichen Aggregationen zu identifizieren und sich dieses Wissen zu Nutze zu machen. In [Sap99] findet sich ein interessanter Ansatz bezüglich dieser Problematik. Der Autor stellt ein mathematisches Modell und eine graphische Notation zur Erfassung von Wissen über typische multidimensionale Interaktions-Muster in OLAP-Systemen vor. Hierbei wird die Session-basierte, interaktive und navigierende Natur der Nutzer-Anfragen berücksichtigt. In der Veröffentlichung ist auch die Rede von einer Architektur zur Beschleunigung von OLAP-Anfragen unter Nutzbarmachung von spekulativen Ausführungstechniken und Techniken der Vorhersage des Nutzer-Anfrage-Verhaltens. Die Tauglichkeit der Vorhersage des Verhaltens der Nutzer(-Anfragen) muss sich jedoch im Performance-Analyse-Szenario mit potentiell nicht vorhersehbaren Systemzuständen, die u.U. eine sofortige Reaktion erfordern, zunächst beweisen.

Typische Optimierungstechniken im DWH-Umfeld vernachlässigen die Wartung bzw. Aktualisierung der zusätzlichen physischen Strukturen. Update-Kosten werden lediglich zu festen, zumeist nächtlichen Zeiten bzw. im Batch-Betrieb berücksichtigt. In Wahrheit jedoch verschieben sich die Szenarien und vor allem unser Use Case mehr und mehr in Richtung Real-Time Data Warehousing [SaCa09], bei dem Daten kontinuierlich aktualisiert werden (müssen).

Abgesehen von den typischen Techniken kann bspw. auch ein separater Fakten-Tablespace (sowie ein entsprechender Bufferpool) mit einer größeren Pagesize erzeugt werden. Da die Fakten-Tabellen im Gegensatz zu den Dimensions-Tabellen in der Regel im Sinne der Zahl der Attribute recht breit werden, scheint die genannte Vergrößerung sinnvoll, um mehr Spalten pro Seite und damit Tabellen-Zeilen unterzubringen. Die Größen für das Prefetching und Pagecleaning sollten klein gehalten werden, da multidimensionale Anfragen auf denselben Daten beinahe ausschließlich einmalig bzw. zu sehr unregelmäßigen Zeiten erfolgen, so dass Caching-Mechanismen wohl nicht den erhofften Erfolg versprechen. Außerdem ist es überlegenswert, für die Faktentabellen Kompressions-Mechanismen einzusetzen, sofern die Kosten der (De-)Kodierung den Nutzen nicht überschreiten.

Auch wenn das Konzept des Multidimensional Clustering [IBM06c] auf den ersten Blick äußerst geeignet zur Steigerung der Anfrage-Geschwindigkeit erscheint, hat es in unserem Fall doch einige gravierende Nachteile. Hierbei werden die Daten physisch gemäß der logischen Strukturierung angeordnet. Es wird also, mit anderen Worten, eine Menge an

unabhängigen Clustering-Indexen für die Dimensionen der Fakten-Tabelle erzeugt, um die typischen Star-Schema-Anfragen zu unterstützen. Die Dichte der Zellen muss hierbei jedoch gesondert berücksichtigt werden. Für jede eindeutige Kombination der Werte der gewählten Indexierungs-Dimensionen wird ein Extent (sprich eine Menge an Seiten) allokiert. Wenn nur wenige Tupel die Werte dieser Dimension besitzen, dann wird unnötig Platz verschwendet. Wird nun, wie bei uns der Fall mit den Surrogat-Schlüsseln auf die jeweiligen Dimensionen, eine Schlüssel-Spalte für die Indexierung herangezogen, würde jedes Extent lediglich ein Tupel beinhalten und damit der worst case für diese physische Organisationsstruktur eintreten. Dies würde in einer enormen Speicherverschwendung ohne jegliche Vorteile bei der Anfrage-Auswertung resultieren.

Kapitel 7

Anwendung von Performance-Daten

In dem vorangegangen vorgestellten Performance Warehouse werden aktuelle bzw. historische Performance-Daten aus verschiedenen Quellen mit Informationen über ihren Kontext integriert und langfristig gespeichert. Es kann und soll somit als zentraler Ausgangspunkt für sämtliche Analysen und Betrachtungen auf den Daten über das überwachte System durch den Menschen oder etwaige Automatismen dienen. Das Spektrum der Anwendungs-Möglichkeiten reicht von der sofortigen bzw. retrospektiven Problem-Erkennung, über die -Diagnose, bis hin zur -Auflösung, der Kapazitätsplanung, der Trend-Erkennung und der Optimierung von System-Ressourcen. Eine exemplarische Aufstellung der relevanten Auswertungs-Möglichkeiten auf dem PWH ist in **Abbildung 7.1** visualisiert.

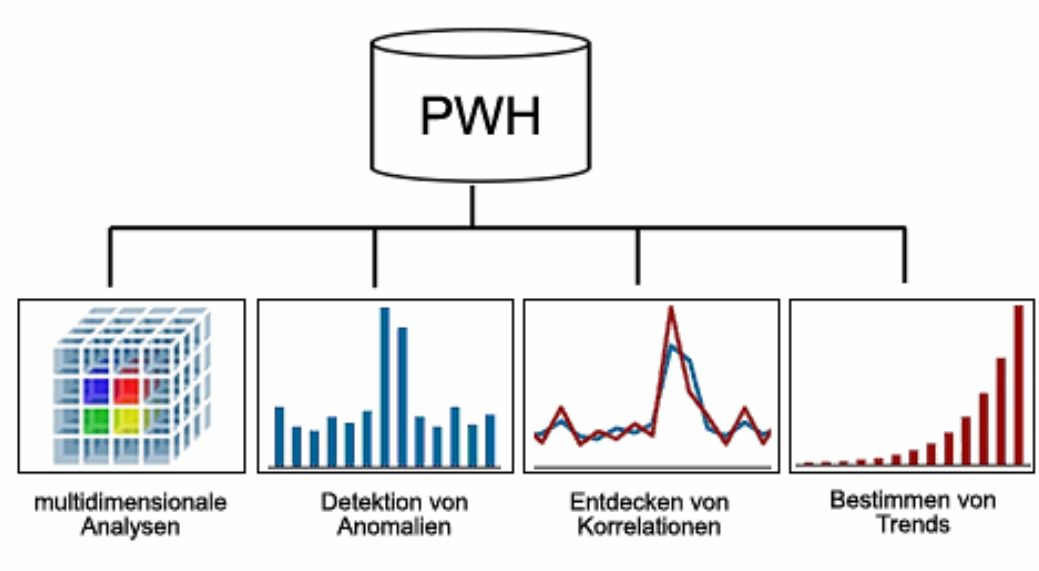


Abbildung 7.1: Exemplarische Auswertungs-Möglichkeiten auf dem PWH

Insbesondere die Techniken des Data Warehousing ermöglichen eine flexible, multidimensionale Analyse und Navigation innerhalb der Performance-Daten. Darüber hinaus bietet das Data Mining Verfahren und Werkzeuge, die scheinbar zusammenhanglose Daten nach noch nicht bekannten, wissenswerten Zusammenhängen durchsuchen, Daten aufspüren, kombinieren und neue Informationen bzw. Erkenntnisse zur Verfügung stellen. Um mit der ständig anwachsenden Menge gesammelter, gespeicherter oder auch im Rahmen der Wissens-Verwaltung generierter, Performance-relevanter Daten zurechtzukommen, sie zu interpretieren und für Optimierungszwecke zu nutzen, werden vor allem automatisierte Verfahren zur Verwaltung und Analyse benötigt.

Das vorliegende Kapitel beschreibt mögliche Anwendungsgebiete und Verknüpfungsmöglichkeiten der (Meta-)Daten eines Performance Warehouse als Planungs- und Entscheidungsgrundlage im Umfeld des manuellen sowie des autonomen Datenbank-Performance-Tuning. Im **Abschnitt 7.1** werden zunächst Anforderungen an die Analyse von Performance-Daten zusammengefasst. Anhand jener hinterlegten Daten und einer multidimensionalen Sichtweise sollen Performance-Probleme (automatisch) erkannt, korreliert und im Detail retrospektiv bzw. in Echtzeit diagnostiziert werden, um die eigentlichen Problem-Ursachen ausfindig zu machen und schließlich aufzulösen. **Abschnitt 7.2** zeigt diesbezüglich exemplarische Vorgehensweisen zur (automatisierten) Navigation durch die restrukturierten Daten auf. Schließlich werden in **Abschnitt 7.3** mögliche Einsatz-Szenarien des Data Mining im autonomen Datenbank-Tuning zum Beobachten und Lernen des Normalzustands, zur Identifikation und Berechnung von Trends und Korrelationen von Problemen sowie zum Aufdecken von Anomalien (Ausreißern) im Überblick zusammengefasst. Die Performance-Daten können weitergehend dazu genutzt werden, um ein in **Abschnitt 7.4** vorgestelltes Modell über die Workload sowie ein in **Abschnitt 7.5** aufgezeigtes Modell über das System mit seinen Ressourcenabhängigkeiten zu erstellen und in den künftigen (intelligenteren) Analysen zu berücksichtigen.

7.1 Anforderungen an die Daten-Analyse

Die vorgelagerten Prozesse zur Daten-Sammlung, -Speicherung und -Bereitstellung bestimmen im Wesentlichen die Möglichkeiten der Daten-Analyse und sollten nicht in Isolation betrachtet werden, um zumindest die nachfolgenden Anforderungen gewährleisten zu können:

- Analysen auf dem Datenbestand des PWH sollen es **in Echtzeit** ermöglichen, problematische Änderungen in der Leistungscharakteristik zu erkennen und bei Bedarf adäquate und vor allem zeitnahe Reaktionen anzustoßen. Dies soll vor allem geschehen, um die Wirkung der Probleme zu reduzieren („Schadensbegrenzung"). Die etwas langwierigere Suche nach den Ursachen von Problemen, die Erkennung von Zusammenhängen oder auch das Bestimmen von Trends können später „in Ruhe" und über einen signifikanten Teilbereich der Performance-Daten-Historie erfolgen.
- Es sind Optimierungs-Methodiken der Analyse in Bezug auf die zu erwartenden **Zugriffscharakteristika** zu berücksichtigen. Typischerweise arbeiten wenige Nutzer gleichzeitig mit dem System. Die Analysen reichen von reaktiven Ad-hoc-Anfragen in Echtzeit und eingeschränktem Zeithorizont bis hin zu historischen Analysen zur Bestimmung von Ursachen von Problemen oder auch Trends. Im letzten Fall sind große Daten-Mengen über einen langen Zeithorizont betroffen.
- Eine zentrale Forderung an die Analyse der Daten ist die bereits in Abschnitt 6.2. eingeführte **integrierte und flexible Auswertbarkeit**, so dass sowohl Daten verschiedener Ressourcen, als auch Daten verschiedener Schnittstellen und Tools gemeinsam betrachtet werden können. Durch geeignete Verknüpfungs-Möglichkeiten zur Analyse-Zeit soll also gewährleistet werden, dass Performance-Daten über verschiedene Ressourcen hinweg korreliert betrachtet werden können. Darüber hinaus soll die Verknüpfung zwischen dynamischen Performance-Daten, semi-dynamischen Struktur- und Topologiedaten und den Ablaufdaten des Tuning-Systems möglich sein (siehe Abschnitt 6.1).
- Autonome Analyse- und Planungsprozesse, aber auch Administratoren sollen die Möglichkeit haben, zur Zeit der Auswertung die Betrachtungsweise auf die Daten zu ändern und Problem-Situationen aus **mehreren Blickwinkeln zu observieren**.

Die Variabilität reicht z.B. von der Änderung des Detaillierungsgrades, über die Anpassung der Betrachtungs-Dimensionen, bis hin zur Korrelation mit zusätzlichen semantischen Informationen.

- Eine graphische und intuitive Präsentation und Verwertung der aufbereiteten Daten und Geschehnisse über ein **zentrales User-Interface** (bspw. über konfigurierbare Dashboards) soll vorausgesetzt werden. Dazu zählen auch Mechanismen zur visuellen Darstellung bzw. Ermöglichung flexibler, intuitiver Navigation, Filterung und Selektion von Performance-Daten. Zur Visualisierung und Auswertung der Kennzahlen sowie zur flexiblen Navigation in der Datenbasis können prinzipiell Standard-Frontends (z.B. aus dem BI-Umfeld) genutzt werden. Der Einsatz von auf das Repository zugeschnittenen Speziallösungen und Eigenentwicklungen bietet gewisse Vorteile hinsichtlich Flexibilität und Funktionalität. Den universell einsetzbaren Werkzeugen ist i.d.R. die spezifische Semantik der Performance-Daten-Analyse typischerweise nicht bekannt.

7.2 Diagnose von Performance-Problemen

Herkömmliche Performance-Analysen sind eher statischer Reporting-Natur [Ree05]. Administratoren versuchen sich durch die Vielzahl an vordefinierten Performance-Kennzahl-Daten zu „wühlen“, um Engpässe in den Ressourcen bzw. Ursachen für Performance-Probleme und Auswirkungen von Konfigurationsparameter-Änderungen auf das System bzw. auf seine repräsentativen Metriken zu erkennen. In dem PWH werden typische Performance-Daten, anders als im herkömmlichen statischen Sinne, effizient und nutzer- bzw. problemspezifisch in multidimensionalen Data Marts abgelegt. Wie wir bereits kennengelernt haben, werden die für die Data Marts zugrundeliegenden Quell-Performance-Daten aus verschiedensten Performance-Daten-Quellen extrahiert, transformiert und in die Zielstruktur(en) geladen. Diese können schließlich zur intuitiven und hierarchischen Problem-Erkennung bzw. -lösung herangezogen werden. Sicher ist es in besonders kritischen Situationen mit Echtzeitanforderungen noch immer sinnvoll, lokale Analysen direkt auf den Quellen durchzuführen. Durch die Fortschritte im (Real-Time) Data Warehousing lässt sich dieser Anteil aber so gut wie vermeiden.

Die Performance-Problem- und Ursachen-Analyse ist überwiegend explorativ und maßgeblich charakterisiert durch ein interaktives Durchstöbern und wiederholtes Lesen großer Daten-Mengen. Da man nicht weiß, wonach man genau sucht, kann ein Großteil der Anfragen unbefriedigende Ergebnisse liefern. Die konsequente Nutzung von BI- und DWH-Konzepten für die Performance-Analyse bietet große Potenziale, um ein tiefgehendes und breites Verständnis des operativen Geschehens zu erreichen und somit zu fundierten und frühzeitigen Entscheidungen zu gelangen. Mittels Techniken zur intuitiveren und hierarchischen Navigation durch aufeinander aufbauende assoziative Anfragen, versuchen wir dem Ziel der schrittweisen Problem-Diagnose näherzukommen und damit sowohl die Administratoren, als auch autonome Mechanismen zu unterstützen.

7.2.1 Multidimensionale (OLAP-)Navigation

Die Analysten navigieren durch den adäquat restrukturierten PWH- bzw. Performance-Data-Mart-Datenbestand mit dem Ziel, Performance-Bottlenecks zu erkennen, Ursachen für Performance-Probleme zu identifizieren oder auch um die Auswirkungen von Tuning-Aktionen bzw. Konfigurations-Änderungen auf das System und die repräsentativen Metriken zu ermitteln.

Multidimensionale Anfragen werden typischerweise aufeinander aufbauend, eine nach der anderen, ausgeführt und beschreiben einen explorativen Navigations-Pfad. Die Analyse auf der multidimensionalen Repräsentation der Performance-Daten beginnt dabei von einer abstrakten (Top-Level-)Sicht mit dem Ziel der Ergründung einer spezifischen Menge an Problem- oder Situations-relevanten Metriken (bzw. deren Werte), die auf ein Problem hindeuten. Das Ziel ist, von Anfrage zu Anfrage, in Beziehung stehende Metriken bzw. Sichten für die weitergehende Analyse zu identifizieren. Der Analyst kann sukzessive den Detaillierungsgrad erhöhen um die interessanten Einzelheiten näher zu ergründen.

Die Nutzung der Dimensions-Hierarchien zur Navigation bietet den Analysten die nötige Flexibilität, um verschiedene Interessenspfade explorativ zu erkunden. Eine Kombination an Drill-Down- bzw. Roll-Up- sowie Slice- und Dice-Operationen (siehe Abschnitt 2.3.3) auf den verschiedenen Dimensionen offenbart möglicherweise Korrelationen zwischen als unzusammengehörig geglaubten Daten-Ausschnitten. Im Gegensatz zu eher statischem Reporting, ist es nicht mehr notwendig einen neuen Report erstellen zu müssen, um den Detaillierungsgrad zu erhöhen. Durch entsprechende OLAP-Methodiken zur Organisation und Präsentation der Daten für die Analysen können Experten produktiver arbeiten und Anfänger schnellere Erfolge erzielen. Mit einem entsprechendem Tool reicht oft eine einfache Nutzer-Interaktion, um einen Wert auf eine tiefere Detaillierungsstufe herunterzubrechen.

Aus dem DWH- und BI-Umfeld gibt es ein breites Spektrum an Tool-Unterstützung, wodurch der Zugang zu wertvollen Informationen ohne größeren Aufwand und das Lernen einer neuer Anfrage- bzw. Programmiersprache gewährt werden kann. Unter dem Begriff OLAP, Online Analytical Processing, werden zumeist Technologien, Methoden und Tools zusammengefasst, welche die Ad-hoc-Analyse multidimensionaler Informationen unterstützen. Die Webseite www.tdwi.org/marketplace birgt eine umfangreiche Liste mit geeigneten Analyse- und Visualisierungswerkzeugen, von den Marktführern mit einem hohen Preis bis hin zu eher unbekannten Konkurrenten mit Freeware-Software. Die Auswahl des passenden Tools richtet sich nach den Analyse-Bedürfnissen, dem Erfahrungsstand der Analysten und aus technischer Sicht nach den unterstützten Formaten bzw. Schnittstellen.

Viele Organisationen verwenden Excel sowie die zugehörigen Dateien als Medium zum Informations-Austausch, da sowohl Analysten als auch IT-Anfänger intuitiv und ohne großen Einarbeitungsaufwand mit dem Look-And-Feel der Technologien umzugehen wissen. *Excel Pivot-Tabellen* beispielsweise ermöglichen die Organisation und Betrachtung von multidimensionalen, importierten Daten auf verschiedene Weisen in Form von Kreuztabellen und Graphen. Zwischen den verschiedenen Darstellungsmodi kann per Drag&Drop beliebig gewechselt werden. Darüber hinaus bietet Excel einfache Verfahren zur Trend-Berechnung und -Darstellung. In [Wie05] wird der Umgang mit derartigen Pivot-Tabellen auf einem exemplarischen Performance-Daten-Bestand demonstriert. Leider lassen sich weder die Erzeugung derartiger Strukturen, der Import, noch die Analyse der Daten über geeignete Schnittstellen (bisher) automatisieren. Die Analysen können erst nach der relativ umständlichen Erzeugung der Tabellen und Hilfs-Strukturen durchgeführt werden.

Neben dem Einsatz anspruchsvoller Tools können die multidimensionalen Daten auch direkt mittels SQL und den ANSI-SQL99-OLAP-Extensionen analysiert werden [Mog05]. OLAP erleichtert die Analyse von Performance-Indikatoren, indem Auswertungen unter verschiedenen Gesichtspunkten in einem interaktiven Abfrageprozess durchgeführt werden können. Das flexible Navigieren in den Datenwürfeln erfolgt mittels der aus Abschnitt 2.3.3 bekannten mehrdimensionalen Operationen. Zum Operationsumfang gehören eine

erweiterte Gruppierungsfunktionalität (Grouping Sets, die GROUPING-Funktion, der ROLLUP- und CUBE-Operator etc.) sowie neben der reinen Aggregation auch die Partitionierung, das Windowing und das Ranking. Dies erlaubt komplexere OLAP-Anfragen in SQL kompakter formulieren und effizienter ausführen zu können und damit eine in SQL92 nur schwer formulierbare Vielzahl statistischer Auswertungen auf dem Data-Warehouse-Datenbestand überhaupt erst möglich zu machen.

Eine typische OLAP-Star-Schema-Anfrage umfasst meist eine große Anzahl an Tabellen und beinhaltet vor allem Equi-Join-Prädikate zwischen Fakten- und Dimensions-Tabellen sowie lokale, hoch selektive Bereichs-Einschränkungen auf den Dimensions-Tabellen. Derartige OLAP-Anfragen haben aufgrund ihrer Komplexität und der Adressierung großer Daten-Mengen in der Regel eine weit längere Ausführungszeit als OLTP-Anfragen, treten dafür aber auch seltener auf.

Durch entsprechende OLAP-Methodiken zur Organisation und Präsentation der Daten für intuitive Analysen können Experten produktiver arbeiten und wenig erfahrene Nutzer schnellere Erfolge erzielen. Auf eine initiale Einführungs- und Lernphase kann jedoch, wie schon im Abschnitt 6.6 angeklungen, nicht verzichtet werden. Anforderungen müssen spezifiziert, entsprechende multidimensionale Datenmodelle entworfen und schließlich mit einer zusätzlichen Warehouse-Architektur mit den Daten gefüllt werden. Personal muss in der Administration und Wartung geschult werden. Unabhängig von dem für das erstrebte Navigieren zusätzlichen Overhead zum Erzeugen und Füllen der repräsentativen Datenstrukturen, müssen auch hohe Anforderungen an die konsistente, detaillierte und qualitativ hochwertige Sammlung und Integration der Daten gestellt werden.

7.2.2 Das Aggregationsgitter

Die sequentielle Natur und Diversität der explorativen Möglichkeiten (Navigationspfade) im Rahmen der multidimensionalen Analyse können am deutlichsten mittels eines Aggregationsgitters (Aggregation Lattice) nahegebracht werden. Das **Aggregationsgitter** repräsentiert einen Teilmengenverband über die einzelnen Gruppierungsattribute (Dimensionen) [HRU96]. Jeder Knoten im Aggregationsgitter entspricht einer konkreten gruppierten Anfrage und damit auch der Möglichkeit, eine materialisierte Sicht zu bilden. Kanten bzw. Pfeile zwischen den Knoten geben an, welche Aggregationen bzw. Gruppierungen aus welchen anderen berechnet werden können. Die typischen damit einhergehenden SQL-Aggregations-Anfragen auf einem Star-Schema haben dabei die in **Listing 7.1** abgebildete Form.

```
SELECT    <gruppierungs-attribute>, AGG(<fakten>)
FROM      <fakten-tabelle(n)>, <dimensions-tabellen>
WHERE     <join-bedingungen> AND <restriktionen>
GROUP BY  <gruppierungs-attribute>
```

Listing 7.1: Schema von SQL-Aggregations-Anfragen

Das Aggregationsgitter repräsentiert den Lösungsraum mittels Knoten, die eindeutige Kombinationen von Dimensions-Ebenen, also Anfragen auf dem Stern-Schema darstellen. Die Kanten stehen dabei für einfache Drill-Down- bzw. Roll-Up-Operationen. Die Kunst der Analyse liegt nun darin, nur diejenigen Queries von Bedeutung zu betrachten und auszuführen.

Abbildung 7.2 soll verschiedene „Bewegungsstrategien“ und den potentiellen Navigations-Prozess eines Analysten durch die beiden Dimensionen aufzeigen. Wir legen

dafür einen Cube mit lediglich 2 Dimensionen zu Grunde: (T)able-(D)atabase-(I)nstance sowie (M)inute-(H)our-(D)ay. Jeder der neun Knoten stellt eine Kombination der Dimensions-Hierarchie-Level-Instanzen bzw. eine SQL-Aggregations-Anfrage dar. Der Knoten auf unterster Ebene verkörpert die höchste Abstraktionsebene mit der geringsten Kardinalität[40]. Auf dem Weg bis zum obersten Knoten erhöht sich der Detaillierungsgrad soweit, dass dort die Fakten auf Table- und Minute-Ebene beschrieben werden.

Sowohl die roten durchgezogenen, als auch die blauen gestrichelten Pfeile verdeutlichen jeweils einen der verschiedenen sequentiellen Pfade unter ausschließlicher Verwendung von Drill-Down- und Roll-Up-Operationen, um ein Datenbank-bezogenes Problem zu lokalisieren. Offensichtlich erscheint der blaue, gestrichelte Pfad in Bezug auf Zeit und Aufwand effektiver, um zum „Ziel“ zu gelangen. Bedauerlicherweise ist der kürzeste Pfad nicht immer der für den Menschen offensichtlichste. Daneben können Analyse-Pfade auch in einer „Sackgasse“ enden, so dass die Nutzer in der Historie zurückgehen müssen, um nach alternativen Pfaden zur Problem-Erkennung bzw. -Lösung zu suchen.

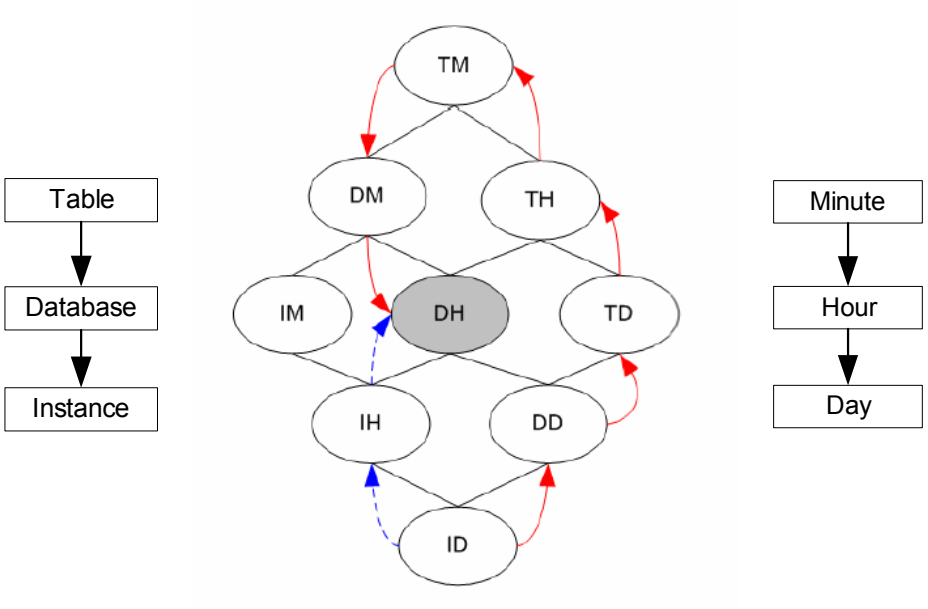


Abbildung 7.2: Ein exemplarisches Aggregationsgitter mit zwei Navigations-Pfaden

Der Ansatz des Aggregationsgitters wird vornehmlich zur Ermittlung von potentiellen Kandidaten für materialisierte Sichten angewendet. Einige der hierfür notwendigen Techniken und Algorithmen werden bspw. in [HRU96] beschrieben. Aufgrund der exponentiell mit der Zahl der Dimensionen (Gruppierungsattribute) wachsenden Zahl der Knoten im Aggregationsgitter, d.h. der potentiell materialisierbaren Sichten, muss aus Speicherplatz- sowie Aufwandsgründen eine Auswahl getroffen werden oder das Aggregationsgitter reduziert werden. Funktionale Abhängigkeiten zwischen den Gruppierungsattributen, wie etwa zwischen den verschiedenen Ebenen einer Dimensions-Hierarchie (Tabelle → Schema → Datenbank → Instanz), erlauben bspw. eine solche Reduktion.

[40] Die Kardinialität eines Knotens beschreibt die Zahl der Ergebnis-Tupel der durch ihn repräsentierten Anfrage.

7.2.3 Ein exemplarisches OLAP-Navigations-Szenario

Der DBA entdecke (möglicherweise in Reaktion auf Nutzer-Beschwerden über lange Wartezeiten) eine ansteigende (Datenbank-)Antwortzeit. In der Annahme, dass dieses Problem aufgrund von mehreren gleichzeitig auf der Datenbank aktiven Applikationen verursacht wird, fokussiert er seine Betrachtungen zunächst auf die Bufferpool-Aktivität, als einen der maßgeblichen Indikatoren für die Gesundheit des Systems.

Ein typisches aus mehreren aufeinander aufbauenden Schritten bestehendes Vorgehen zur Problem-Isolation soll mittels der nachfolgenden Tabellen vermittelt werden. Der Nutzer analysiert dabei den in Tabelle 6.3 aus Abschnitt 6.3.2.3 aufgeführten Cube mit der Nummer 4 anhand der BPHR. Zu Beginn der Analyse berechnet der DBA die BPHRs auf höchster Abstraktions-Ebene in den jeweiligen Hierarchien. Die in **Listing 7.2** dargestellte Query bestimmt genau diese durchschnittliche BPHR auf Datenbank- und Bufferpool-Ebene und resultiert in **Tabelle 7.1**.

```
SELECT dbo.db_name as DB, dbo.BP_name as BP,
       AVG(fact.pool_hit_ratio) as BPHR
FROM   d_db_objects dbo, f_bp_io_info fact, d_time t
WHERE  fact.d_db_objects_id = dbo.d_db_objects_id
       AND fact.d_time_id = t.d_time_id
GROUP BY ROLLUP(dbo.db_name, dbo.BP_name)
```

Listing 7.2: Exemplarische SQL-Aggregations-Query

Der hier verwendete, intra-dimensionale ROLLUP-Operator dient der Generierung von Attribut-Kombinationen entlang der Hierarchie in einer Dimension. Die Zwischensummen dieser Gruppierung werden durch das Schlüsselwort ALL dargestellt.

DB	BP	TBS	Zeit-Intervall	BPHR
PEDEMO	BUILDINGS	ALL	ALL	99.61
PEDEMO	FRUITS	ALL	ALL	20.35
PEDEMO	IBMDEFAULTBP	ALL	ALL	89.55
PEDEMO	LOCKTEST	ALL	ALL	99.69
PEDEMO	MOTION	ALL	ALL	98.98
PEDEMO	ALL	ALL	ALL	83.08
SAMPLE	IBMDEFAULTBP	ALL	ALL	85.35
SAMPLE	ALL	ALL	ALL	85.35
TESTDB	IBMDEFAULTBP	ALL	ALL	82.39
TESTDB	ALL	ALL	ALL	82.39
ALL	ALL	ALL	ALL	83.49

Tabelle 7.1: Beginn der Analyse

Es stellt sich heraus, dass die BPHR des FRUITS-Bufferpool einen abnormalen Durchschnittswert aufweist. Aufgrund der vorliegenden Daten (siehe grau hinterlegte Zeile in Tabelle 7.1) entscheidet sich der DBA tiefer, auf Tablespace-Ebene, herunterzubrechen und die Ergebnisse mittels Ranking-Funktionalität nach den schlechtesten 10 BPHRs anzuordnen. Das Statement ähnelt dem vorherigen und liefert als Ergebnis die **Tabelle 7.2**. Prinzipiell hätte man die Ergebnismenge mittels einer Restriktionsbedingung auf den FRUITS-Bufferpool weiter einschränken können. Dieses und alle weiteren Statements können in den Skripten zur Arbeit [Wie05] nachvollzogen werden.

DB	BP	TBS	Zeit-Intervall	BPHR
PEDEMO	FRUITS	GROWTH	ALL	15.27
PEDEMO	FRUITS	TRADE	ALL	49.77
TESTDB	IBMDEFAULTBP	USERSPACE1	ALL	65.72
TESTDB	IBMDEFAULTBP	SYSCATSPACE	ALL	81.96
SAMPLE	IBMDEFAULTBP	SYSCATSPACE	ALL	85.02
PEDEMO	IBMDEFAULTBP	SYSCATSPACE	ALL	88.93

TESTDB	IBMDEFAULTBP	SYSTOOLSPACE	ALL	97.03
SAMPLE	IBMDEFAULTBP	SYSTOOLSPACE	ALL	98.75
PEDEMO	MOTION	VEHICLES	ALL	99.01
PEDEMO	BUILDINGS	LOCATIONS	ALL	99.64

Tabelle 7.2: Drill-Down und Ranking

Der DBA kann nun mittels Drill-Down (auf der Zeit-Dimension) einen höheren Detaillierungsgrad für den offensichtlich eine zu niedrige BPHR aufweisenden GROWTH-Tablespace der PEDEMO-Datenbank wählen. **Tabelle 7.3** zeigt das Resultat der entsprechenden Anfrage.

DB	BP	TBS	Zeit-Intervall	BPHR
PEDEMO	FRUITS	GROWTH	Month11/Day2	57.20
PEDEMO	FRUITS	GROWTH	Month11/Day2	2.12
PEDEMO	FRUITS	GROWTH	Month11/Day8	24.41
PEDEMO	FRUITS	GROWTH	Month11/Day8	33.58
PEDEMO	FRUITS	GROWTH	Month11/Day9	55.71
PEDEMO	FRUITS	GROWTH	Month11/Day9	14.09
PEDEMO	FRUITS	GROWTH	Month11/Day9	10.76
PEDEMO	FRUITS	GROWTH	Month12/Day14	12.52
PEDEMO	FRUITS	GROWTH	Month12/Day15	21.97
PEDEMO	FRUITS	TRADE	Month11/Day2	76.51
PEDEMO	FRUITS	TRADE	Month11/Day2	52.07
PEDEMO	FRUITS	TRADE	Month11/Day8	56.99
PEDEMO	FRUITS	TRADE	Month11/Day8	54.79
PEDEMO	FRUITS	TRADE	Month11/Day9	77.90
PEDEMO	FRUITS	TRADE	Month11/Day9	58.45
PEDEMO	FRUITS	TRADE	Month11/Day9	45.58
PEDEMO	FRUITS	TRADE	Month12/Day14	45.54
PEDEMO	FRUITS	TRADE	Month12/Day15	55.50

Tabelle 7.3: Weitergehendes Drill-Down

Durch das allmähliche Tiefersteigen in den einzelnen Hierarchien kann der DBA sowohl den Datenhorizont für die nachfolgenden Analysen beschränken, als auch den Detaillierungsgrad erhöhen. Basierend auf dem in Tabelle 7.3 vorliegenden Datenbestand erscheint es von Interesse, das exakte Zeit-Intervall einzugrenzen, in dem die Performance abzunehmen begann und somit einen weiteren Drill-Down vorzunehmen (siehe **Tabelle 7.4**).

DB	BP	TBS	Zeit-Intervall	BPHR
PEDEMO	FRUITS	GROWTH	Month11/Day2/12:45-13:00	65.99
PEDEMO	FRUITS	GROWTH	Month11/Day2/13:00-13:15	10.85
PEDEMO	FRUITS	GROWTH	Month11/Day2/13:15-13:30	10.50
PEDEMO	FRUITS	GROWTH	Month11/Day2/13:30-13:45	10.01
PEDEMO	FRUITS	GROWTH	Month11/Day2/13:45-14:00	8.98
PEDEMO	FRUITS	GROWTH	Month11/Day2/14:00-14:15	8.46
PEDEMO	FRUITS	GROWTH	Month11/Day2/14:15-14:30	8.22
PEDEMO	FRUITS	GROWTH	Month11/Day2/14:30-14:45	8.18
PEDEMO	FRUITS	GROWTH	Month11/Day2/14:45-15:00	7.97
PEDEMO	FRUITS	GROWTH	Month11/Day2/15:00-15:15	8.37

Tabelle 7.4: Bestimmung der Ursache des Problems

Mit Hilfe dieser Ergebnisse und der Inspektion der Konfigurations-Historie gelingt es dem DBA letztlich festzustellen, dass die Größe des FRUITS-Bufferpool zur Laufzeit um ca. 13:00 Uhr durch eine andere Person heruntergesetzt wurde. Von diesem Moment an, begann der rapide Abfall der BPHR und damit die einhergehende Beeinflussung der Performance. Mit diesem gewonnenen Wissen über die Problem-Ursache kann der DBA nun Maßnahmen einleiten um das Problem zu beheben oder den Ausgangszustand wiederherzustellen. So könnte er in diesem Fall u.a. die initiale Bufferpool-Größe wiederherstellen, mehr physischen Speicher zur Verfügung stellen oder einen neuen Bufferpool für den Zugriff auf spezifische Tabellen erzeugen.

Neben der Analyse innerhalb einer Cube-Struktur, können auch der Wechsel von Cube-Strukturen oder gar Cube-übergreifende Auswertungen von Belang sein. Angenommen, die Konfigurations-Historie bietet keinen Aufschluss über die Ursache des obigen Problems. Indem der DBA die anderen Fakten des genannten Cubes betrachtet, stellt er zudem fest, dass die Wartezeit auf Sperren deutlich angestiegen ist. Durch die Erkenntnisse dieser Analyse kann der DBA einen neuen Cube im Kontext der identifizierten, problembehafteten Datenbank-Ressourcen (oder auch der entsprechenden verantwortlichen Applikation bzw. des Nutzers) erzeugen. Die Auswertung der Daten dieses Cubes zeigt nun deutlich, dass die gewisse Applikation eine ungewöhnlich große Anzahl an Einfüge-Operationen auf einer spezifischen Tabelle innerhalb des besagten Tablespace ausführt. Durch die einhergehende große Zahl an Tupel-Sperren wird beträchtlicher Platz in der Sperrtabelle belegt. Die dadurch und aufgrund des zu gering adjustierten Konfigurationsparameters LOCKLIST auftretenden Sperr-Eskalationen sorgen für die unnötigen Wartezeiten. Eine logische Konsequenz der Analyse besteht demnach in der Vergrößerung der Sperrtabelle, sprich dem Speicherbereich für (Tupel-)-Sperren, um zukünftige Sperr-Eskalationen möglichst zu vermeiden.

Typischerweise werden bei Analysen (reine Zähler-)Statistiken über die Zeit und sämtliche Anfragen gemittelt, so dass es nicht ohne Weiteres möglich ist, zu bestimmen, welche Anfragen die meisten Ressourcen bzw. die meiste Wartezeit verursachen. Betrachten wir hierzu kurz eine Analogie mit dem Ziel der Verkürzung der Zeit bis zur Arbeitsstätte. Um das Ziel zu erreichen, würde man schlichtweg auch nicht (nur) die Temperatur des Motors oder die Zahl der Reifenumdrehungen messen. Maßgeblich sind vor allem die Dinge, welche die Zeit auf dem Weg und den einzelnen Etappen beeinflussen. Alle anderen Statistiken lenken eher von dem eigentlichen Problem (z.B. einer regelmäßig verstopften Kreuzung ab). Auch ein „blindes“ tägliches oder wöchentliches Vergleichen der verschiedenen Trips zur Arbeit, birgt nicht genug Detaillierungsgrad um die Situation zu verbessern. In jedem Falle spielen die Details eine herausragende Bedeutung: Wartezeiten an den einzelnen Ampeln, Staupotential und -zeiten auf den Strecken-Abschnitten oder auch Geschwindigkeits-Begrenzungen. Die multidimensionalen Analysen sollten sich daher vermehrt auf im Abschnitt 2.1.4 eingeführte Bottleneck-Analysen und damit auf ein Herunterbrechen sowie Bestimmen von Wartezeiten-verursachenden Teiloperationen bzw. Ressourcen konzentrieren. Das multidimensionale Datenmodell und insbesondere die Dimensionen sowie die Dimensionshierarchien genügen diesen Forderungen und bieten ausreichend Potential für Analysen dieser Art.

7.2.4 Automatismen bei der multidimensionalen Analyse

Um die Daten-Mengen handhabbarer werden zu lassen und um den Speicherbedarf zu optimieren, liegt eine automatische **Komprimierung** bzw. **Aggregation** älterer Daten nahe. In einem beweglichen Zeitfenster kann und sollte die Granularität auf dem detailliertesten Niveau gehalten werden, die restlichen älteren Daten können automatisch aggregiert und der Umfang der Daten geschrumpft werden. Vorstellbar ist es bspw. die feingranularen Daten (auf Minuten-Basis) über einen Monat aufzuheben. Sämtliche Daten, die älter als dieses bewegliche Zeitfenster von einem Monat sind, werden etwa auf Stunden-Basis aggregiert. Für eine Analyse der Daten über den Zeitraum eines ganzen Jahres genügt wohl eine Genauigkeit der (gruppierten) Datensätze von einem Tag.

Der in Abschnitt 6.5 eingeführte Cube Advisor ermöglicht zusammen mit einer Event-Erkennungs-Komponente eine eigenständige Reaktion auf automatisch detektierte Performance-Probleme, indem entsprechende, für das Problem relevante Cube-Strukturen

aus einer vordefinierten Menge **selektiert**, **erzeugt**, **gefüllt** und dem Nutzer (ggf. voraggregiert materialisiert) präsentiert werden (können). Die Integration und Anwendung dieser Komponente in einer umfassenden, prototypischen Monitoring- und Tuning-Architektur im Umfeld des Autonomic Database Performance Tuning erfolgt im Abschnitt 8.4.

Es ist nun naheliegend, an diesen automatisierten Prozess mit einer ebenso automatischen Analyse anzuschließen. Die Autoren von [HHY99] beschreiben einen interessanten Ansatz sowie einen ersten Prototypen zur Automatisierung der Problem-Diagnose auf multidimensionalen Daten. Mit Hilfe einer Scoring-Funktion auf Ebene der Cubes sowie auf Ebene der Dimensionen und anhand von Schwellwerten entscheidet der Algorithmus automatisch, welcher Cube bzw. welche Dimension das Problem am besten charakterisiert. Eine Folge von **automatischen Drill-Down**-Operationen erhöht schrittweise den Detailgrad auf den Daten mit dem geringsten oder von einem nutzerdefinierbaren bzw. durch eine Policy vorgegebenen Schwellwert bestimmbaren Abstand zum Minimum bzw. Maximum einer Menge von (aggregierten) Metrik-Werten. Die Administratoren müssen jedoch noch immer eine (Vor-)Auswahl treffen, welche Cubes bzw. Dimensionen das erkannte Problem am adäquatesten beschreiben. Inwieweit diese Konzepte bei uns Anwendung finden und zusammen mit dem Event-basierten Cube Advisor kombiniert werden können, obliegt weitergehenden Untersuchungen.

Vor allem der zusätzliche Einsatz nachfolgend dargestellter Data-Mining-Algorithmen mit dem Ziel der Informationsgewinnung zur automatisierten Entscheidungs-Unterstützung birgt großes Potential. Der Grundgedanke ist dabei, ein System anhand von Beispielen auf den durch die Sensoren bereitgestellten sowie die im Tuning-Prozess anfallenden Daten lernen und anschließend verallgemeinern zu lassen. Durch erkannte Gesetzmäßigkeiten in den Lerndaten kann das System, z.B. durch logisches Schließen, auch unbekannte Daten beurteilen.

Aber auch wenn die Analyse nicht in allen Fällen vollständig automatisierbar ist, so ist bereits viel gewonnen, dem Nutzer zumindest eine verlässliche Vorauswahl an Cubes zu präsentieren und damit den Datenraum zusammen mit dem Aufwand zu verringern.

7.3 Data Mining im Kontext des autonomen Datenbank-Tuning

Neben der Problem- sowie Ursachen-Erkennung und -Auflösung liegt ein weiterer Nutzen der in dem PWH verwalteten Performance-Daten in der Bestimmung bisher unbekannter Informationen (Korrelationen, Muster, Trends, Anomalien, …). Der Mensch alleine ist dazu allerdings nicht immer in der Lage. Techniken des Data Mining versprechen durch unterschiedlichste Algorithmen zur Auswertung großer Datenbestände und zum logischen Schließen das (teil-)automatisierte Aufspüren derartiger geltender Regeln, Ursache-Wirkungs-Zusammenhänge, Muster und statistischer Auffälligkeiten. Die dafür verwendeten Methoden können sehr unterschiedlich sein und wurden bereits ausreichend in Abschnitt 2.4.2 eingeführt.

Das Mining soll jedoch nicht nur auf den „normalen“, aufgezeichneten Performance-Daten über das System und die anliegende Workload, sondern vielmehr auch auf den im und über den Tuning-Prozess anfallenden Ausführungsdaten (z.B. die Historie der erkannten Probleme und daraufhin angestoßenen Maßnahmen) erfolgen. Dies befähigt zur Generierung von neuem Wissen über das operative und das Tuning-System.

Im Grunde lassen sich nach [Böt08, Göb08] beinahe sämtliche bekannte Methoden des Data Mining im Kontext des (autonomen) Performance Tuning für die Entscheidungsunterstützung anwenden und wie folgt gruppieren:

- **Analyse und Erklärung**
  - *Daten-Vorverarbeitung*: Hierunter fallen Mechanismen zur initialen Behandlung der Daten mit dem Ziel der Datenselektion/-filterung/-reduktion. Als Beispiel sei die Korrelation von Spalten und damit die Reduktion des Inputs für die anschließende Klassifikation zu nennen. Eine Reduktion der Variablen-Anzahl lässt sich ferner realisieren, indem man anstelle der einzelnen Beobachtungen Ähnlichkeits- und Distanzmaße zwischen Beobachtungen als Analysegrundlage benutzt.
  - *Abweichungs-Erkennung:* Im Sinne eines Frühwarnsystems kann man (bspw. anhand einer Korridor-Analyse) Ausreißer im Datenverlauf, sprich sich andeutende Probleme im Voraus erkennen um diese als Problemkandidaten zu untersuchen, inakkurate Schwellwerte anzupassen, neue Events zu definieren oder automatische Aktionen einzuleiten. Wichtig ist hierbei, nicht jeden Ausreißer als Abnormalität zu betrachten. Es sind durchaus Szenarien denkbar, in denen im Rahmen einer Vorverarbeitung der Datenbestand um atypische Beobachtungen bereinigt wird. Dadurch produzieren die Verfahren zum einen keine Artefakte in Form von nicht im Datenbestand existierenden Effekten und können sich auf die wesentlichen charakteristischen Merkmale des Datenbestands konzentrieren. Auch Daten, die nie angefasst oder kontinuierlich geprüfte, aber nie aufgetretene Events können identifiziert und entsprechend adaptiert werden. Abweichungen sind darüber hinaus potentiell auch bei der Betrachung von (Policy-basierten) Soll- und gemessenen Ist-Werten ermittelbar.
  - *Muster-Erkennung*: Retrospektive Analysen auf der Suche nach Auffälligkeiten in den (historischen) Daten oder auch zur Erkennung von ähnlichen Situationen.
  - *Ursachen-Analyse:* Beschreibung des direkten bzw. indirekten Zustandekommens bestimmter Sensor- bzw. Effektor-Werte oder auch der Ursachen von Performance- und deren Folge-Problemen. Ursache-Wirkungsketten können mittels statistischer Analyse identifiziert werden.
  - *Regression/Assoziation:* Erklärung und Aufdecken von Zusammenhangsstrukturen, Abhängigkeiten, Auffälligkeiten und Korrelationen. Beispielhaft zu nennen sind quantitative Abhängigkeiten zwischen Tuning-Plänen bzw. Effektoren und Sensoren (siehe Abschnitt 7.5). Assoziationsregeln können beispielsweise auch zum automatischen Bestimmen von Problem-Situationen oder gar zur automatischen Bestimmung von Korrelation von Ereignissen verwendet werden. Dabei kann untersucht werden, welche möglichen Ereignisse oftmals in Kombination auftreten. Dadurch ist es möglich, nutzvolle Erkenntnisse über die Ursache von Performance-Verschlechterungen zu erhalten. Stellt man in der Event-Historie bspw. fest, dass das Event E1 immer im Cluster mit den Events E2 und E3 auftritt, lässt auf eine mögliche Korrelation (und ein zusammenzusetzendes Event) schließen. Ebenso lassen sich mit Hilfe eines Ressourcen-Abhängigkeits-Modells konfligierende Tuning-Aktionen bzw. -Ziele erkennen und schließlich zur Ausführungszeit berücksichtigen.

  - *Klassifikation* (Diskriminanz-Analyse): Der naheliegende Einsatz der Klassifikation liegt in der Erkennung der Workload bzw. Einordnung in eine von mehreren vordefinierten Workload-Klassen. Sie ist vor allem sinnvoll bei im Laufe der Zeit wechselnden Workload-Charakteristika. Hierbei wird eine bereits klassifizierte Menge an Testdaten verwendet, um einen Klassifikator zu erstellen, dessen Aufgabe es ist, den aktuellen Workload-Typ eines Datenbanksystems zu bestimmen (siehe Abschnitt 7.4). Mit Hilfe dieser gewonnenen Informationen können u.a. die Systemparameter an den aktuell vorherrschenden Workload-Typ angepasst werden. Alternativ können Performance-Probleme klassifiziert werden. Eine solche Einordnung kann wiederum in die Cube-Advisor-Architektur integriert werden und zur Analysezeit die Menge an relevanten Cubes einschränken.
  - *Segmentation* (Cluster-Analyse): Anwendungen des Clustering finden sich in der Charakterisierung von komplexen Zuständen oder auch in der Komprimierung der zu betrachtenden Daten bzw. Workload-Klassen. Clustering-Algorithmen können zudem eingesetzt werden, um beispielsweise SQL-Anfragen zu kategorisieren. So können Anfragen, die auf die gleichen Tabellen zugreifen zu einer Klasse zusammengefasst werden. Damit ist es möglich, häufig abgefragte Daten mit einem zusätzlichen Index zu versehen oder die Bufferpool-Konfiguration entsprechend anzupassen, damit die entsprechenden Anfragen möglichst schnell abgearbeitet werden können.
  - *Modell-Bildung:* Die Ergebnisse der Verfahren zur Analyse und Erklärung werden in der Regel mathematisch, in ein Modell abgebildet und kontinuierlich angepasst bzw. verbessert. Nachfolgend stellen wir zwei konkrete Modelle zur Beschreibung der Workload und der Abhängigkeiten zwischen den Ressourcen bzw. der Sensoren und Effektoren des operativen Systems dar.

- Beobachten und **Lernen:** Ein kontinuierliches Beobachten, Analysieren und Lernen des normalen Zustands bzw. Verhaltens („Baseline“) kann mit dem Augenmerk auf eine (automatische) Reaktion bei aktuellen oder berechneten Abweichungen durchgeführt werden. Das Lernen muss als inkrementeller Prozess begriffen werden, bei dem aktuelle Entscheidungen verzögerte Konsequenzen haben können (z.B. Latenz-Zeit der Tuning-Aktionen). Eine zeitnahe, adaptive Anpassung ermöglicht es, von den Konsequenzen der eigenen Entscheidungen zu lernen.
- **Vorhersage** (Wirkungsprognose): Auf den gesammelten, historischen Performance-Daten ausgeführte *Trend-* und Zeitreihen-Analysen können proaktives Handeln unterstützen. Die der Klassifikation ähnliche Regressionsanalyse ist bspw. die typische Vertreterin der statistischen Verfahren zur Formalisierung von solchen Wirkungs-Zusammenhängen und zur Bestimmung von (künftigen) Zielwerten. Anders als bei der Klassifikation sind diese Zielwerte jedoch quantitativer Natur. Dies kann nützlich sein, um den Verlauf von Performance-Metriken abzuschätzen und frühzeitig (unerwünschte) Änderungen zu erkennen. Durch den damit verbundenen Zeitgewinn ist man proaktiv in der Lage adäquater zu (re)agieren und frühzeitig weitere Monitoring-, Analyse- oder Simulations-Schritte einzuleiten, um eine möglichst optimale Performance sicherzustellen. Inwieweit man in der Lage

ist, auf eine Problem-Situation oder eine vorhergesagte Workload zu reagieren, die noch gar nicht existiert, soll zunächst nicht näher ausgeführt werden.

- **Kontrolle** und Feedback: Die Evaluation bzw. Verifikation der Ausführung bzw. Effizienz von Tuning-Plänen will bspw. ergründen, ob Tuning-Aktionen korrekt in Folge eines Events ausgelöst und ob sie den versprochenen Erfolg oder gar zusätzliche Seiten-Effekte ausgelöst haben.

Die meisten der Anwendungen im Performance-Monitoring- und -Tuning-Kontext lassen sich durch mehrere oder auch eine Kombination der vorgestellten Methoden realisieren. Auch die Techniken sind oft nicht disjunkt bzw. in Isolation zu betrachten, insbesondere dann, wenn sie einander zwingend bzw. aus Optimierungsgründen voraussetzen. Beispielsweise erscheint einer Ausreißer-Elimination vor einer eingehenden Muster- bzw. Trend-Erkennung als sinnvoll.

Data-Mining-Methoden können zusammenfassend eingesetzt werden, um die Administratoren bei Entscheidungen zu unterstützen, aber auch, um den Grad der Autonomie in einem **MAPE-Regelkreis** durch komplexe Analysen, selbstständiges Folgern und Automatisierung grundlegender Abläufe zu erhöhen. In [Göb08] und [Böt08] zeigen wir am Beispiel des IBM Intelligent Miner [IBM06d] wie ein konkretes Data-Mining-Tool für diese Entscheidungs-unterstützenden Zwecke eingesetzt werden kann und welche verfügbaren Funktionen für die entsprechenden MAPE-Phasen von Bedeutung sind. Wir wollen an dieser Stelle jedoch lediglich die zugrundeliegenden Konzepte aufzeigen.

Da das Sammeln, Einbinden und Filtern der Performance-Daten im Rahmen der **Monitoring**-Phase aufwendig und kostspielig ist, in der Regel jedoch nur ein Bruchteil derer benötigt wird, sind Methoden wichtig, die den Nutzen der Daten für die nachfolgenden Schritte bzw. das Performance-Tuning im Allgemeinen einschätzen können. So sollen bspw. Spalten in Tabellen, deren Informationsgehalt für die weitere Arbeit von Bedeutung sein könnte sowie Spalten, die stark mit anderen Spalten korrelieren, keinen weiteren Informationsgewinn besitzen und somit in der Regel nicht benötigt werden, erfasst und gesondert behandelt werden. Die Resultate können behilflich sein bei der Optimierung des Datenbank-Designs und der Monitoring-Phase, in welcher folglich weniger Daten gespeichert, gefiltert und geprüft werden müssen.

Zusätzlich lassen sich die Performance-Daten durch eine Einteilung in Gruppen (Cluster) und der anschließenden Betrachtung der Cluster-Merkmale statt der Rohdaten-Merkmale komprimieren. Datensätze, die durch unerwartete Werte bzw. Auffälligkeiten stark von zugehörigen Cluster-Merkmalen abweichen (Ausreißer), können gezielt auf Symptome geprüft oder bewusst eliminiert werden. Daneben können Data-Mining-Methoden auch einzelne Felder einer Tabelle identifizieren, die (im Werteverlauf) von den restlichen Daten abweichen. Dieses Bereinigen des Datenbestands von nicht repräsentativen, atypischen Effekten, erlaubt die Konzentration auf die wesentlichen, charakteristischen Merkmale.

Die sich der Monitoring-Phase anschließende **Analyse** (erkannter Symptome) kann von Prognosen über das zukünftige Systemverhalten (Trend-Erkennung), auf Basis historischer und aktueller (Performance-)Daten, profitieren. Das Teilgebiet des Temporal Data Mining [ChSt98a] bietet dafür Methoden zur Analyse von zeitlichen Zusammenhängen innerhalb einer oder zwischen mehreren Zeitreihen. Somit ist es möglich, kritische Situationen vor ihrem Eintreten zu erkennen und frühzeitig vorsorgende Maßnahmen einzuleiten. Die Folge derartiger „Trend Events“ können bspw. das Anstoßen von Monitor-Strategien zur

Sammlung erweiterter Daten oder auch ein Tuning mit einfachen, „leichtgängigen“ Mitteln sein.

Um die Menge an zu betrachtenden Ereignissen einzuschränken und um weitergehende Erkenntnisse über die Ursache(n) von Performance-Problemen zu erlangen, kann Data Mining auch zur Aufdeckung bzw. Untersuchung möglicher Symptome bzw. Ereignisse verwendet werden, die oftmals in Kombination auftreten bzw. in irgendeiner Form korrelieren.

Die **Plan**-Phase der MAPE-Schleife bestimmt das in Folge eines durch ein (korreliertes) Event erkannten Problems optimale Vorgehen zur Auflösung bzw. Verbesserung der vorliegenden kritischen Situation. Dafür ermittelt es aus einer vordefinierten Menge die in Frage kommenden Tuning-Pläne und leitet sie an die Execute-Phase weiter. Die Zuweisung bzw. die Reihenfolge der Tuning-Pläne kann dabei von dem Wissen über den (Miß-)Erfolg vorheriger Tuning-Plan-Ausführungen profitieren. Aus diesem Grund sollten, wie bereits angedeutet, neben den reinen Performance-Daten auch Laufzeitdaten über das Tuning-System gesammelt werden.

Tuning-Pläne nehmen über die Effektor-Schnittstellen in der Regel einen ändernden Einfluss auf die Ressourcen des zu tunenden Systems vor. Optimal für die Plan-Phase ist daher eine Möglichkeit zur Bestimmung des (qualitativen bzw. quantitativen) Einflusses von Effektoren (z.B. Konfigurationsparameter) auf den Verlauf von Sensoren (z.B. Performance-Metriken) und damit auch auf den Systemzustand. Umgekehrt wäre es denkbar den Wert einer (kritischen) Metrik durch eine Kombination von Konfigurationsparameter-Werten zu erklären. So können das Wiederholen erfolgloser Tuning-Pläne verhindert und suboptimale Tuning-Pläne verbessert werden. Derartige Methoden sind nicht nur zur Kontrolle geeignet, sondern können auch zur (Ad-hoc-)-Adaption bestehender bzw. Ermittlung und Erzeugung neuer Tuning-Pläne führen (siehe dazu auch die Ausführungen im Abschnitt 4.4.1). Wir greifen im kommenden Abschnitt 7.5 diese Idee im Rahmen eines Ressourcen-Abhängigkeits-Modells auf, beschreiben die Ergebnisse unserer Arbeiten in diesem Kontext [Exn07, Gan06, RaWi07, WiRa09] und präsentieren Anknüpfungsmöglichkeiten für das Data Mining. Die Versuchsergebnisse zeigen, dass sowohl die Erstellung von Assoziations-Modellen als auch das Bestimmen von Korrelations-Koeffizienten geeignet ist, Zusammenhänge von Effektoren auf Sensoren und unter Sensoren zu erkennen.

Während der Großteil der (OLAP-)Analysen auf dem PWH eher kurzfristiger Natur ist und in die Analyze-Phase der MAPE-Schleife (Erkennen und Korrelieren von Events) eingeordnet werden kann, sind Data-Mining-Methoden zumeist mittel- bis langfristiger Natur. Wir haben uns zur Unterstützung unseres in Abschnitt 8.3 vorgestellten, auf dem MAPE-Paradigma basierenden Autonomic Tuning Expert, zunächst für eine Workload-Klassifikation zur Erkennung der momentanen Systemlast auf der Datenbank und für ein Ressourcen-Modell zur Beschreibung der Abhängigkeiten zwischen den Effektoren und Sensoren der Ressourcen entschieden.

Bereits in den Arbeiten [Böt08, Göb08, Kra08] haben wir am Beispiel von DB2 sowohl theoretische als auch praktische Untersuchungen zur Anwendbarkeit und Übertragbarkeit von Data-Mining-Analyse-Techniken auf Performance-Daten durchgeführt. Insbesondere in [Göb08] wurden die Möglichkeiten des IBM Intelligent Miner aufgezeigt und Konzepte entwickelt, um das autonome Datenbank-Tuning zu unterstützen. Intelligent Miner bietet dank einer Vielzahl von Techniken und Werkzeugen, benutzerfreundlicher Handhabbarkeit sowie visuelle Unterstützung in Form des Design Studios und IM Visualization eine geeignete Umgebung, um Data-Mining-Funktionalitäten in Anwendungen zu integrieren.

Nach der Evaluierung verschiedener Konzepte wurden mit der Workload-Klassifikation und der Korrelationsbestimmung von Konfigurationsparametern und Metriken zwei wesentliche Konzepte prototypisch umgesetzt und erzielte Ergebnisse hinsichtlich ihrer Einsatzmöglichkeiten bewertet. Die wesentlichen Ergebnisse werden im weiteren Verlauf der Arbeit veranschaulicht.

7.4 Ein Workload-Modell zur System- und Tuning-unabhängigen Workload-Erkennung

Dank universeller, in einen vielschichtigen Software-Stack eingebundener DBMS können Organisationen (beinahe) die gesamte Breite ihrer Geschäftsbedürfnisse mit einem technologischen Fundament abdecken [Eln04, Li08]. Die Systeme werden daher während des normalen Betriebs zunehmend wechselnde Arbeitslasten (Workloads) ausgesetzt.

Die **Workload** definiert sich als eine Menge von Anfragen, Transaktionen, Skripten, Kommandos oder Jobs mit verschiedensten Anforderungen an die Ressourcen. Sie ist im Allgemeinen nicht änder- bzw. kontrollierbar und ständigen Änderungen ausgesetzt. Dennoch kann sie bzw. deren Charakteristika über Sensoren ausgelesen bzw. überwacht werden.

Die Art der Workload eines DBMS und die durch sie verursachte Ressourcen-Nutzung ist ein entscheidender Punkt für das Tuning seiner Performance [Eln04]. Die Konfiguration sollte daher stets an den aktuellen (oder auch erwarteten) Workload-Typ angepasst werden. Wird hingegen auf eine zeitnahe Anpassung verzichtet, so stellt die Konfiguration stets nur einen Kompromiss aller möglichen Situationen dar, die über die Zeit möglich sind. Dies kann zu einer längeren Verarbeitungszeit von Anfragen führen und nutzt die vorhandenen Ressourcen des Systems nicht optimal.

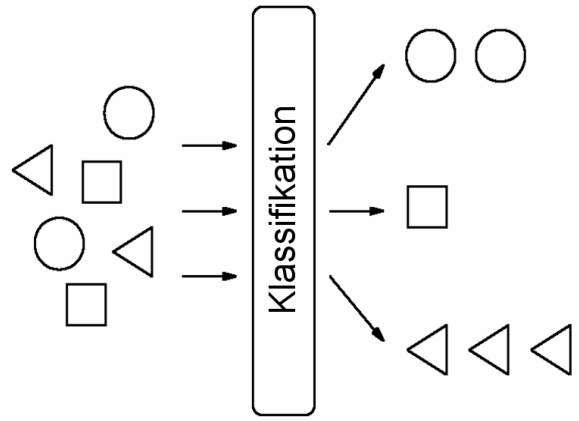


Abbildung 7.3: Schematische Veranschaulichung der (Workload-)Klassifikation

Um ein Workload-orientiertes Performance-Tuning zu unterstützen, ist daher ein intelligenter Mechanismus zur Workload-Erkennung hilfreich. Eine Modell-basierte **Workload-Klassifikation** soll nunmehr gewährleisten, dass sowohl bei der (automatischen) Problem-Erkennung als auch bei der Problemlösung zur Laufzeit die aktuell identifizierte Workload adäquat berücksichtigt wird [Göb08, RAWR08].

Dieser Ansatz kann mit Hilfe von Data-Mining-Techniken implementiert werden. Der Name Workload-Klassifikation deutet auf die Nutzung des Klassifikations-Algorithmus hin, um anhand von Datenbank-Informationen die aktuelle Workload einer Datenbank zuvor definierten Klassen zuzuweisen (siehe **Abbildung 7.3**).

In den nachfolgenden Unterabschnitten werden Konzepte, zugrundeliegende Techniken sowie erste Ergebnisse unserer auf den Arbeiten [Böt08, Göb08, Kra08, RAWR08, WiRa07, WRRA08, WRRA09] basierenden Workload-Klassifikation erläutert. Die ausführliche Beschreibung der eigentlichen Integration in unseren Prototypen zum autonomen Datenbank-Tuning und die damit verbundene Anreicherung der MAPE-Phasen um Workload-Wissen lassen sich im Abschnitt 8.3 nachlesen.

7.4.1 Anforderungen

Die Workload-Erkennung sollte parallel zum laufenden Betrieb des jeweiligen Systems möglich sein und in ihren Berechnungen die aktuellen, Geschäfts-kritischen Anfragen nicht merklich behindern. Die aktuell anliegende System-Workload und mögliche Änderungen sollten dabei schnell und verlässlich erkannt sowie unmittelbar allen relevanten Tuning-Komponenten für die weitere Behandlung zur Verfügung gestellt werden. Für ein autonomes System ermöglicht dies eine **Workload-spezifische Problem-Erkennung** sowie -Auflösung zur Laufzeit und damit die Möglichkeit, frühzeitig auf die Veränderungen zu reagieren und so das System optimal an die wechselnden Anforderungen einzustellen.

Bereits in [Eln04] beschreibt der Autor eine System-unabhängige Workload-Erkennung für DB2 mit Hilfe von Data-Mining-Techniken. Dabei bleibt jedoch der Einfluss des (autonomen) Tunings auf die Workload-Erkennung unberücksichtigt. Wir greifen diesen Ansatz grundlegend auf, erweitern ihn jedoch, um sowohl eine System- als auch Tuning-unabhängige Workload-Erkennung zu erzielen.

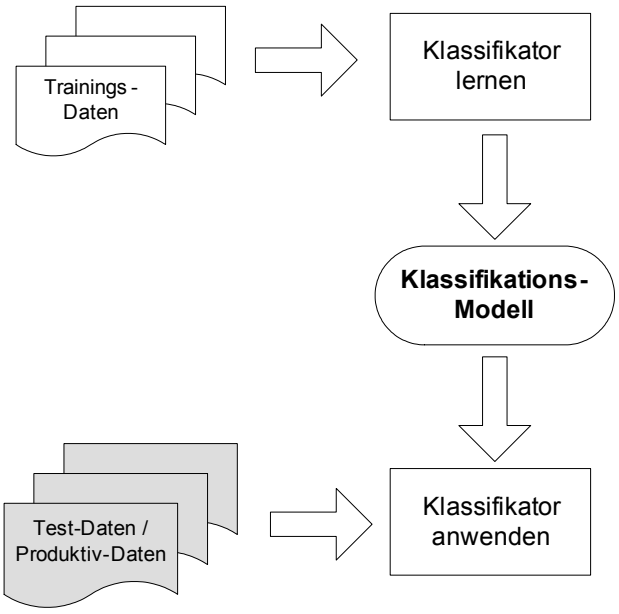


Abbildung 7.4: Funktionalitäten des Klassifikations-Frameworks in der Übersicht

Das generische, in den Autonomic Tuning Expert (vgl. Abschnitt 8.3) integrierte **Klassifikations-Framework**, welches im Grundsatz auch in dem Patent-Antrag [RAWR08] beschrieben wird, basiert auf der IBM Intelligent-Miner-Produktfamilie [IBM06h, IBM06d] und bietet die folgenden Funktionalitäten:

1. Definition von Klassifikations-relevanten Workload-Charakteristika und deren Zuordnung zu den zu erstellenden Klassifikations-Modellen,
2. Daten-Sammlung und Vorverarbeitung für die Klassifikations-Modelle,
3. Erlernen von Klassifikations-Modellen sowie

4. Anwenden von Klassifikations-Modellen zur Identifikation von unbekannten Workloads.

Die folgenden Abschnitte bieten einen kurzen Einblick in durch die **Abbildung 7.4** vergegenständlichten Funktionsweisen einzelner Bereiche des Klassifikations-Frameworks.

7.4.2 Auswahl klassifikationsrelevanter Metriken

Die **Klassifikation** ist der Prozess der automatischen Erstellung eines Modells aus einer Reihe von Datensätzen, deren Klassenzugehörigkeit bekannt ist. Das Klassifikations-Verfahren analysiert diese so genannten Trainings-Datensätze und erstellt ein Profil für die Merkmale von Datensätzen aller vorhandenen Klassen. Anschließend erlaubt das Klassifikations-Modell die Klasse unklassifizierter Datensätze vorherzusagen.

In unserem Ansatz sollen durch das PWH bereitgestellte (Snapshot-)Performance-Metriken Merkmale und das Verhalten der aktuellen Workload auf der Datenbank widerspiegeln. Die Herausforderung ist dabei, einerseits Metriken zu finden, die bei der Unterscheidung der zu unterstützenden Workload-Typen von Bedeutung sind. Andererseits spielt im Kontext des autonomen Performance-Tunings der Aspekt der *System-* und vor allem *Tuning-Unabhängigkeit* der Metriken eine große Rolle.

Elnaffar beschäftigt sich in seinen Arbeiten mit der Identifikation von DB2-Metriken, die von der eigentlichen Hardware-Konfiguration des Systems nicht abhängen (System-unabhängige Metriken) [Eln04]. Die auf solchen Metriken basierenden Klassifikations-Modelle können somit auf einem beliebigen Zielsystem zur Klassifikation der Workload eingesetzt werden. Eine Optimierung des Systems im laufenden Betrieb macht jedoch die Tuning-Unabhängigkeit der zur Bildung des Klassifikations-Modells verwendeten Metriken unabdingbar. Während beispielsweise die Anzahl der dynamischen SQL-Anweisungen vom Datenbankdurchsatz und somit vom Tuning abhängig und daher zur Klassifikation im Tuning-Betrieb nicht geeignet ist, hängt der Anteil der Select-Statements bezüglich aller abgesetzten Anweisungen lediglich von der Workload ab und kann von einem Tuning des Systems nicht beeinflusst werden.

Metrik	System-Abhängigkeit	Tuning-Abhängigkeit	Beschreibung
THROUGHPUT	Hoch	Hoch	Anzahl der erfolgreich ausgeführten SQL-Anweisungen
UID_SQL _STMTS	Hoch	Hoch	Anzahl aller SQL-Anweisungen zum Ändern, Löschen und Einfügen von Tupeln (Update, Insert, Delete)
LOCKS_HELD	Hoch	Mittel	Die Anzahl der momentan gehaltenen Sperren
AVG_SORT_TIME	Hoch	Gering	Durchschnittliche Sortierdauer
TOTAL_SORTS	Mittel	Hoch	Anzahl aller ausgeführten Sortierungen
AVG_NUMBER_-ROWS_SELECTED_ROW	Gering	Gering	Durchschnittliche Anzahl der Zeilen, die ausgewählt und an die Anwendung zurückgegeben wurden
RATIO_UID_ TOTOT_STMT	Gering	Gering	Anteil der Änderungen, Löschungen und Einfügungen von Tupeln bezüglich aller dynamischen und statischen SQL-Anweisungen an ein System

Tabelle 7.5: Übersicht geeigneter Metriken zur Workload-Klassifikation (Auszug aus [Göb08])

Das PWH kann eine Vielzahl unterschiedlicher Metriken bereitstellen. Für die Erstellung von Workload-Modellen sollte man sich jedoch auf eine Auswahl an zu klassifizierenden

Elementen beschränken, die relevant, leicht zugänglich sowie System- und Tuning-unabhängig sind. **Tabelle 7.5** fasst eine Auswahl relevanter, nach ihrer System-Abhängigkeit geordneter Metriken zusammen. Die Informationen beruhen auf eigenen Erfahrungen und teilweise auf Ausführungen in [Eln04].

Man kann sich demnach zur Workload-Klassifikation nun auf Tuning-unabhängige Metriken beschränken, auf die sich Änderungen der System-Einstellungen und das Tuning an sich, also z.B. die Änderungen von Konfigurationsparametern oder die Anpassung der Bufferpool-Größe, nicht bzw. nur unwesentlich auswirken. Alternativ dazu könnte man durch *Transformationen*, wie z.B. die Division durch die Anzahl der erfolgreich ausgeführten SQL-Anweisungen (Datenbank-Durchsatz), den Einfluss von Performance-Tuning oder anderen Systemkonfigurationen aus den Metriken eliminieren (bspw. bei Metriken wie der Anzahl an dynamischen SQL-Anweisungen oder der geschriebenen Log-Seiten).

7.4.3 Daten-Vorverarbeitung

Um ein adäquates Abbild der Workload zu schaffen und damit zeitnah auf Veränderungen reagieren zu können, setzen wir in unseren Untersuchungen ein minütliches PWH-Intervall zum Auslesen und Speichern der Metrik-Werte voraus. In DB2 stellen Elemente der Typen *Counter* und *Time* seit Monitoring-Start aufsummierte Werte dar und vermischen somit Werte des aktuellen Snapshot-Intervalls mit vorherigen. Metriken dieser Typen, die ausschließlich erhöht werden, müssen **normalisiert** werden, um von der Workload-Klassifikation genutzt werden zu können. Wir sind zur Bestimmung der aktuellen Workload lediglich an der Charakteristik der Metrik innerhalb des letzten Zeitintervalls, d.h. der Differenz zwischen zwei aufeinander folgenden Snapshots, interessiert. Im Gegensatz dazu repräsentieren *Gauges* (Pegel) stets aktuelle Werte, die insbesondere auch mittels Formeln aus Snapshot-Elementen (ggf. anderen Typs) abgeleitet sein können.

Wir bestimmen daher in einem Vorverarbeitungs-Schritt zunächst für jeden neuen, im PWH gespeicherten Snapshot die **Differenz** zum vorherigen und speichern diese in einer entsprechenden Tabelle ab. Man könnte diese Differenz einer Counter-Metrik zur vorherigen beispielsweise mit einem Insert-Trigger auf den entsprechenden Strukturen des PWH bei jedem Einfügen automatisch ermitteln lassen. Wir haben uns jedoch für eine Umsetzung mittels Sichten entschieden, auf welche die entsprechenden Komponenten später zugreifen. Für alle Metriken werden Metadaten über ihren Typ (Counter, Gauge etc.) und im Falle von zusammengesetzten Metriken auch die Berechnungsvorschrift gespeichert. Die erste der beiden Sichten basiert auf den PWH-(Fakten-)Tabellen und ist für die Delta-Bildung entsprechender Metriken verantwortlich. In unseren Untersuchungen zeigen wir auf, wie die Delta-Bildung mit reinem SQL und der ausschließlichen Verwendung arithmetischer Operationen über Partitionen des Metrikverlaufs in einem beweglichen Zeitfenster verwirklicht werden kann. In einer zweiten, darauf aufbauenden Sicht, können schließlich die Berechnungsvorschriften auf die bereits einer Differenzbildung unterzogenen Metriken angewendet werden. Die Sichten wiederum werden anhand der Metadaten über die Metriken automatisch generiert und aktualisiert, so dass die Workload-Erkennungs- und später auch die Visualisierungs-Komponente von zusätzlichem, initialen Aufwand „verschont“ bleiben.

Dabei erfolgt eine weitere **Normierung** auf eine konstante Intervall-Länge, da nicht davon ausgegangen werden kann, dass zwischen zwei Snapshots immer dieselbe Zeitspanne liegt. Die normierten Daten können nun sowohl zur Modell-Erzeugung, zur Klassifikation als auch zur Visualisierung der Tuning-Ergebnisse verwendet werden.

7.4.4 Modell-Erzeugung

Zur Erstellung eines repräsentativen **Workload-Modells** (im Folgenden auch als **Classifier** bezeichnet) werden nacheinander die später zu erkennenden Workloads auf dem System simuliert. Eine zeitgleich und periodisch durch den PE gesammelte Auswahl an Metriken wird, wie in Abschnitt 7.4.3 beschrieben, von Fehlern bereinigt, normiert und anschließend mit dem Namen der jeweiligen Workload in einer Tabelle gespeichert. Vorzugsweise kann hierbei ein Teil der gesammelten Daten zum Erlernen und der andere Teil zur Validierung (Modell-Überprüfung) genutzt werden.

Nachdem dies für alle gewünschten Workload-Typen durchgeführt wurde, analysiert ein vom IBM DB2 Intelligent Miner Modeling zur Verfügung gestelltes **Entscheidungsbaum**-basiertes Klassifikationsverfahren diese Trainings-Datensätze und erstellt ein Profil für ihre Merkmale. Das Ergebnis des gewählten und auf die Eingabedaten angewandten Algorithmus ist ein Workload-Modell, was nach optionaler Prüfung auf Korrektheit visualisiert oder zur Klassifikation, d.h. zur Erkennung unbekannter Workloads, genutzt werden kann. Die Klassifikation wird im folgenden Abschnitt beschrieben.

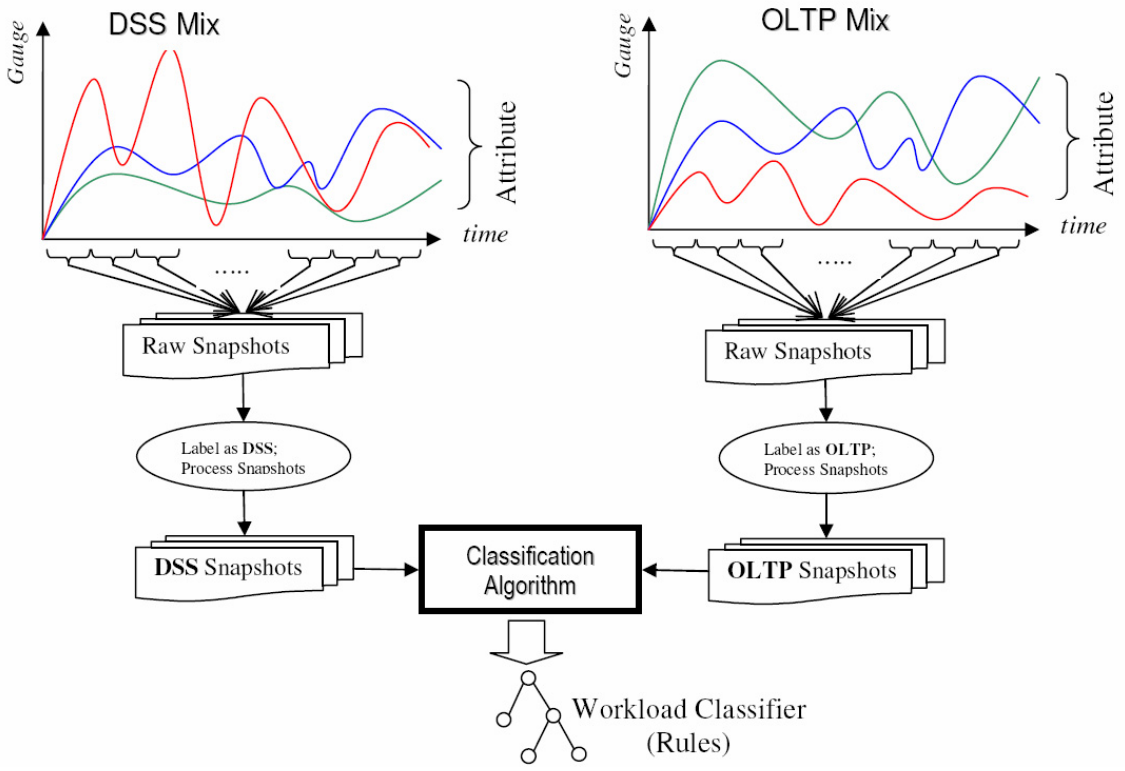


Abbildung 7.5: Die Workload-Classifier-Erzeugung in der Übersicht (nach [Eln04])

Abbildung 7.5 veranschaulicht den beschriebenen Klassifikations-Ansatz zur Erzeugung eines Workload-Modells für zwei nacheinander auf dem System ausgeführte und in Form von Snapshots erfasste Workloads.

Wir übergeben dem Intelligent Miner der Einfachheit halber alle im PWH verfügbaren Metriken. Eine Beschränkung bringt an dieser Stelle keinen Performance-Gewinn, sofern das PWH sämtliche Metriken bereitstellt. Durch eine autonome Variablenwahl des Intelligent Miner Modeling werden die zur Modell-Erzeugung benötigten Metriken automatisch gewählt. Mittels der Auswerte-Möglichkeiten des Intelligent Miner

Visualization können die ermittelten Relevanzen der Metriken für das Workload-Modell zusätzlich in einem Feldbedeutungsdiagramm dargestellt werden (siehe dazu [Göb08]).

7.4.5 Workload-Erkennung mittels Klassifikation

Die Klassifikation ähnelt der Modell-Erzeugungs-Phase und wird in **Abbildung 7.6** verbildlicht. Analog zur Modell-Erstellung wird dieselbe Auswahl an Metriken gesammelt und vorverarbeitet. Die Datenbank wird jedoch mit einer unbekannten Workload belastet. Die Aufgabe der Klassifikation ist es nun, die Daten mit Hilfe des vorgegebenen Workload-Modells einer bekannten Workload-Klasse zuzuordnen. Dazu wird die Tabelle mit vorverarbeiteten Metriken dem Intelligent Miner über eine Stored Procedure als Input übergeben. Durch die Prozedur wird eine Sicht erstellt. Diese enthält die Eingabe-Tabelle mit den Metriken sowie zwei neue Spalten: das Ergebnis und die Güte (Qualität) der Klassifikation jedes Tupels.

Die Klassifikation kann aufgrund von starken Workload-Schwankungen oder schlechter Workload-Modelle (beispielsweise aufgrund von geringer Zahl der Datensätze in der Lernphase) ungenau sein, weshalb man die aktuelle Workload z.B. über eine **Mittelung** mehrerer Workload-Klassifikationen (eines gleitenden Zeitfensters) bestimmen kann. Dieses nahe liegende **Glätten** beseitigt fehlerhafte Klassifikationen und erhöht die Genauigkeit. Eine gesteigerte Genauigkeit erkauft man sich jedoch mit einer höheren Dauer bis zum Erkennen der aktuell anliegenden Workload.

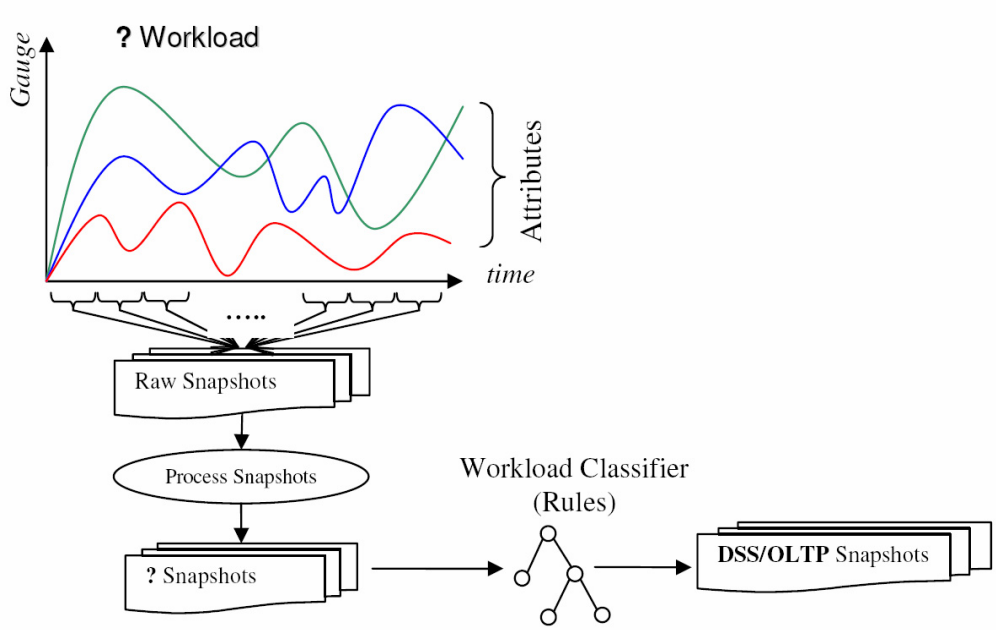


Abbildung 7.6: Nutzung des Classifiers zur Workload-Erkennung (nach [Eln04])

Wird eine Änderung des Workload-Typs (**Workload-Shift**) erkannt, so kann die Kenntnis darüber die in Zukunft erzeugten Ereignisse beeinflussen und die damit verbundenen Tuning-Pläne bzw. deren Schritte dynamisch anpassen. Konkrete Beispiele für die (automatisierte) Nutzung des Wissens um die Workload werden in Abschnitt 8.3 zusammen mit der Beschreibung unserer Architektur für das autonome Datenbank-Performance-Tuning aufgeführt.

7.4.6 Übersicht und Bewertung erstellter Workload-Modelle

Verfahren, die auf empirischen Vergleichen oder analytischen Modellen basieren, benötigen für ihre Entscheidungen einen großen Erfahrungsschatz an verschiedenen, vor allem aber reproduzierbaren Eingabewerte-Kombinationen. Oftmals ist dies nicht durch einfaches „Wiederholen" der idealerweise in Trace- oder Log-Dateien gespeicherten Workload-Aufzeichnungen möglich. Daher ist ein **Workload-Generator** zur Erzeugung künstlicher Workload-Szenarien erforderlich, der genau die Workloads auf dem System simuliert, für die das System optimiert werden soll. Typischerweise bietet es sich an, die Modelle in einem Offline-Modus auf dem oder einem vergleichbaren System zu erlernen und später zur Laufzeit inkrementell zu verbessern, um so die zusätzliche Last auf dem System gering zu halten.

Die generierten Workloads müssen keineswegs exakt den späteren realen Workloads gleichen. Eine derartige Reproduktion ist in den meisten Fällen auch nicht möglich, sei es aufgrund der Menge an Daten, der Zahl paralleler Zugriffe oder ganz einfach aufgrund der Unvorhersehbarkeit von Workload-Schwankungen. Die künstliche Last soll lediglich plausible Start-Schätzungen für die Parameter von Modellen bereitstellen. Das „Fein-Tuning" kann und sollte letztendlich sukzessive mit der fortschreitenden Zeit erfolgen. Ein Modell-basierter Ansatz ist zudem in der Lage von konkreten Trainings-Eingaben zu generalisieren. Er kann daher mit einer Vielzahl unterschiedlicher, nicht exakt zu charakterisierender Workloads belastet werden.

Für das Training unserer Workload-Modelle und das anschließende Testen wurden TPC-C- bzw. TPC-H-angelehnte Workloads [TPCC07, TPCH08] mit einem auf unsere Bedürfnisse zugeschnittenen IBM-eigenen und -internen, Java-basierten Workload-Generator, dem **IBM Workload Expert**, erzeugt. Er bietet eine XML-basierte Sprache zur Definition und eine Laufzeit-Umgebung zur Ausführung von einfachen bzw. komplexen, parametrisierbaren Workload-Sequenzen und -Szenarien auf einer oder mehreren Ziel-Datenbanken. Die Workload-Transaktionen bzw. -Statements können hierbei dynamisch erzeugt oder aufgezeichnet und später analog wieder abgespielt werden, um eine vergleichbare Last zu generieren.

Aufgrund der großen Unterschiede der beiden Workloads kann von einem geringen Anspruch an die Klassifikation ausgegangen werden. Um die Klassifikation zu erschweren und ihre Grenzen aufzuzeigen, wurde die Datenbasis des TPC-H-Workloads mit einem weit unter der Spezifikation liegenden Skalierungsfaktor von 0.05 erstellt, wodurch die Größen beider Datenbasen in etwa angeglichen wurden. Zwar wird dadurch die Art der SQL-Statements nicht verändert, jedoch die hohe Ausführungszeit der TPC-H-Statements reduziert. Des Weiteren wurde die Anzahl der Nutzer beider Workloads angeglichen [WRRA08].

Zunächst haben wir uns auf die Erstellung eines Workload-Modells zur Unterscheidung zweier Workload-Klassen (OLAP und OLTP) unter Verwendung von Entscheidungsbäumen konzentriert. In Tests ließ die Verwendung von Entscheidungsbäumen gegenüber mittels logistischer Regression erstellter Modelle eine deutliche Qualitätsverbesserung feststellen [Göb08, Kra08, RAWR09].

Für die Erstellung des Workload-Modells wurden dem Intelligent Miner Modeling etwa 1500 Trainings-Datensätze zur Verfügung gestellt. Die Trainings-Daten stammen zu annähernd gleichen Teilen aus zwei Workload-Klassen und bestehen aus minütlich erstellten Metrik-Werten des PE.

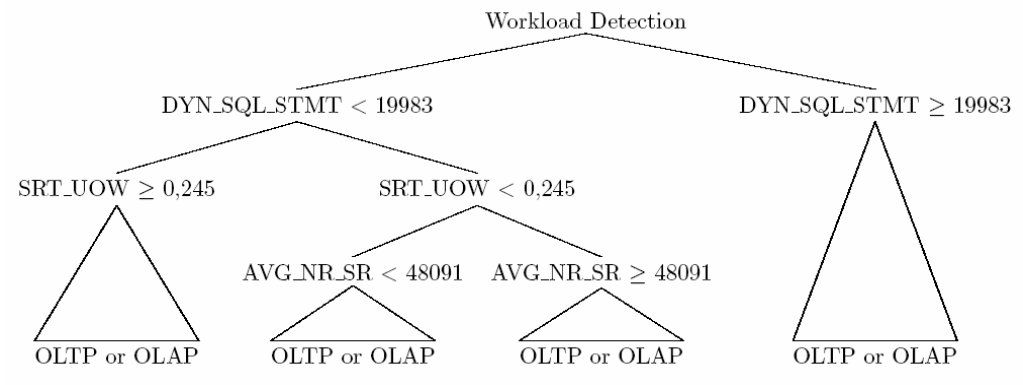
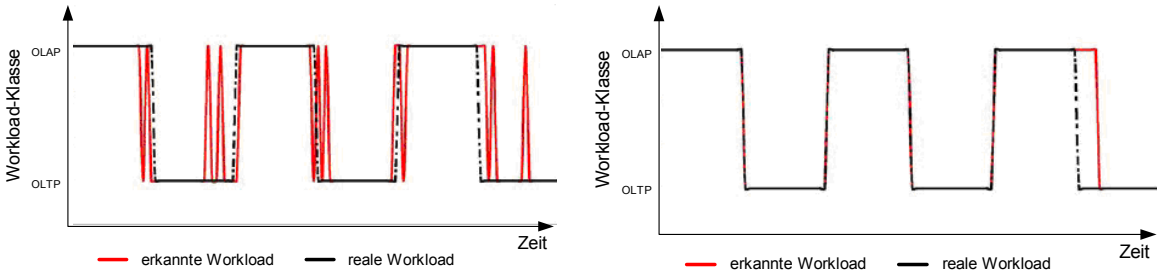


Abbildung 7.7: Schematischer Classifier-Baum für zwei Workload-Klassen (nach [RAWR09])

Abbildung 7.7 zeigt schematisch einen Ausschnitt aus dem resultierenden Baum zur Klassifikation von OLAP- und OLTP-Workloads [RAWR09]. Die drei Entscheidungs-Ebenen werden genau durch die im Rahmen der Feldbedeutung gewichteten Metriken repräsentiert. Aus vereinfachenden Gründen haben wir die restlichen Entscheidungsknoten repräsentierenden Teile des Baums lediglich als Dreiecke dargestellt. Auch sie werden benötigt, um die Workload korrekt zu klassifizieren.

Zur Bestimmung der Qualität des Workload-Modells haben wir einen 120-minütigen Testlauf, der alle 20 Minuten abwechselnd Workloads beider Klassen erzeugt, durchgeführt und die Ergebnisse des Classifiers beobachtet. Durch die beim ersten Lauf häufigen Ausreißer (**Abbildung 7.8a**) beträgt die Güte des Modells nur 88%. Durch eine Glättung (**Abbildung 7.8b**), die wir durch 5-minütiges Sammeln von Klassifikations-Ergebnissen und eine anschließende Mehrheits-Entscheidung bzgl. der Workload-Klasse des gesamten 5-minütigen Intervalls erzielten, konnten wir die Güte des Modells auf 96% erhöhen [Göb08].



a) Qualitätstest ohne Glättung

b) Qualitätstest mit Glättung

Abbildung 7.8: Qualität des 2-Klassen-Workload-Modells (in Anlehnung an [Göb08])

Die Verwendung des Entscheidungsbaum-Algorithmus macht es möglich, mehr als zwei Workload-Klassen vorauszusagen, so dass man weitere Workload-Modelle, die ggf. neben den reinen OLTP- und OLAP-Klassen auch Mischformen unterscheiden, ohne größeren Aufwand in das System einbringen kann. In einem weiteren Test wurden daher zwei zusätzliche Mischformen-Workloads integriert, so dass der Intelligent Miner einen Trainingsdateninput von nunmehr vier Workload-Klassen zu bewältigen hat [Göb08]. Die „unreine" Workload „66DS33OLTP" (Schwach-OLTP) führt stochastisch gesehen zu zwei Dritteln OLAP-Transaktionen und zu einem Drittel OLTP-Transaktionen aus. Analog dazu wurde die unreine Workload „33DS66OLTP" (Schwach-OLAP) erzeugt. Die etwas andere Art der (Entscheidungs-)Baumdarstellung unseres zweiten, auf diese Art erlernten, Classifiers durch Intelligent Miner Visualization findet sich in **Abbildung 7.9**. Hieran ist

ersichtlich, dass die grundsätzliche Entscheidung zur Bestimmung der Workload-Klasse auf der Zahl der dynamischen SQL-Statements sowie der Häufigkeit von Sortier-Operationen basiert.

Generell lässt sich sagen, dass sich die Glättung der Ergebnisse bei allen ermittelten Klassifikatoren sehr positiv auf die Genauigkeit auswirkt und eine durchschnittliche Verbesserung von ca. 10% bewirkt. Damit ist sie gerade zur Vermeidung vorschnell erkannter Workload-Änderungen aufgrund vereinzelter Fehlklassifikationen von Wichtigkeit.

Die Tests geschahen jedoch unter einer Reihe von Annahmen, welche in der Praxis so kaum gegeben sind und Änderungen der Workload-Klassifikation in einer möglichen praktischen Anwendung mit sich bringen. Workload-Shifts werden durch die Benchmarks recht abrupt vollzogen. Dieses Zeitintervall kann in der Realität mehrere Minuten bis hin zu Stunden einnehmen. So gibt es eine Übergangsdauer, in der vor allem das Klassifizieren von gemischten Workloads eine größere Bedeutung zukommt.

Die Resultate in [Göb08, WRRA08] zeigen daher, dass die Classifier teilweise Workloads mit einer Verzögerung bestimmen. Dieses Phänomen lässt sich zudem dadurch erklären, dass das PWH die Ergebnisse eines Zeitraumes verzögert zur Verfügung stellt und die Metrik-Werte des Wechselzeitraums Informationen der alten Workload enthalten. Die Ergebnisse zeigen auch, dass die Classifier fast ausschließlich beim Wechsel der Workload Schwierigkeiten haben. So werden die Workloads von allen Klassifikatoren nach einer Einschwingphase beim Workload-Wechsel nahezu perfekt erkannt.

Überdies gibt es einige Einschränkungen bezüglich der verwendeten Workload-Klassen. Es wurde versucht, die Workloads TPC-C und TPC-H anzupassen, um möglichst realistische Workloads zu erzeugen. Dennoch kann in der Realität nicht davon ausgegangen werden, dass sich Workloads anhand von Metriken leicht unterscheiden lassen. Das beschriebene Konzept setzt weiterhin voraus, dass verwendete Workload-Klassen im Vorhinein bekannt sind. In praktischen Anwendungen kann es jedoch zu unvorhergesehenen Änderungen und neuen Anwendungsbereichen der Verwendung einer Datenbank kommen.

Dennoch wurde mit der implementierten Workload-Klassifikation ein Framework einfacher Handhabbarkeit und einer Vielzahl von Konfigurations-Möglichkeiten geschaffen, mit dessen Hilfe beliebige Workloads und Workload-Shifts robust und zuverlässig detektiert werden können. Bei Versuchen erzielten generierte Workload-Modelle selbst unter den bewusst erschwerten Umständen vorzeigbare Ergebnisse. Sowohl bei den Tests ohne Tuning als auch mit parallel durchgeführtem Performance-Tuning durch den ATE (vgl. Abschnitt 8.3) wurde die anliegende Workload mit einer geringen zeitlichen Verzögerung richtig erkannt. Die umfangreichen Ergebnisse und Qualitäts-Untersuchungen sämtlicher, auf verschiedenen Paradigmen basierender Workload-Klassifizierungen sind vor allem den Arbeiten [Göb08] und [Kra08] zu entnehmen.

7.4.7 Ausblick auf mögliche Erweiterungen

Das Klassifikations-Framework unterstützt die Entwicklung von Klassifikatoren, die in Tuning-Plänen (indirekt) Verwendung finden sollen. Um die Generizität und **Austauschbarkeit** zu gewährleisten, sollte ein DBA die Klassifikatoren für seine Tuning-Pläne mitliefern. Somit muss für den Einsatz eines neuen Tuning-Plans bzw. die Belastung mit einer neuen Workload der nötige Klassifikator nicht erst neu gelernt, sondern lediglich in Details angepasst werden. Die System- und Tuning-Unabhängigkeit der in einem Klassifikator verwendeten Metriken begünstigt diesen Umstand umso mehr.

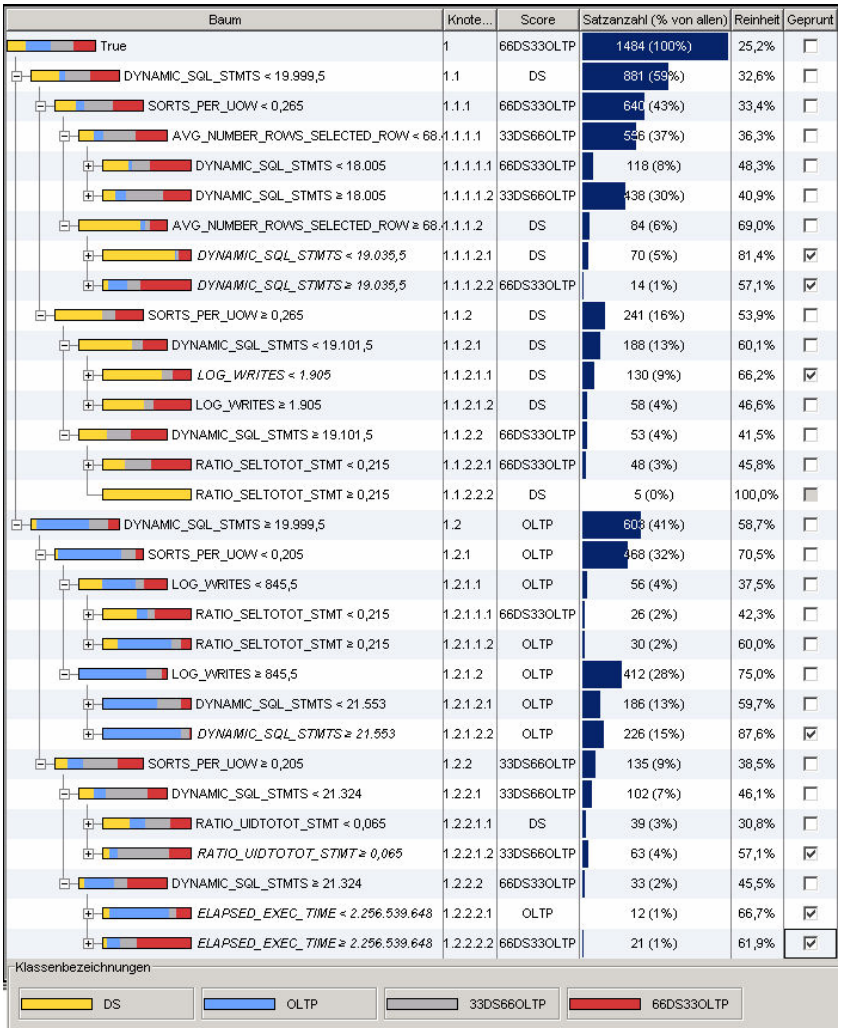


Abbildung 7.9: Auszug eines Classifier-Baums für vier Workload-Klassen [Göb08]

Es sollte zudem untersucht werden, wie das Modell zur Laufzeit anhand weiterer Test- oder auch Produktiv-Daten optimiert werden kann. Erfordernisse für ein derartiges Vorgehen können eine schlechte Güte der Klassifikation oder auch plötzliche sowie allmähliche Workload-Änderungen sein. Die **Güte** der Klassifikation kann wohl auch durch eine Verfeinerung des Glättungsverfahrens verbessert werden. Derzeit verwenden wir ein eher „naives“ Verfahren der Maximumbildung. Eine Modifikation der Glättungs-Intervalllänge unter Beachtung der sich dadurch ergebenden verzögerten Ergebnisse, die Verwendung variabler Glättungs-Intervalllängen in Abhängigkeit von Ereignissen oder auch Techniken der Zeitreihen-Analyse lassen Spielraum für Optimierungen. In jenem Fall sollten dann aber die Abstände zwischen den einzelnen Ergebnissen und damit zeitnahe Ergebnisse bei einer Mittelwert- bzw. Maximumbildung stärker berücksichtigt werden.

7.4.7.1 Die erweiterte, System-abhängige Klassifikation

Eine unbekannte Workload kann mittels der vorgestellten Workload-Komponente bereits grundlegend analysiert werden. Im Abschnitt 8.3 wird auf die Integration in den Prototypen zum autonomen Tuning eingegangen. Dieser Prototyp kann abhängig von der Workload und der Menge an durch diese bestimmten Ereignissen eine Auswahl an im Abschnitt 8.3.2 aufgeführten Tuning-Plänen und damit eine System-unabhängige Optimierung durchführen. In [Kra08] zeigen wir an praxisrelevanten Beispielen, wie mit

spezifischeren Workload-Informationen eine viel genauere Problem-Erkennung und Tuning-Plan-Auswahl möglich wird. Die allgemeine Workload-Erkennung besitzt durch ihre Differenzierung weniger, grobgranularer Workload-Klassen nicht das nötige Unterscheidungsvermögen und damit unzureichende Informationen für ein gezieltes Tuning. Die **erweiterte, System-abhängige Klassifikation** hat das Ziel, neben den typischen Workload-Klassen zusätzliche Informationen über die durch die Workload beeinflussten, logischen und physischen Ressourcen des konkreten Systems anhand geeigneter Classifier mit aufzunehmen. Somit wird zwar die Zahl der Workload-Klassen erhöht, aber das Tuning kann spezifischer erfolgen und mehr Besonderheiten berücksichtigen. Das in diesem Abschnitt präsentierte Tuning-unabhängige Workload-Modell zur Erkennung von u.a. OLAP- und OLTP-Workloads dient dabei als Grundlage und soll übernommen werden.

Mit einer erweiterten Workload-Erkennung kann dem Tuning-System ein feineres Bild über die vorherrschende Workload geliefert werden. Die zusätzlichen Informationen bieten die Möglichkeit, System-abhängige Tuning-Pläne auszuführen, die zu einer weitaus gezielteren Optimierung der Systeme als ihre generischen Gegenstücke führen können. Die Nutzer definieren zusätzliche eigene Workload-Klassen (Entscheidungsbäume) mit spezifischen Merkmalen zur Unterscheidung der Workloads. Differenzierungs- bzw. Gruppierungsmerkmale mit einer festen Menge an disjunkten Ausprägungen können Zugriffs-Charakteristiken auf bestimmte Objekte sein, Typen von (häufig ausgeführten) Statements, deren Ursprung in Form von Applikationen oder Nutzern, Datenbank-Ressourcen oder auch eine Menge an Zeiträumen, denen durch Tuning unterschiedlich begegnet werden soll. Auch eine Separierung und gesonderte Behandlung der klassischen Endnutzer-, der DBA- und der ATE-Workload (vgl. Abschnitt 4.1) ist möglich. Mit Hilfe einer erweiterten Workload-Erkennung könnte demnach bspw. die Charakteristik von Queries bestimmt und zur Ermittlung optimaler Indexe verwendet werden. Durch eine Gruppierung der Anfragen nach Bufferpools könnte zudem festgestellt werden, dass bestimmte Bufferpools für die aktuelle Workload überhaupt nicht benötigt werden. Deren Größe ließe sich in Folge reduzieren und der gewonnene Arbeits-Speicher stünde für momentan wichtigere Bufferpools bzw. andere Speicherkonsumenten zur Verfügung.

Jedes der ausgewerteten aktiven Workload-Modelle liefert ein Ergebnis zurück. Diese Informationen müssen nun noch untereinander sowie mit der eigentlichen Workload-Klasse verknüpft und (an die entsprechenden Komponenten) weitergegeben werden. Die Korrelation der einzelnen Ergebnisse kann bspw. auf Basis einer UND-Verknüpfung geschehen.

Da die Auswertung der System-abhängigen und System-unabhängigen Modelle einheitlich, idealerweise mit demselben Algorithmus, möglich sein soll, müssen diese in einer gemeinsamen Beschreibungs-Sprache beschrieben werden. Weiterhin soll die Spezifizierung in einem von Menschen leicht verständlichen und austauschbaren Format erfolgen. Wir haben uns daher für einen XML-Dialekt entschieden. Da die erweiterten Workload-Modelle System-abhängig sind, sollten sie von Nutzern erstellt und zusammen mit den Tuning-Plänen bereitgestellt und bestenfalls in einer Tuning-Community ausgetauscht werden (vgl. Abschnitt 9.2.3).

7.4.7.2 Methoden zur Workload-Vorhersage

Auch im Rahmen von [Böt08] haben wir untersucht, ob es möglich ist, Techniken des Data Mining zur Erreichung der Ziele beziehungsweise zur Unterstützung des Autonomic Computing einzusetzen und in welcher Art und Weise dies erfolgen kann. Dabei liegt der

Schwerpunkt auf den Möglichkeiten zum Lernen und der **Vorhersage** der künftigen Workload bzw. des Systemverhaltens. Für die Erstellung von Vorhersage-Modellen können unterschiedliche Techniken angewendet werden. So ist es beispielsweise möglich, Markov-Modelle, Neuronale Netze oder Data-Mining-Methoden einzusetzen. Die dadurch gewonnenen Informationen machen es möglich, bereits vor Eintreten eines die Performance oder die Gesundheit negativ beeinflussenden Ereignisses Kenntnis davon zu besitzen und rechtzeitig zusätzliche Monitoring- oder Analyse-Prozesse zu starten oder auch durch präventives Tuning aktiv zu werden.

Nach einer Einführung in die Grundkonzepte des Lernens und Vorhersagens schildern wir in genannter Arbeit zunächst den Nutzen des Einsatzes von Vorhersagen in Bezug auf das Datenbank-Tuning. Neben der allgemeinen Bestimmung zukünftiger Werte unterschiedlichster Performance-Metriken liegt der Fokus im Speziellen auf Vorhersage von Workload-Typen.

Zur Bestimmung von Prognosen können lediglich bereits in der Vergangenheit aufgetretene Ereignisse berücksichtigt werden, von denen man erwartet, dass diese in Zukunft erneut auftreten. Aufgrund dieses Prinzips der *Kontinuität* darf nicht allen einzelnen Vorhersagen ohne Weiteres vertraut werden. Um auch für bisher unbekannte Ereignisse gewappnet zu sein, müssen die Vorhersagen sowie die damit gewonnenen Informationen stets geeignet validiert und angepasst werden. Sollte es dennoch zu bisher unbeobachteten Ereignissen kommen, wird das IT-System ein von der Voraussage abweichendes Verhalten zeigen, wodurch sich eine Prognose als „wertlos“ erweist. Für nahegelegene Zeitpunkte kann ein Wert noch mit einer relativ hohen Wahrscheinlichkeit vorherbestimmt werden. Die Prognosen werden jedoch umso ungenauer und fehlerträchtiger, je weiter sie in die Zukunft gerichtet sind.

Neben der Vorhersage einzelner Metrik-Werte betrachten wir die *Korridor-Analyse* als weitere Informationsquelle. Das Ziel ist hierbei, verschiedene Metriken miteinander zu korrelieren, Beziehungen zwischen ihnen zu untersuchen und signifikante (gegenwärtige oder zukünftige) Änderungen in deren Verhältnis möglichst frühzeitig zu erkennen.

Auch Methoden der *Muster-Erkennung* können auf die durch die Metrik-Historie repräsentierten Werte-Verläufe angewendet werden, um Muster bzw. Merkmale in bestimmten Zeit-Intervallen aufzudecken und mit Hilfe dieser Annahmen über die Zukunft zu treffen. Die gefundenen Muster lassen sich auch dazu verwenden, um in der Gegenwart Situationen zu erkennen und adäquat zu behandeln, die in dieser oder ähnlicher Form bereits in der Vergangenheit aufgetreten sind.

Vorhersagen können auch zur *Validierung* und Adaption von Ereignissen über signifikante Änderungen der Workload oder des Systemzustands dienen. Für eine in der Monitoring- bzw. Analyse-Phase erkannte und als problematisch identifizierte Änderung kann nicht in jedem Fall eine Aussage getroffen werden, ob dieser Übergang lediglich für eine kurze Dauer oder auch für einen längeren Zeitraum anhalten wird. Es kann sich durchaus um eine kurzfristige Verschiebung der betrachteten Metrik(en) handeln, die, sofern sie verfrüht durch Tuning-Maßnahmen zu behandeln versucht wird, in eine daraus resultierende, unnötige (Zwischen-)Situation führen kann, die innerhalb kürzester Zeit erneut angepasst werden muss.

Weiterhin wurden mögliche Zeitpunkte untersucht, zu denen eine Vorhersage durchgeführt werden sollte. Hierbei kann unterschieden werden zwischen der *Veranlassung* durch den System-Administrator, der kontinuierlichen Vorhersage, Prognosen in bestimmten Zeitintervallen und der Steuerung durch Ereignisse. Um das System nicht mit zusätzlichem Overhead zu belasten, sollte von einer kontinuierlichen Initialisierung

abgesehen werden. Eine Möglichkeit, um Ereignisse zur Steuerung von Prognosen zu verwenden, basiert auf der Berechnung und dem Vergleich der Erwartungswerte einer Metrik. Dazu wird, wie bei den gängigen Data-Mining-Verfahren auf bisher gesammelte historische Daten zurückgegriffen. Eine ungewöhnlich hohe Differenz kann ein Anzeichen für eine Änderung in der Belastung des Systems sein und somit als Ereignis zur erneuten Vorhersage zukünftiger Werte angesehen werden. Um zu gewährleisten, dass ältere historische Werte einen kleineren Einfluss auf den Erwartungswert ausüben als aktuellere, werden die einzelnen Werte mit Gewichten versehen. In Kombination mit einer Korridor-Analyse für eine Metrik, die den Bereich um den Werte-Verlauf bestimmt, der gesunde bzw. performante Zustände charakterisiert, können neue präzisierende Vorhersagen oder auch Daten-Sammlungen sowie Tuning-Vorgänge angestoßen werden, sofern die Metrik diesen Bereich verlässt oder in Zukunft zu verlassen droht. Eine andere Möglichkeit, um Vorhersagen anzustoßen, basiert auf ermittelten Zeitabschnitten. Zunächst wird eine grobe, langfristige Prognose durchgeführt, die für eine breite Zeitdauer gültig ist. Mit Hilfe der dabei gewonnenen Informationen können einzelne, kritische Zeitabschnitte identifiziert werden, für die sich voraussichtlich eine signifikante Änderung des Systemzustands bzw. der Workload ergibt. Für die somit gewonnenen Zeitpunkte können mit entsprechendem zeitlichen Abstand kurzfristige, genauere Vorhersagen veranlasst werden.

Zur Durchführung der praktischen Untersuchungen wurde eine *Test-Umgebung* entwickelt, deren Aufgabe das Sammeln und Vorhersagen von Metrik-Werten eines DB2-Datenbanksystems ist. Dazu wurde der Einsatz sowohl von linearen als auch polynomiellen Vorhersagemodellen untersucht. Die Modell-Erstellung basiert dabei auf einer konstanten Anzahl an Schnappschüssen, die zuletzt beobachtet wurden und dem Rapid Miner[41], einer Open-Source-Software zur Durchführung von Data-Mining-Aufgaben, als Eingabe dienen. Beide Modell-Arten ergaben ähnliche Ergebnisse, doch zeigten sich jeweils auch Stärken und Schwächen.

Analog zur MAPE-Schleife wurde die Funktionsweise unserer Test-Umgebung als Kreislauf realisiert, in dem kontinuierlich der vorhergesagte mit dem aktuellen WerteVerlauf verglichen und bei Abweichungen ein neues bzw. angepasstes Modell zur Prognose bestimmt wird. Um die Ergebnisse der unterschiedlichen Experimente miteinander vergleichen zu können, hat sich als Gütemaß sowohl der prozentuale Anteil der erfolgreichen Validierungen als auch der durchschnittliche Abstand zwischen den beobachteten und vorhergesagten Werten als geeignet herausgestellt. Wenig erstaunlich ist auch der Fakt, dass die Qualität der Prognosen stark von der Wahl der entsprechenden Eingabeparameter, insbesondere der Anzahl zur Modell-Erstellung verwendeter Metrik-Werte, abhängt.

Die Experimente zeigen, dass mittels Vorhersagen gute Ergebnisse erzielt werden können. Demnach eignet sich die *lineare Regression* unter Verwendung von wenigen historischen Werten vorwiegend, um den aktuellen Trend zu bestimmen und darauf basierend die unmittelbar nachfolgenden Werte vorherzusagen. Obwohl der Beginn von Workload-Übergängen zuverlässig erkannt wird, gibt es dennoch Probleme mit der Erkennung eines neuen Zustands und der Behandlung von Ausreißern. Ein *nichtlineares Modell* kann hingegen genutzt werden, um langfristige Trends zu erkennen. Dazu ist allerdings eine entsprechend große Anzahl an historischen Daten sowie eine geeignete Aufbereitung dieser unumgänglich. Schwierigkeiten ergeben sich im polynomiellen Fall bei der Erkennung von

[41] http://rapid-i.com/content/view/181/190/lang,en/

beginnenden Workload-Shifts. Dafür gibt es weniger Probleme mit der Feststellung der neuen vorherrschenden Workload oder aber mit der Erkennung von Ausreißern.

Auch die Vorhersage kann von anderen Data-Mining-Methoden profitieren. Bei der Vorhersage mittels *Clustering* wird der Ansatz verfolgt, ähnliche Eingabedaten zu einer Nachbarschaft zusammenzufassen und für jede Nachbarschaft ein separates Vorhersage-Modell zu erzeugen. Durch diese Problem-Reduzierung kann die eigentliche Vorhersage u.U. vereinfacht werden. In [PTV05] wird dieser Ansatz aufgegriffen und ein Verfahren vorgestellt, mit dessen Hilfe für gegebene Zeitreihen Vorhersagen über den zukünftigen Verlauf getroffen werden können.

Darüber hinaus bleibt zu untersuchen, inwieweit zusätzliche Informationen durch den *Nutzer* zur Vorhersage mitverwendet werden können. Der Mensch besitzt im Vergleich zu einem IT-System in der Regel eine andere Betrachtungsweise auf die Daten und kann evtl. andere Zusammenhänge erkennen, die dem System sonst verborgen bleiben würden. Des Weiteren kann er dem System seine Kenntnis über zukünftige, durch ihn initiierte Ereignisse (z.B. Rabatt-Aktionen in einer Shop-Anwendung mit dem damit verbundenen erwarteten Workload-Anstieg) zur Verfügung stellen.

Wenn man davon ausgehen kann, dass der Workload-Typ innerhalb eines definierten Zeit-Intervalls stabil bleibt und dass Nutzer über vordefinierte Eingabemasken eine gewisse Menge an fixen Statements immer wieder absetzen, dann kann ein noch genaueres Workload-Modell erstellt werden. Hierbei ist man u.U. in der Lage, einzelne *Transaktionen* entsprechender Nutzer bzw. Nutzergruppen vorherzusagen. Einen interessanten Ansatz in diese Richtung diskutieren [HoRi07]. Dort wurde ebenfalls erkannt, dass sich die Workload eines Datenbanksystems nicht rein zufällig zusammensetzt. Die Autoren beschränken sich dabei auf spezifische OLTP-Umgebungen und demonstrieren durch einen direkten Eingriff in das DBMS, wie diese Informationen ein effektiveres Prefetching ermöglichen. Dadurch können relevante Tabellendaten bereits in den Bufferpool geladen und somit die Ausführungszeiten der Statements reduziert werden. Auch [Bro08] hat sich u.a. mit dem möglichen Nutzen derart repetetiver Workload-Szenarien beschäftigt und weitere Anwendungsgebiete gewonnen. So kann die Ausführungszeit einer Anfrage nicht nur durch Prefetching, sondern auch durch eine Vorkompilierung und Optimierung dieser reduziert werden. Durch ein Sammeln von Transaktionen in einer Warteschlange und eine Priorisierung dieser können Ausführungsdauern und Wartezeiten adäquater koordiniert und eine Verbesserung der Systemperformance erreicht werden.

Neben den angedeuteten Anwendungsgebieten der Transaktions- bzw. Anfragen-Vorhersage, lässt sich dieses Wissen auch für Aussagen über mögliche Entwicklungen von Performance-Metriken verwenden. Mittels der über die Statements zugegriffenen Tabellen bzw. Sichten lassen sich die dafür verantwortlichen (physischen) DB2-Ressourcen zusammen mit ihren (künftigen) Sensor-Werten ermitteln und an die künftige Situation konfigurierend anpassen.

Weiterführende Arbeiten haben aufzuzeigen, wie eine Einbindung der Vorhersage in unseren Prototypen konkret auszusehen hat, zu welchem genauen Zweck im Sinne des Datenbank-Tuning Vorhersagen durchgeführt werden sollen und inwieweit sich die obigen Vorgehen der Transaktionsprognose auf unser Szenario übertragen lassen.

7.5 Ein Ressourcen-Modell zur Abbildung quantitativer und qualitativer Abhängigkeiten

Eine der wesentlichen Charakteristiken selbst-verwaltender Systeme ist deren Bewusstsein über den Aufbau und die Wirkungsweise ihrer (dynamischen) Umgebung. Dazu gehören neben den im vorangegangenen Abschnitt vorgestellten Informationen über die Workload vor allem das Wissen über die einzelnen Ressourcen, ihre Eigenschaften und zulässigen Operationen zur Beeinflussung deren Verhaltens bzw. deren Struktur sowie ihre Abhängigkeiten untereinander. Ursachen für schlechte Performance können vielfältig, aufgrund von Wechselbeziehungen zwischen Systemkomponenten oftmals schwer nachvollziehbar und häufig nicht offensichtlich sein. Stellschrauben, die heutige DBMS anbieten, um Performance-Engpässe zu beseitigen, sind zum Teil sehr verschieden und wurden bereits u.a. in Abschnitten 2.1.2 sowie 4.1.1 detailliert. Viele dieser Eingriffe können im laufenden Datenbankbetrieb vorgenommen werden, andere wiederum nicht. Es gibt Parameter-Änderungen, die nur wirksam werden, wenn sie zusammen mit abhängigen ausgeführt werden. Aufgrund dieser daraus resultierenden Vielfalt an denkbaren Tuning-Aktionen, ist es umso wichtiger, die nahezu bestmögliche Tuning-Maßnahme ermitteln zu können.

Mittels Metriken, welche die Auswirkungen bzw. die Performance des Systems repräsentieren, können Wirkungen solcher Maßnahmen ermittelt und bewertet werden. Ein allgemeines Verständnis über die relevanten Ressourcen und aus Abhängigkeitsbeziehungen resultierende Interaktionen kann daher für viele Entscheidungsprozesse und damit in einem autonomen System für beinahe sämtliche Phasen von der Überwachung, über die Analyse, bis hin zur Planung von großem Nutzen sein. Auf diese Weise lässt sich bspw. ermitteln, welche Stellschrauben unter welchen Bedingungen bzw. in welcher Reihenfolge betrachtet werden sollen und inwiefern sie relevant für die Performance bzw. für eine Verbesserung/Verschlechterung sind. Der direkte bzw. indirekte Einfluss auf die entsprechenden Metriken sollte ebenso abgeschätzt werden können wie die Höhe der Veränderung und etwaige damit verbundene Seiten-Effekte. Den Großteil der konkreten Anwendungsfälle haben wir bereits in Abschnitt 4.4 ausführlich erläutert.

Die eigentliche Herausforderung liegt nunmehr in der Konzipierung eines Ressourcen-Abhängigkeits-Modells zur Beschreibung *qualitativer* und *quantitativer*, Workload-abhängiger (Effekte von) Interdependenzen zwischen (ausgewählten) Effektoren und Sensoren (und damit auch zwischen den Ressourcen des betrachteten Systems). Dieser Abschnitt setzt sich zum Ziel, die diesbezüglich in [Exn07, Gan06, Göb08, RaWi07, WiRa07, WiRa09] ergründeten Konzepte sowie die im Rahmen zahlreicher Evaluationen gewonnenen Ergebnisse näher zu beleuchten.

7.5.1 Bestimmung qualitativer Abhängigkeiten

Unsere Ausführungen in [Gan06] befassen sich detailliert mit der Bestimmung **qualitativer Abhängigkeiten**. Darunter verstehen wir das Wissen um die Existenz und Wirkung von Verbindungen zwischen Sensor- und Effektor- bzw. zwischen Effektor-Elementen, die mathematisch (noch) nicht genau beschrieben werden können.

Es wird hierbei jedem Paar, bestehend aus einem Effektor und einem Sensor, eine Abhängigkeit zugeordnet. Natürlich stellt dabei Unabhängigkeit auch eine Art Abhängigkeit dar. Eine solche **Sensor-Abhängigkeit** sagt aus, dass aus einer Effektorwert-Änderung eine Veränderung des Sensorwerts resultiert. Ein Sensor ist demnach genau dann von einem Effektor (qualitativ) abhängig, wenn eine (im Detail unbekannte) Funktion

f existiert, welche unter Einhaltung gewisser Nebenbedingungen (an bzw. durch die Umgebung und Workload) die Effektorwerte auf die Sensorwerte abbildet.

Neben der paarweisen Abhängigkeit der Sensoren von den Effektoren existieren auch Beziehungen zwischen Effektoren und Effektoren, die aus deren Beschaffenheit und Definition resultieren. So können im Rahmen einer solchen **Effektor-Abhängigkeit** die Werte des Effektors e1 abhängig sein von Werten des Effektors e2. Um bspw. den DB2-Datenbankkonfigurationsparameter LOGSECOND[42] auf „-1“ zu setzen, muss der Parameter USEREXIT[43] bereits auf „yes“ stehen. Es kann zudem Effektoren geben, die zusammen geändert werden sollten, um einen maximalen Einfluss zu gewährleisten (z.B. die Konfigurationsparameter MAXLOCKS[44] und LOCKLIST[45]). Zusammengefasst unterscheiden wir also die Fälle, dass Parameter in Kooperation geändert werden sollten oder nur in einer bestimmten Reihenfolge adjustiert werden können. Dabei kann es entweder vorkommen, dass die Wirkung von e1 die Wirkung von e2 verstärkt (komplementäre Parameter) oder auch verringert (kompetetive Parameter).

Im Verlauf unseres Beitrags [Gan06] wurde eine formale System-Modellierung erarbeitet und in [RaWi07, WiRa09] erweitert, die Parameter und Metriken als spezielle Formen der Effektoren und Sensoren in Beziehung setzt. Dies erlaubt es, Aussagen treffen zu können, wie sich die Performance bei bestimmten Veränderungen der Konfiguration einzelner Datenbank-Objekte verhält. Eingebettet in eine Objekthierarchie können diese Beziehungen handhabbar gemacht und dazu verwendet werden, Tuning-Maßnahmen zu bewerten und Performance-Auswirkungen vorhersagbar zu machen. Grundlage zur Beschreibung eines derartigen Abhängigkeits-Modells ist die geeignete Abbildung der Ressourcen-Topologie, vor allem ihrer hierarchischen Beziehungen untereinander. Auf dieses **Topologie-Modell** können schließlich die modellierten Wechselwirkungen zwischen den Sensor- und Effektor-Elementen der Ressourcen aufgesetzt werden. In unseren genannten Arbeiten geben wir nicht nur einen Einblick in die formalen Definitionen der angesprochenen Modelle, sondern klassifizieren und betrachten exemplarisch auch einige der den Inhalt ausmachenden Elemente im Kontext von DB2.

Abbildung 7.10 zeigt einen Ausschnitt unseres DB2-Abhängigkeits-Modells. Dabei werden zum einen Sensor-Abhängigkeiten innerhalb einer Ressource sowie zwischen Ressourcen demonstriert. Zum anderen ist erkenntlich, dass die Datenbank-Konfigurationsparameter ESTORE_SEG_SZ und NUM_ESTORE_SEGS zur Beeinflussung der Größe und Anzahl von Bufferpool-Erweiterungs-Segmenten nur dann einen Einfluss auf die BPHR haben, wenn für den entsprechenden Bufferpool auch der Extended Storage über den entsprechenden Effektor aktiviert wurde [IBM06c]. In dem dargestellten Beispiel existiert zusätzlich auf Objekttyp-Ebene eine hierarchische Beziehung zwischen der Datenbank und dem Bufferpool. Ein Auszug der hierarchischen Beziehungen des Topologie-Modells findet sich bereits in Abschnitt 6.3.2.3, Abbildung 6.7. Die konkreten Ausprägungen dieser hierarchischen Beziehungen auf der dargestellten Objekt-Ebene wurden jedoch aus Gründen der Übersichtlichkeit nicht eingezeichnet.

Ein Teil der Informationen über existente Abhängigkeiten zwischen Ressourcen in DB2 stammt aus Produkt-Dokumentationen und eigenen Erfahrungen. Darüber hinaus wurden im Rahmen unserer Arbeiten zwei Programme erstellt, die sich mit der Ermittlung und

---

[42] Der Parameter LOGSECOND beschreibt die Anzahl an sekundären Log-Dateien.
[43] Der Parameter USEREXIT dient der Aktivierung der Archivierung von Log-Dateien mit Hilfe einer User-exit-Routine.
[44] Der Parameter MAXLOCKS spezifiziert den prozentualen Anteil einer Applikation an der Sperrliste.
[45] Der Parameter LOCKLIST definiert den für die Sperrliste allokierbaren Gesamtspeicher.

Bewertung von Sensor-Abhängigkeiten beschäftigen. Sie dienen aufgrund ihrer Beschaffenheit als Test-Umgebung für zukünftige Arbeiten.

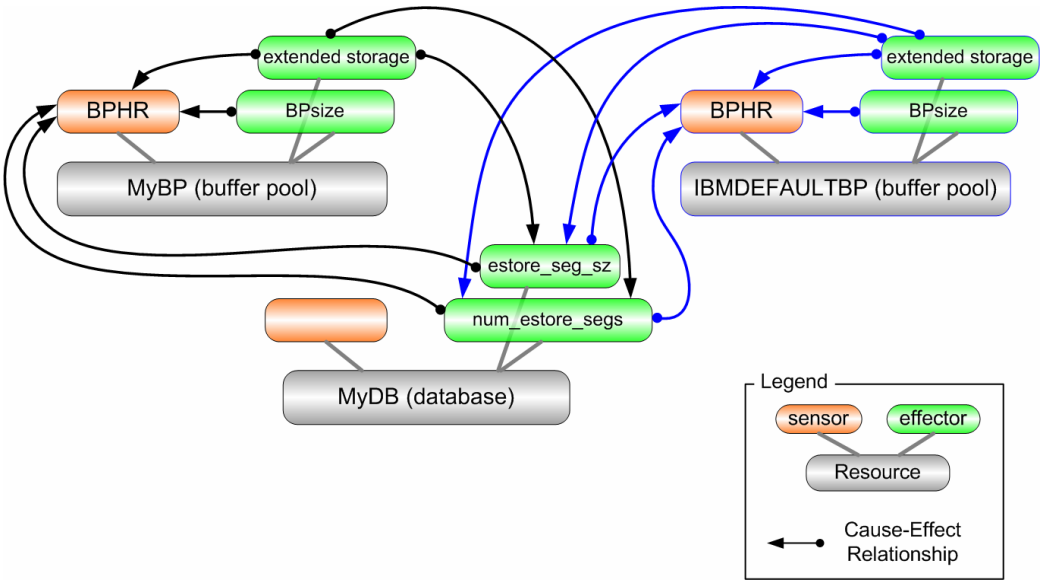


Abbildung 7.10: Beispiel eines Abhängigkeits-Modells für DB2 (nach [RaWi07])

Zum einen handelt es sich dabei um das **Dependency Detection Tool (DTT)**, welches zu einem ausgewählten Effektor (Parameter) sämtliche abhängige Sensoren (Metriken) ermittelt [Gan06]. Es kann bspw. verwendet werden, um alle durch einen Tuning-Plan beeinflussten Ressourcen zu bestimmen und über ein Kosten-/Nutzen-Modell die Tuning-Plan-Auswahl zu unterstützen. Das **Dependency Evaluation Tool (DET)**, zum anderen, misst für eine Menge an gegebenen Parametern und Metriken die Abhängigkeiten und stellt die Messwerte zur Weiterverarbeitung bereit [Gan06, Exn07]. Sowohl das DDT als auch das DET führen Messungen auf der Grundlage von Parameterwert-Änderungen durch. Dafür müssen zusätzliche Metadaten über den Typ und Wertebereich der Effektoren im Ressourcen-Modell hinterlegt und zugreifbar sein. Zusammenfassend kann die Überprüfung auf Abhängigkeit damit beschrieben werden, dass im Laufe der Parameterwert-Änderungen der Wert einer Metrik eine gewisse Veränderung aufweisen und diese sich in mehreren Durchläufen bestätigen muss.

Beide Tools sollen als Test-Umgebung für das Ermitteln und Bewerten von Sensor-Abhängigkeiten fungieren und verfügen jeweils über eine eigene Benutzer-Oberfläche, von denen aus der Nutzer die Messvorgänge steuern kann. Die graphische Benutzer-Oberfläche des DDT zeigt nach und nach alle Metriken an, die bereits überprüft wurden oder gerade überprüft werden. Ergibt die Überprüfung eine Abhängigkeit, so wird die abhängige Metrik entsprechend präsentiert. Die Ergebnisse aus den einzelnen Messvorgängen werden dem Nutzer in unterschiedlicher Weise präsentiert.

Mittels DET können Abhängigkeiten zwischen Parametern und Metriken nicht nur ermittelt sondern auch mit Messwerten beschrieben werden. Es eignet sich besonders zur automatisierten, systemspezifischen Aufdeckung von Abhängigkeiten unter einer gegebenen Workload. Dabei werden in einem iterativen Prozess sämtliche Werte-Kombinationen der Parameter anhand der gleichen Workload-Bedingungen getestet und etwaige Metrikwerte-Änderungen nach einer Stabilisierungsphase festgehalten. Durch gezieltes Anpassen der Messvorgänge von Metrikwert-Änderungen aufgrund von Parameter-Änderungen kann die mit DET aufgebaute und gespeicherte Messwerte-

Sammlung zudem zur Bestimmung funktionaler Zusammenhänge zwischen Parametern und Metriken verwendet werden (siehe Performance-Funktionen im Abschnitt 7.5.2).

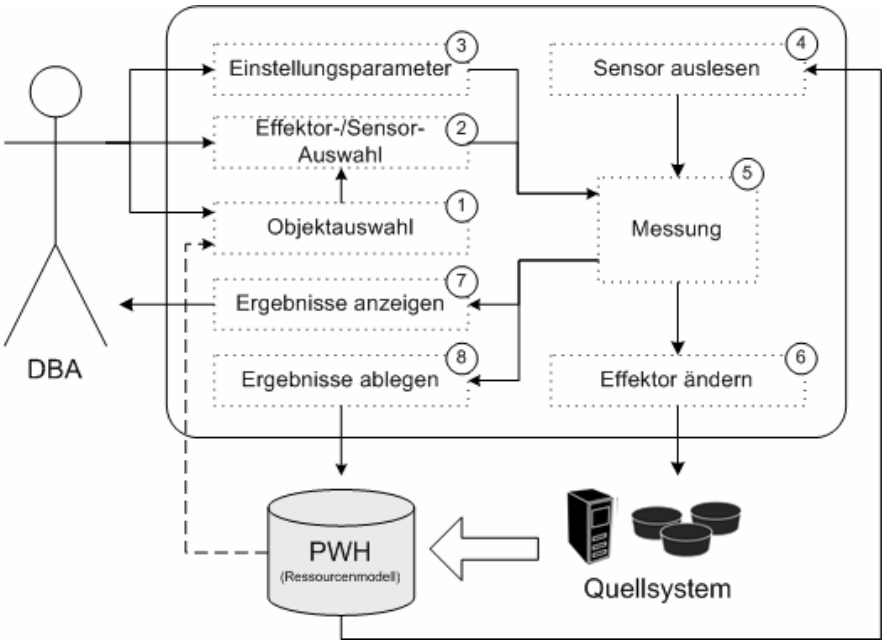


Abbildung 7.11: Workflow des DET-Prototypen und Anbindung an das PWH

Ein Abhängigkeits-Modell kann mit Hilfe des DET erstellt und zur Laufzeit aktualisiert werden. Wir haben DET insoweit angepasst, dass dem Tool nun u.a. das hierarchische (Ressourcen-)Modell des PWH zu Grunde liegt. Der grundsätzliche Arbeitsablauf des DET-Tools wird in **Abbildung 7.11** präsentiert. Über das User-Interface wählt man zunächst einer Ressource oder mehreren Ressourcen zugeordnete Parameter und Metriken (1,2). Das DET benötigt für seine Messungen mindestens einen Parameter eines Objekts und eine Metrik eines eventuell anderen Objekts. Für die Auswahl der Objekte für das DET (und auch des Objekts im DDT) werden alle Objekte, die dem Topologie-Modell zur Verfügung stehen, über eine hierarchische Baum-Struktur angezeigt. Über spezielle Einstellungsfenster hat der Nutzer die Möglichkeit, mittels Einstellungsparameter die Art der Messungen und den Messablauf (z.B. Anzahl an Messpunkten oder Zeitintervall zwischen zwei Messpunkten) zu steuern (3). Für jede der Parameterbelegungen (6) erfolgen mehrere Messungen (4,5), die anschließend gemittelt, dem Nutzer präsentiert (7) und zusätzlich für die spätere Analyse im PWH gespeichert werden (8). Ein Screenshot des Tools ist in **Abbildung 7.12** zu finden.

Die in der Arbeit vorgestellte System-Modellierung ermöglicht es, die Auswirkungen von Tuning-Plänen bestimmen und bewerten zu können. Die Beziehungen zwischen den in den Tuning-Plänen enthaltenen Parameter-Änderungen und Veränderungen von die Performance des Systems repräsentierenden Metriken können vorausschauend Auskunft über das Verhalten der Performance geben, bevor eine bestimmte Tuning-Aktion ausgeführt werden soll. Auf diese Weise können mehrere mögliche Tuning-Pläne miteinander verglichen und entsprechend ihrer Wirkung eingeordnet werden. Für weitere Informationen zu den theoretischen Konzepten, den darauf basierenden Modellen und den entwickelten Tools soll an dieser Stelle nochmals auf unsere Arbeiten [Gan06, Exn07, RaWi07, WiRa07, WiRa09] verwiesen werden.

Abbildung 7.12: Die graphische Benutzer-Oberfläche des DET

Darüber hinaus evaluieren wir in [Göb08] neben der Bestimmung der Workload, auch **Data-Mining-Methoden** zur Ermittlung von Korrelationsbeziehungen. In der Arbeit wurden Tests zur praktischen Untersuchung sowohl der Möglichkeiten der Assoziations-, als auch der Möglichkeiten der Korrelations-Bestimmung aufgenommen, ausführlich erläutert und anhand der Ergebnisse bewertet.

Assoziationen wurden bereits im Abschnitt 2.4.2 eingeführt. Der empirische *Korrelationskoeffizient* nach Pearson gibt laut [Tre08] das Maß der linearen Abhängigkeit zweier Zufallsvariablen an. Das Vorzeichen des Korrelationskoeffizienten ist bei einer direkten Proportionalität der Zufallsvariablen positiv und bei indirekter Proportionalität negativ. Ein Betrag nahe eins deutet auf einen linearen Zusammenhang hin, ein Betrag nahe null zeigt keinen wesentlichen Zusammenhang. Dabei können neben den qualitativen Abhängigkeiten der Metriken von Konfigurationsparametern, auch (existentielle) Zusammenhänge unter Metriken und unter Parametern identifiziert werden. Die Korrelation kann nicht genutzt werden, um nichtlineare Zusammenhänge zu erkennen. Wir befassen uns jedoch in Abschnitt 7.5.2 mit dieser Herangehensweise.

IBM Intelligent Miner (IM) bietet eine Reihe an Methoden, um Zusammenhänge und Korrelationen zwischen den Spalten innerhalb einer Tabelle zu ermitteln. Letztere muss dafür Werte sowohl von Metriken, als auch von Konfigurationsparametern enthalten. Um repräsentative Zusammenhänge zu erhalten, sollten zahlreiche und stark variierende Konfigurationen gewählt werden. Die Metrik-Werte verschiedener Konfigurationen müssen unter homogenen Testbedingungen geschaffen werden. Hierfür kann das obige

DET-Tool verwendet werden, das ausgehend von einer nutzerdefinierten Auswahl von Konfigurationsparametern systematisch Kombinationen verschiedener Werte durchgeht und parallel Konfigurationsparameter- und Metrik-Werte in einer Tabelle sichert. Um den Problemraum einzugrenzen, beschränken wir uns auf lediglich drei verschiedene, vom Nutzer zu spezifizierende bzw. automatisch vorbelegbare Werte je Parameter. Das Tool aus [Gan06] wurde zudem in [Göb08] soweit angepasst, dass auch das gleichzeitige Ändern mehrerer Parameter in einem Durchlauf unterstützt wird. Der Nutzer hat die maximale Zahl der Parameter auszuwählen, die gleichzeitig verändert werden sollen und das Tool generiert selbstständig alle möglichen Kombinationen (durch Bildung einer modifizierten Potenzmenge aller möglicher Parameter-Werte). Auch hier spielt die Homogenität der Testszenarien eine bedeutende Rolle, um korrekte Zusammenhänge der Konfigurationsparameter und Metriken zu erkennen. Jedes Szenario beginnt mit der Änderung der Konfigurationsparameter. Daraufhin wird das DBMS neu und mit stets der gleichen Workload gestartet. Verschiedene Prozeduren des Intelligent Miner können im Anschluss an die Warmlauf- und die Mess-Phase dazu genutzt werden, Zusammenhänge und Korrelationen innerhalb der Tabelle durch Erstellung eines Assoziationsmodells und eines Korrelationskoeffizienten zu erkennen.

Das Assoziationsmodell wird nach seiner Erzeugung durch einen Aufruf von IM Visualization dargestellt und kann durch die GUI des Visualizers nach interessanten Zusammenhängen untersucht werden. Um die Möglichkeiten der Assoziationsmodelle aufzuzeigen, haben wir in [Göb08] den Einfluss des im Laufe der Messung zu vergrößernden privaten Sortier-Speichers (SORTHEAP) auf die durchschnittliche Sortierdauer (AVG_SORT_TIME) und den gesamten, gemeinsam genutzten Sortier-Speicher (SORT_SHRHEAP_TOP) untersucht. **Abbildung 7.13** beinhaltet die Darstellung dieses Modellteils durch IM Visualization. Dabei wurde die Regelmenge auf Regeln eingeschränkt, deren Rumpf SORTHEAP jeweils einen Wert zuweist. Die verschiedenen Werte der Eingabefelder ermöglichten 13 Items, die in 29 verschiedenen Kombinationen auftraten und zu 32 Assoziationsregeln führten [Göb08].

Diese Darstellung kann nun zur Bestimmung von Zusammenhängen genutzt werden [Göb08]. So ist ersichtlich, dass jede Änderung von SORTHEAP eine Änderung von SORT_SHRHEAP_TOP zur Folge hat. Da mit zunehmender Sortierspeicher-Größe auch der gemeinsam genutzte Sortier-Speicher erhöht wird, handelt es sich um einen direkt proportionalen Zusammenhang. Für die Sortierdauer ergibt sich ein ähnliches Bild. Da in der Regel jeder Sortierspeicher-Größe eine verschiedene durchschnittliche Sortierdauer zugewiesen wurde, ist auch hier eine Korrelation vorhanden, die jedoch indirekter Proportionalität entspricht. Die für eine Sortierspeicher-Größe von 1024 Seiten ersichtliche variierende Sortierdauer lässt sich wohl auf die Test-Umgebung zurückführen, deren absolute Homogenität nicht erreicht werden kann. Dies zeigt sich auch in den sehr geringen Abweichungen von 0% und 1%. Die Ergebnisse entsprechen daher den vorherigen Erwartungen. Eine Erhöhung des privaten Sortier-Speichers führt zu einer schnelleren Sortierung und zu einer Vergrößerung des gemeinsam genutzten Sortier-Speichers.

Tabelle 7.6 gibt einen Überblick über einen kleinen Teil der Ergebnisse der Korrelationsbestimmung. Insgesamt wurden 25 Konfigurationsparameter und 204 Metriken in der erzeugten Tabelle erfasst. Dabei haben wir uns auf relevante Korrelationskoeffizienten beschränkt, die einen Wert von 0.95 über- bzw. einen Wert von -0.80 unterschreiten. Zwischen Konfigurationsparametern wurden keine relevanten Korrelationen entdeckt. Der Großteil der Ergebnisse beschränkt sich auf Korrelationen zwischen Metriken. So wurde u.a. erkannt, dass eine Erhöhung ausgeführter Updates,

Inserts und Deletes (RATIO_UIDTOTOT_STMT) in jedem Fall zu einem erhöhten Schreiben von Protokollen (LOG_WRITES) und mehr eingefügten Zeilen (ROWS_INSERTED) führt. In DB2 lässt sich die durchschnittliche Anzahl aktiver Anwendungen einer Datenbank (AVG_APPLS) konfigurieren. Dieser Parameter wird vom Anfrage-Optimierer zur Ermittlung der vorhandenen Bufferpool-Speichergröße einer Anwendung genutzt. Der hohe Korrelationskoeffizient der beiden Felder besagt nun, dass eine höhere Anzahl aktiver Anwendungen zu einer niedrigeren Trefferquote von Bufferpool-Abfragen führt.

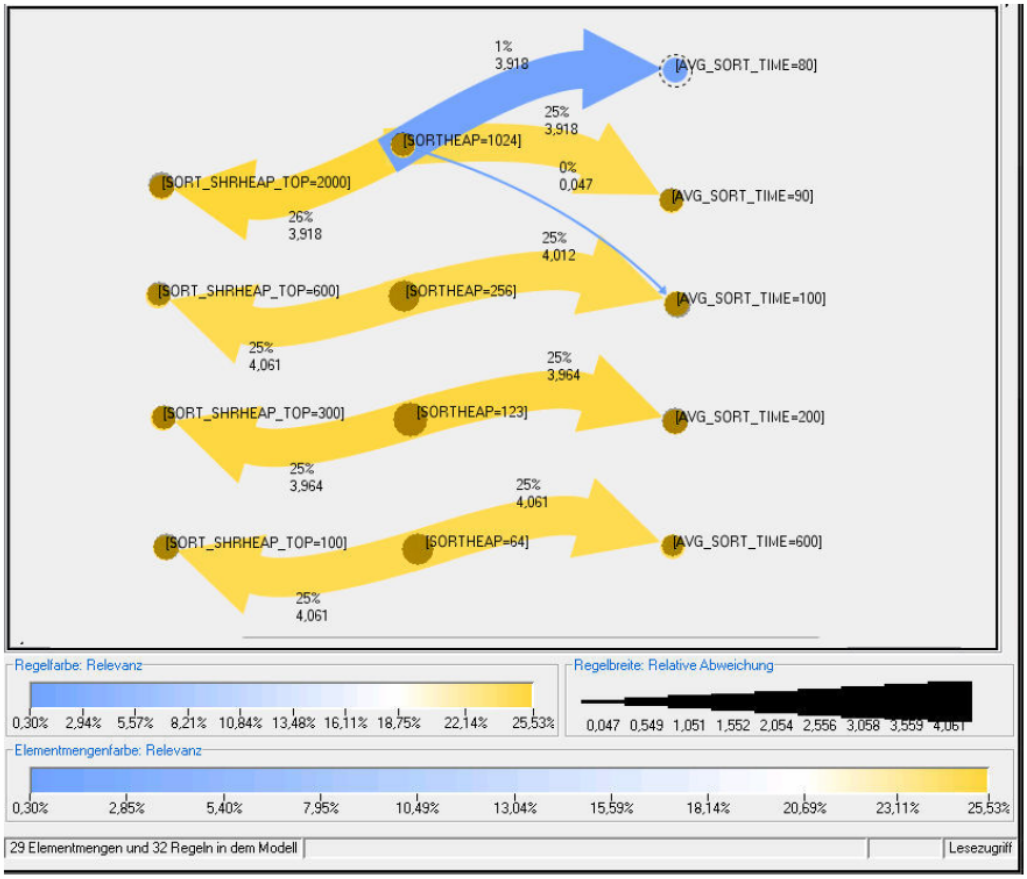


Abbildung 7.13: IM-Visualization- Auszug unseres Assoziationsmodells (nach [Göb08])

Durch die Evaluierung wurde deutlich, dass Assoziationsregeln lediglich als visuelle Stützen (zur Erstellung neuer Tuning-Pläne) dienen können und Regeln hierbei durch Nutzung von IM Visualization gezielt begrenzt werden müssen. Die Korrelationsbestimmung ist in der Lage, eigenständig gewünschte Zusammenhänge aufzudecken und in Zusammenarbeit mit anderen Werkzeugen, Tuning-Pläne zur entwerfen oder den Entwurf durch visuelle Darstellung von Korrelationen zu unterstützen. Durch eine Anwendung zur Laufzeit könnten wiederholt erfolglose Tuning-Pläne in der erneuten Ausführung verhindert und suboptimale verbessert werden. Der Korrelationskoeffizient kann bspw. auch dafür genutzt werden, um zu ändernde Konfigurationsparameter zu finden, wenn eine Metrik einen kritischen Wert erreicht. Der Koeffizient gibt dabei lediglich den Einfluss des Parameters wieder und ist nicht in der Lage optimale Werte der Konfigurationsparameter zu errechnen. Dies kann beispielsweise durch die im nachfolgenden Abschnitt vorgestellten Techniken der nichtlinearen Optimierung erreicht werden, die Intelligent Miner nicht zur Verfügung stellt.

Variable 1	Variable 2	Faktor
ROWS_INSERTED	UID_SQL_STMTS	1.0
LOG_WRITES	NUM_LOG_PART_PAGE_IO	1.0
LOG_WRITES	NUM_LOG_WRITE_IO	1.0
LOG_WRITES	**ROWS_INSERTED**	**1.0**
LOG_WRITES	**UID_SQL_STMTS**	**1.0**
NUM_LOG_PART_PAGE_IO	NUM_LOG_WRITE_IO	1.0
NUM_LOG_PART_PAGE_IO	ROWS_INSERTED	1.0
NUM_LOG_WRITE_IO	ROWS_INSERTED	1.0
NUM_LOG_WRITE_IO	UID_SQL_STMTS	1.0
NUM_LOG_PART_PAGE_IO	UID_SQL_STMTS	1.0
APPL_SECTION_LOOKUPS	DYNAMIC_SQL_STMTS	1.0
COMMIT_SQL_STMTS	STATIC_SQL_STMTS	1.0
AVG_APPLS	**POOL_HIT_RATIO**	**-0.9**
POOL_HIT_RATIO	SORT_OVERFLOWED_RATIO	-0,9
AVG_NUMBER_ROWS_-SELECTED_ROW	POOL_HIT_RATIO	-0.89

Tabelle 7.6: Auszug der ermittelten Parameter-Sensor-Korrelationen (nach [Göb08])

7.5.2 Bestimmung quantitativer Abhängigkeiten

Das Ziel des Performance-Tuning liegt zumeist in der Maximierung der Performance unter Beachtung von auf höherer Ebene vorgesetzter *variabler Nebenbedingungen* (Policies). Dabei wird mit Hilfe von Best-Practices-Methoden versucht, die Performance *schrittweise* zu verbessern und Seiten-Effekte zu minimieren. Alternativ dazu sehen wir die Konzipierung eines mathematischen Modells (quantitativer) Sensor-Abhängigkeiten, mit dessen Hilfe eine **Performance-Funktion** f_p aufgestellt werden kann, welche sämtliche Abhängigkeiten modelliert und auf ein Performance-Maß abbildet [Exn07].

Mittels **quantitativer Abhängigkeiten** lassen sich Wechselbeziehungen zwischen Effektoren und Sensoren eines Systems mathematisch anhand einer Funktion beschreiben, die Effektor- auf Sensor-Werte abbildet. So können der Werteverlauf eines Sensors und die dafür verantwortlichen Effektoren charakterisiert werden. Darüber hinaus lassen sich konkrete Wertebelegungen für Effektoren ermitteln, um für einen Sensor einen spezifischen Wert zu erzeugen. Bei der Beschreibung solcher Abhängigkeiten müssen jedoch neben den *Effektoren*, wie Konfigurationsparameter oder Umgebungsvariablen, auch nicht direkt regulierbare Faktoren bzw. *fixe Nebenbedingungen*, wie seine Umgebung und Workload, berücksichtigt werden. Diese beeinflussen sich nicht nur gegenseitig, sondern vor allem die Performance des Systems, die bspw. mittels Sensoren wie Antwortzeit oder Anzahl an Transaktionen pro Zeiteinheit gedeutet werden kann.

Sofern man die Performance (bzw. den Zustand) eines Systems zum Zeitpunkt t als Tupel von Metriken $M = (M_1, \ldots, M_l)$ beschreibt, kann die **Performance-Funktion** f definiert werden als

$$f_{P,M}: D_{P_1} \times \ldots \times D_{P_k} \rightarrow D_{M_1} \times \ldots \times D_{M_l}$$

Dabei entsprechen $D_{P_1},\ldots, D_{P_k}$ bzw. $D_{M_1},\ldots, D_{M_l}$ den diskreten Wertebereichen des Parameter-Tupels $P=(P_1, \ldots, P_k)$ bzw. des Metrik-Tupels $M=(M_1, \ldots, M_l)$. Die Funktion f drückt somit den funktionalen Einfluss der Parameter P_i auf die Metriken M_i aus. Auch gemeinsame Einflüsse von mehreren Parametern auf die Performance können so dargestellt werden. Auf die Betrachtung der Nebenbedingungen und der Zeit sei an dieser Stelle vereinfachend verzichtet. Vergleiche hierzu die Ausführungen in [Exn07].

Ist nun für ein System die zugehörige Performance-Funktion und damit der Einfluss sämtlicher Parameter auf die Performance bekannt, lässt sich ein entsprechendes **Optimierungsproblem** ohne Weiteres ableiten und lösen. Zur Laufzeit müsste die zu erweiternde, selbst-optimierende Komponente „lediglich" die Parameterbelegung berechnen, die ein Optimum für f_p und damit die höchste Performance liefert. Leider sind der Einfluss, den verschiedene Parametereinstellungen auf die Performance haben und damit auch die konkrete Performance-Funktion, weitestgehend unbekannt.

Um mathematische Optimierung für das Performance-Tuning dennoch verwenden zu können, müssen in einem ersten Schritt zunächst die quantitativen Abhängigkeiten, die den Einfluss von Parametern auf Metriken beschreiben approximiert werden. Daran schließt sich das Herleiten einer Zielfunktion *f* für die Performance und das Lösen des so erhaltenen Optimierungsproblems *max f* an, so dass alle Nebenbedingungen erfüllt sind.

Aufbauend auf den Erkenntnissen aus [Gan06] untersuchen wir in [Exn07] daher Methoden und Lösungsverfahren der nichtlinearen Optimierung zur Ermittlung der quantitativen Zusammenhänge zwischen Effektoren und Sensoren und damit zur Erarbeitung eines solchen mathematischen Modells (Zielfunktion, Nebenbedingungen). Die Approximation des Zusammenhangs zwischen Effektoren und Sensoren erfolgt durch eine **nichtlineare Regression**. Bei dem nichtlinearen Regressionsproblem handelt es sich um ein spezielles Optimierungsproblem, das mit Hilfe der Methode der kleinsten Quadrate gelöst werden kann [Alt02].

Für die Näherung der quantitativen (Performance-)Funktionsbeziehungen müssen zunächst Messdaten in einer kontrollierten Test-Umgebung erhoben werden. Dazu wird mittels eines eigenen Workload-Generators eine reproduzierbare Workload auf einer DB2-Datenbank simuliert und eine initiale Parameter-Einstellung vorgenommen. Gleichzeitig werden ausgewählte Metriken (bzw. Sensoren) als Kenngrößen für die Performance gemessen und abgespeichert. Die Parameter (bzw. Effektoren) werden nun schrittweise geändert und die Simulation inklusive der Datensammlung wird (unter gleichen Bedingungen) mehrmals wiederholt. Auf diese Weise können wir alle Änderungen der Performance-Metriken genauestens verfolgen.

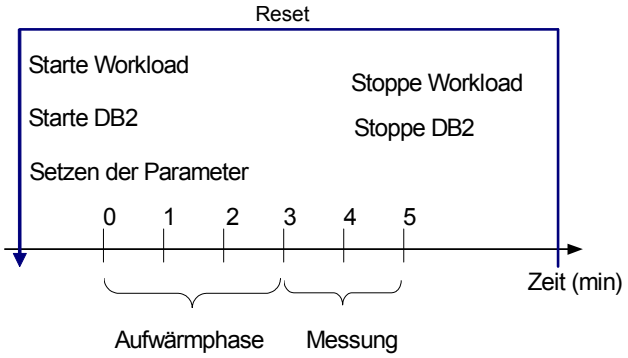


Abbildung 7.14: Messung zur Bestimmung quantitativer Abhängigkeiten (nach [Exn07])

Für jede der $|D_{P_1}| * |D_{P_2}| * \ldots * |D_{P_k}|$ Parameter-Werte-Kombinationen besteht der Messablauf grob aus den folgenden in **Abbildung 7.14** veranschaulichten Phasen. Nach einer dreiminütigen Aufwärmphase, in der die Bufferpools gefüllt werden und das System in einen stabilen Ausgangszustand gelangen soll, beginnt die eigentliche Messung. Zwei Minuten lang werden alle 5 Sekunden die Werte relevanter Metriken gesammelt. Nach Abschluss werden der Durchschnitt über die insgesamt 25 Messwerte gebildet und die

Datenbank-Instanz neu gestartet (*db2stop* und *db2start*), so dass der nächste Messzyklus mit den gleichen Ausgangsbedingungen anlaufen kann. Das in [Gan06] bereitgestellte und angepasste DET-Tool kann diesen Prozess automatisieren. In der Arbeit [Exn07] beschreiben wir im Detail die verwendeten Workloads, die Bereiche, für die Änderungen an der Konfiguration vorgenommen wurden sowie den konkreten in die Test-Umgebung eingebetteten Mess-Ablauf.

Aus den so gewonnenen Daten wird nun ein funktionaler Zusammenhang zwischen den geänderten Parametern und den Sensoren der Performance approximiert. Dazu erfolgt nach Abschluss der Messungen eine Analyse der Messdaten in Matlab [HaLi04], wobei im Unterschied zu [Sch05] mit Hilfe der nichtlinearen Regression auch und insbesondere *nichtlineare Zusammenhänge* Berücksichtigung finden. Die Daten können letztlich nach dem Aufstellen der Kriterien für eine „optimale" Performance für die Lösung unseres Optimierungsproblems verwendet werden. Für die mathematischen Einzelheiten sei an dieser Stelle auf [Exn07] verwiesen.

Da es in DB2 mehr als 100 konfigurierbare Parameter und weitaus mehr auswertbare Sensoren gibt, haben wir unsere Betrachtungen in [Exn07, RaWi07, WiRa09] auf einige elementare Bereiche, wie bspw. die Konfiguration von Bufferpools, des Locking und des Sorting, beschränkt. Zur Bestimmung der Effekte im und um den Bufferpool, wurden die Größe (BUFFERPOOL *SIZE*) und der DB2-Konfigurationsparameter NUM_IOCLEANERS[46] jeweils in einem festen Wertebereich geändert. Zur Charakterisierung der Bufferpool-Performance haben wir uns für die Metriken POOL_HIT_RATIO (BPHR) und „%_OF_ASYNC_WRITES" (POOL_ASYNC_DATA_WRITES / POOL_DATA_WRITES) entschieden. Letzterer, berechneter Performance-Sensor gibt den Prozentsatz derjenigen Schreib-Operationen an, bei denen geänderte Seiten des Bufferpools auf Festplatte durch Page Cleaner (bzw. I/O Cleaner) zurück geschrieben werden. Wir wissen bereits, dass die BPHR von der Größe des Bufferpools abhängig sein wird. Ebenso erwarten wir eine Korrelation zwischen der Anzahl an Page Cleaners und dem Prozentsatz der asynchronen Schreib-Operationen. Es gilt zudem festzustellen, ob auch die BPHR-Metrik von der Anzahl an Page Cleaners oder der Prozentsatz asynchroner Schreib-Operationen von der Bufferpool-Größe abhängen (siehe **Abbildung 7.15**).

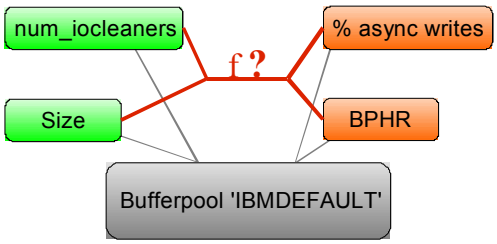


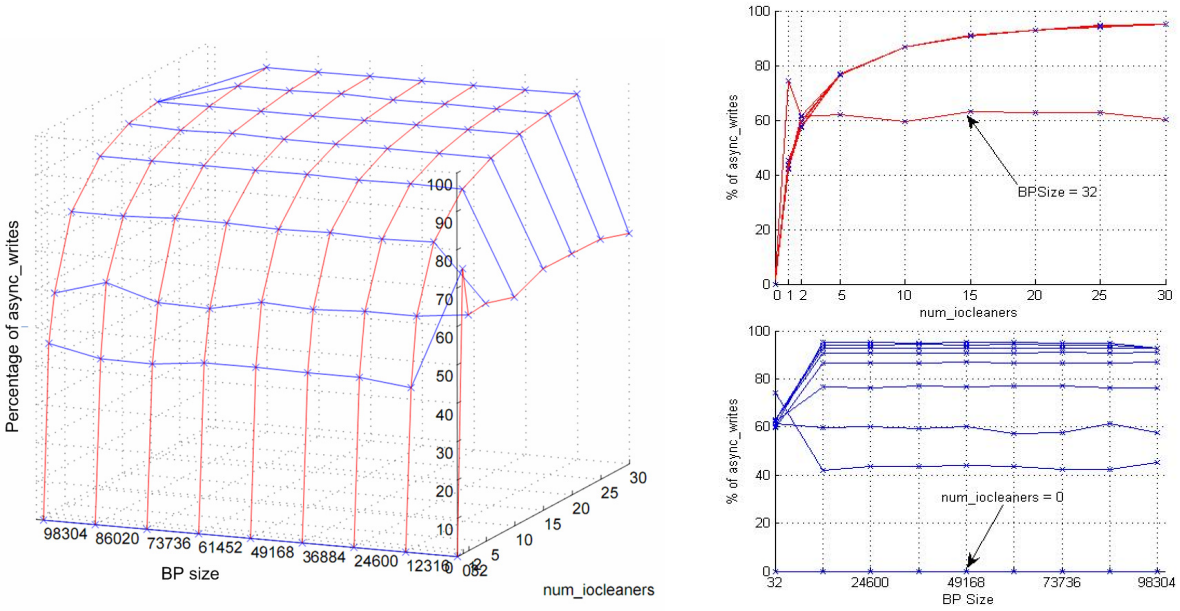
Abbildung 7.15: Potentielle Abhängigkeiten des Bufferpools (nach [RaWi07])

In einem ersten Schritt wurden die Einflüsse der beiden Konfigurationsparameter auf die *BPHR* hin untersucht. Die in [Exn07] ausführlich veranschaulichten Ergebnisse bestätigen die Vermutung, dass die Performance-Metrik unabhängig von der Anzahl der verwendeten Page Cleaner ist, aber wie erwartet von der Größe des Bufferpools abhängt. Mit zunehmender Größe erhöht sich auch die Zahl der Seiten, die (asynchron) in den Bufferpool geladen werden können. Somit steigt die Wahrscheinlichkeit, dass sich bei der

[46] Über den Datenbank-Parameter *NUM_IOCLEANERS* kann die Anzahl an Page Cleaners (I/O Cleaner) kontrolliert werden, die asynchron nicht mehr benötigte, geänderte Seiten aus dem Bufferpool auf Platte herausschreiben.

Ermittlung eines Anfrage-Ergebnisses eine Seite bereits im Bufferpool befindet, und nicht erst ein teurer physischer Zugriff von Platte erfolgen muss.

Die Resultate der in einem zweiten Schritt durchgeführten Untersuchungen für den *Anteil der asynchronen Schreib-Operationen* zeigt **Abbildung 7.16** in Form einer 3-D-Darstellung und den entsprechenden zweidimensionalen Projektionen. Die untere Projektion der **Abbildung 7.16b** verdeutlicht, dass die BUFFERPOOL SIZE für jede verwendete Anzahl von Page Cleaners ab einer bedeutsamen Größe konstant bleibt. Folglich hat die Größe des Bufferpools keinen Einfluss auf die Asynchronität der Schreib-Operationen. Es bestätigt sich hingegen der vermutete Einfluss des Parameters NUM_IOCLEANERS für die Metrik, einen genügend großen Bufferpool vorausgesetzt. Auch in diesem Fall liegt keine einfache lineare Abhängigkeit zwischen Effektor und Sensor vor, wie deutlich in **Abbildung 7.16a** und **Abbildung 7.16b** (oben) ersichtlich ist.



(a) 3D-Darstellung (b) 2-D-Projektionen

Abbildung 7.16: Mess-Ergebnisse für den *Anteil asynchroner Schreib-Operationen* (nach [RaWi07])

Je mehr Page Cleaner zur Verfügung stehen, desto mehr geänderte Seiten können asynchron vom Bufferpool auf Festplatte geschrieben werden. Aufgrund weniger Page Cleaner gibt es im Bufferpool auch eine geringere Zahl freier Seiten. Daher müssen durch die DB2-Prozesse geänderte Seiten unerwünschterweise synchron auf Platte geschrieben werden. Die Asynchronität der Schreib-Operationen erhöht sich demnach durch die Verwendung zusätzlichen Page Cleaner, kann jedoch nicht beliebig gesteigert werden, da sich die Stärke des Einflusses bis zum Erreichen einer Sättigung verringert. Werden mehr Page Cleaner verwendet, so erhöht sich auch der Aufwand für deren Verwaltung. Die steigende Zahl der damit zu koordinierenden parallelen Prozesse, schlägt sich wiederum in einer erhöhten CPU-Auslastung nieder.

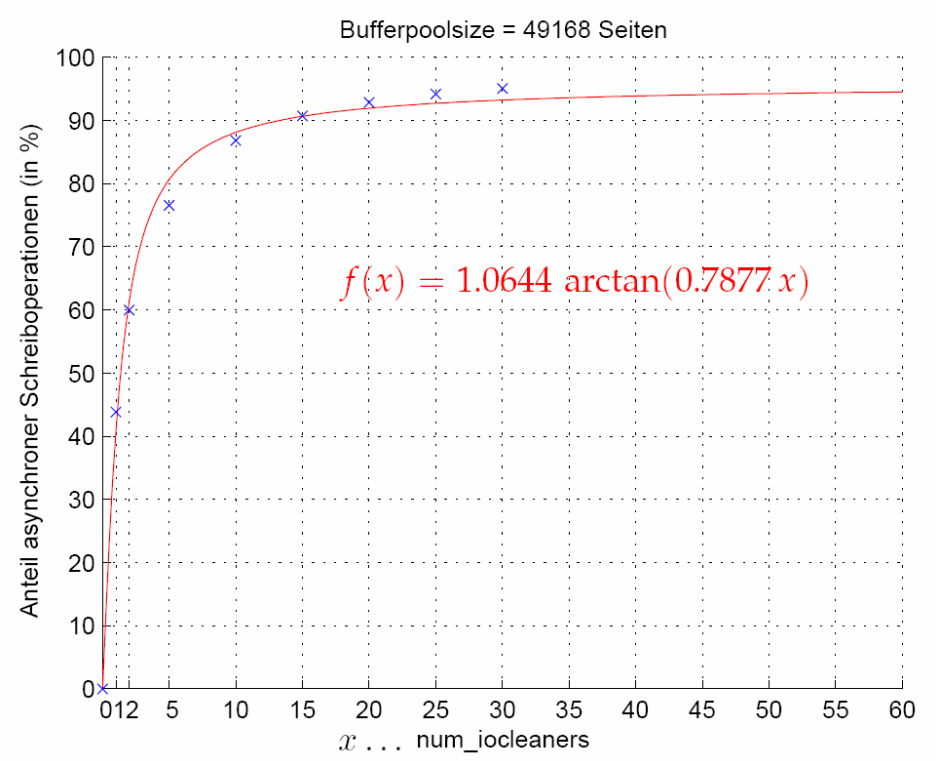


Abbildung 7.17: Resultat der nichtlinearen Regression nach Gauss-Newton (nach [RaWi07])

Für die Messdaten der Konfiguration des Bufferpools werden geeignete (Performance-) Funktionen zur quantitativen Beschreibung der Abhängigkeiten mit Hilfe der Methoden der nichtlinearen Regression gesucht. Zur Lösung werden die Algorithmen der Optimization Toolbox von Matlab [Mat04], einem Matrix-basierten, interaktiven System zur numerischen Berechnung und graphischen Darstellung der Lösung technisch-wissenschaftlicher Probleme, verwendet. Um nun einen passenden funktionalen Zusammenhang zwischen dem Parameter NUM_IOCLEANERS und dem Sensor zum *Anteil asynchroner Schreibopeartionen* mittels nichtlinearer Regression zu finden, wird zunächst jedoch eine geeignete Ansatzfunktion benötigt. Die sich aus dem Verlauf der Effektor-Sensor-Wertepaare ergebenden Anforderungen an die monoton wachsende Funktion werden zusammen mit einer Liste an potentiellen Kandidaten umfangreich in [Exn07] ausgeführt. Wir haben uns an dieser Stelle zur Approximation des empirisch gewonnenen Verlaufs für eine Funktion(sklasse) vom Typ *f(x) = a*arctan(b*x)* entschieden. Zur Bestimmung der Parameter *a* und *b* von *f(x)* wird das Gauß-Newton-Verfahren nach der Methode der kleinsten Quadrate angewendet. **Abbildung 7.17** stellt den resultierenden funktionalen Zusammenhang für die gemessenen Wertepaare des Parameters NUM_IOCLEANERS und des zugehörigen Sensors *Anteil asynchroner Schreib-Operationen* anschaulich dar.

Nun stellt sich die Frage, welcher Wert für NUM_IOCLEANERS im Sinne einer optimalen Konfiguration zu wählen ist. Dafür lässt sich ein weiteres Optimierungsproblem im Sinne der zu maximierenden Performance-Funktion und der Einhaltung gewisser Nebenbedingungen (in diesem Fall z.B. begrenzter Speicher, Grad der CPU-Auslastung etc.) aufstellen.

Wird der Parameterwert sehr klein gewählt, so liegt der resultierende Sensorwert unter nicht zufriedenstellenden 80%. Wählt man NUM_IOCLEANERS größer als 25, so hat die Hinzunahme eines weiteren Page Cleaner nur noch einen sehr geringen Anstieg des Sensors aber eine enorme Zunahme der Kosten für die Verwaltung der asynchronen

Prozesse zur Folge. Um eine möglichst performante Prozessaufteilung und CPU-Auslastung zu gewährleisten, ist man daher daran interessiert, die Zahl der Page Cleaner nicht größer als nötig zu wählen.

Um über ein Optimierungsproblem die „passende" Anzahl an Page Cleaners zu finden, eignen sich u.a. die folgenden drei Entscheidungskriterien (bzw. Nebenbedingungen) [Exn07]:

- „Wähle den Wert für NUM_IOCLEANERS möglichst groß. Der Anteil der asynchronen Schreib-Operationen soll dabei bei der Hinzunahme eines weiteren Page Cleaner um mindestens 1% wachsen". Das Anwenden des Optimierungsproblems führt zu einem zu selektierenden Wert von 8 für den Parameter. Der zugehörige Sensorwert liegt dann bei 86,198%.
- „Wähle den Wert für NUM_IOCLEANERS so, dass der Wert des Sensors 90% oder 95% des Maximalwertes der Funktion f beträgt". Da die Zahl der num_iocleaners begrenzt ist, gibt es einen Maximalwert für die ansonst monoton steigende Funktion f. Dieser liegt bei 95,796%. Für einen Sensorwert in Höhe von 90% des Maximalwertes werden 9 Page Cleaner benötigt, für 95% muss der Wert NUM_-IOCLEANERS auf 17 gesetzt werden.
- „Wähle den Wert für NUM_IOCLEANERS so, dass der Anteil asynchroner Schreib-Operationen größer als 90% ist". Um diese in [HaGu06] aufgestellte, erfahrungsbasierte Forderung zu erfüllen, muss NUM_IOCLEANERS mindestens 14 betragen.

Jedes der unterschiedlich restriktiven Kriterien liefert eine andere Lösung und damit ein anderes Kosten-Nutzen-Verhältnis. Darüber hinaus sind noch weitere Kriterien als Grundlage für die Festlegung einer endgültigen Konfiguration denkbar.

7.5.3 Zusammenfassung und Ausblick

Schon Benoit beschreibt in seiner Dissertation [Ben03] und in [Ben05] einen Algorithmus zur automatischen Problem-Diagnose von Performance-Engpässen. Ausgehend von einem Ressourcen-Modell wird ein Diagnosebaum entwickelt, dessen Aussagen bezüglich der Problem-Ressourcen genutzt werden, um durch schrittweise Anpassung dieser Ressourcen die Gesamt-Performance des Systems zu verbessern. Auf diese Weise ist möglich, das Tuning inkrementell vorzunehmen, indem schrittweise einzelne Parameter geändert werden. Das Aufstellen des Diagnosebaumes ist jedoch sehr System-spezifisch und beruht auf vielen Heuristiken. Zur Daten-Erhebung wurde immer genau ein DB2-Konfigurationsparameter variiert (die anderen Einstellungen wurden mit Default-Werten belegt) und das resultierende Systemverhalten beobachtet. Im Gegensatz zu unserem Ansatz sind die ermittelten Abhängigkeiten in Benoits Ressourcen-Modell rein qualitativer Natur, da keine funktionale Zusammenhänge zwischen Effektoren und Sensoren in Betracht gezogen werden.

Unsere Arbeiten [Gan06, Exn07, RaWi07, WiRa09] haben hingegen konkrete Möglichkeiten zur Modellierung, Speicherung und zum automatisierten Bestimmen qualitativer sowie quantitativer (nicht-linearer) Abhängigkeiten zwischen mehreren Effektoren und mehreren Sensoren aufgezeigt. Darüber hinaus verhilft die Auswertung der empirisch gewonnenen Messdaten für verschiedene Workloads zur Ermittlung einer optimalen Konfiguration mittels numerisch-analytischer Techniken der nichtlinearen Optimierung. In den einfachen, betrachteten Szenarien liefert die Optimierung auf der Basis von Peformance-Funktionen unverzüglich (in einem Schritt) die gleichen Ergebnisse wie

Tuning-Ansätze, die ein eher langwierigeres, iteratives Vorgehen verfolgen und auf Erfahrung beruhen. Erhöht sich die Anzahl an Effektoren, so bieten Optimierungsverfahren die bessere Alternative zu einfachen Heuristiken, die jeden Effektor nacheinander durchprobieren. Während der Feedback-Ansatz demgegenüber sehr wenig Overhead erzeugt, erfordert die Daten-Sammlung, Regressions-Analyse sowie das Aufstellen und Lösen des Optimierungsproblems einen nicht unwesentlichen Anteil an Zeit und Berechnungsaufwand.

Die Komplexität und die Zahl der Seiten-Effekte steigen überdies erheblich mit einer wachsenden Anzahl an Effektoren und den sich ändernden System- bzw. Workload-Bedingungen. Daher sollte bereits zu Beginn eine Einschränkung der zu untersuchenden Parameter getroffen und das System zur Aufdeckung potentieller Seiten-Effekte kontinuierlich überwacht werden. Unser in Abschnitt 6.3 betrachtetes Performance Warehouse als zentrale Anlaufstelle von Performance-Daten (in einem MAPE-Kreislauf) kann sowohl bei der Überwachung und Kontrolle der Auswirkungen, als auch bei der Speicherung der qualitativen und quantitativen Zusammenhänge unterstützen.

Wie wir bereits in Abschnitt 4.4 aufgeführt haben, können die Informationen über (qualitative und quantitative) Abhängigkeiten in vielerlei Hinsicht nützlich für die Analyse- und Plan-Phase einer MAPE-basierten Tuning-Architektur sein. Dadurch lassen sich u.a. Tuning-Pläne in ihren (semantischen) Auswirkungen und Seiten-Effekten besser evaluieren bzw. verstehen. Konfligierende Pläne können (einfacher) erkannt und Ursachen von Problemen schneller identifiziert werden.

Neben der retrospektiven Evaluation erscheint uns auch die Vorhersage über den Erfolg bzw. die (Seiten-)Effekte von Tuning-Plan-Ausführungen ein näher zu ergründendes Aufgabengebiet. Auf der Grundlage einer solchen Erkenntnis könnten jeweils die Tuning-Pläne ausgewählt werden, welche den größten Erfolg versprechen. Hierzu muss jedoch zunächst untersucht werden, wie der zukünftige Erfolg eines Tuning-Plans bestimmt werden kann.

Basierend auf den Performance-Funktionen können bspw. Kosten- und Nutzenbewertungen einzelner Konfigurations-Änderungen und damit erneute Optimierungsprobleme aufgestellt werden. Da Tuning-Pläne im Grunde Sequenzen einzelner Tuning-Aktionen, sprich Effektor-Anwendungen darstellen, verursachen auch sie Kosten. Diese können im Hinblick auf die Tuning-Plan-Selektion oder -Priorisierung in der Plan-Phase zusätzlich ausgewertet und berücksichtigt werden. In der Regel wird man versuchen, die Sequenz an Tuning-Aktionen zu finden/selektieren, welche den Gesamtprofit, unter Berücksichtigung der Kosten, maximiert. Aber auch die beiden Extremfälle des maximalen Nutzen bzw. der minimalen Kosten sind denkbare Alternativen.

Durch den verursachten Aufwand zur Bestimmung und Auswertung der Messwerte ist es obendrein schwerlich möglich, das Abhängigkeits-Modell erst zur Laufzeit füllen zu wollen. Um daher mit einer gewissen Vorlaufzeit die Wechselbeziehungen ermitteln zu können, ist das gesamte zu betrachtende System in einen (Offline-)Zustand zu bringen, der dem zur Laufzeit zumindest nahe kommt. Es müssen demnach nicht nur alle für den realen Datenbankbetrieb benötigten Datenbank-Objekte vorhanden sein und ein vergleichbarer Datenbestand simuliert werden. Darüber hinaus sind Workload-Szenarien zu simulieren, die vorausschauend die tatsächliche Systemlast zur Laufzeit repräsentieren. Da eine reale Workload gewissen Schwankungen unterliegt, bietet sich hierfür die in Abschnitt 7.4 eingeführte Klassifizierung an. Damit können die voraussichtlich häufigsten Workload-Szenarien abdeckt und anhand derer die Abhängigkeitsfunktionen ermittelt werden. Es kann allerdings zur Laufzeit nötig sein, sich über eine inkrementelle Anpassung des Modells

zügig auf die neuen Probleme bzw. die dynamischen System- und Workload-Bedingungen einzustellen.

Neben der recht umfangreichen Datensammlung stellt sich die Wahl geeigneter Ansatzfunktionen für die Regression als weiteres Problem heraus heraus, wobei mehrmals ein stückweise linearer Ansatz gute Ergebnisse lieferte [Exn07]. Die Erfahrungen der Datenbank-Administratoren sind damit weiterhin unentbehrlich. Selbst dann ist es schwierig, gute manuelle Approximationen zu finden. Um den Prozess des Auffindens einer geeigneten Funktion und damit auch das nachfolgende Optimierungsproblem zu unterstützen bzw. gar zu automatisieren, kann man zunächst eine Menge geläufiger Funktionskandidaten vorgeben und jenen mit der passendsten Approximation auswählen (lassen). Faustregeln für eine grobe Orientierung der Parameterwahl würden wohl aufgrund der Komplexität der DBMS häufig ebenso weiterhelfen. Die Methode der (multiplen) linearen Regression eignet sich bspw. für eine solche Aufgabe, bei der man einen gewissen Grad an Ungenauigkeit in Kauf nimmt, indem man stückweise lineare Zusammenhänge betrachtet [Exn07]. Im Fall der multiplen linearen Regression, können im Gegensatz zur linearen Regression nicht nur einer sondern mehrere Effektoren Einfluss auf einen Sensor haben. Um grob herauszufinden, inwieweit ein Effektor einen Sensor qualitativ beeinflusst, kann ein statistischer Test auf Signifikanz durchgeführt werden [Exn07]. Diese Testmethode stellt also neben dem DET-Tool eine weitere Möglichkeit dar, Abhängigkeiten zwischen Effektoren und Sensoren zu untersuchen, da den DBA häufig nur interessiert, ob es überhaupt einen (vermuteten) Zusammenhang gibt. Aufbauend auf diesem etwas gröberen Ansatz könnten sich weitere Untersuchungen und eine etwaige Automatisierung (zur Suche nach Abhängigkeiten zwischen mehreren Effektoren und einem Sensor) anschließen. Ist man an möglichst guten Performance-Funktionen ohne Verringerung der Genauigkeit interessiert, um daraus weitere Optimierungsprobleme herzuleiten, so benötigt man weiterhin nichtlineare Regression.

Im Rahmen weiterführender Arbeiten sind auch die Untersuchungen hinsichtlich Konfigurations- und Tuning-Szenarien auszuweiten. Des Weiteren sind Evaluierungen unter verschiedenen, dynamischen Workloads angedacht. Möglicherweise lässt sich daraus eine Klassifikation der Performance-Funktionen bezüglich der Workload ableiten. Ist eine Abhängigkeit erst einmal ermittelt und im Modell enthalten, kann diese im Zuge von Tuning-Maßnahmen zur Entscheidungsfindung herangezogen werden. Es muss jedoch überprüft werden, unter welchem Workload-Einfluss die Funktion zustande gekommen und demnach für welche Workload sie gültig ist. Anschließend kann die Funktion Anwendung finden, deren zugewiesene Workload am ehesten der aktuellen Systemlast entspricht.

Im Vordergrund sollten jedoch Betrachtungen des Nutzens und der mögliche Integrierbarkeit in unseren Prototypen zum autonomen Datenbank-Tuning stehen. Anhand dieser Ergebnisse sollte dann abgeschätzt werden können, ob der gewählte Ansatz, das Datenbank-Tuning unter Verwendung nichtlinearer Optimierungsprobleme vorzunehmen, eine Verbesserung gegenüber der „herkömmlichen“ iterativen Vorgehensweise darstellt. Auch eine Kombination der beiden Verfahren erscheint aussichtsreich. Mit Hilfe der berechneten optimalen Lösung auf Basis der approximierten Performance-Funktionen wählt man eine Konfiguration aus. Von hier aus versucht man nun die gefundene Konfiguration des DBMS schrittweise mit Best-Practices-Heuristiken noch zu verbessern. Stets ist dabei zu beachten, dass die Komplexität des Tuning-Problems überschaubar und damit eine spätere automatische Umsetzung realisierbar bleibt.

Kapitel 8

Prototypische Perfomance-Monitoring- und -Tuning-Architektur

Um ein auf dem Kontroll-Schleifen-Paradigma basierendes autonomes Datenbank-Tuning zu ermöglichen, sind eine Monitoring-Umgebung und eine entsprechende Datenbasis als Grundlage für die weitere Analyse, Planung und Entscheidungsfindung erforderlich. Das vorliegende Kapitel versucht sich an einer inhaltlichen und technischen Verknüpfung der drei in den Kapiteln 5, 6 und 7 vorgestellten Phasen zur Sammlung, Speicherung und Anwendung von Performance-Daten.

Nachdem wir in **Abschnitt 8.1** auf einige grundlegende allgemeine Anforderungen einer erweiterten Monitoring-Infrastruktur eingehen, soll in dem sich anschließenden **Abschnitt 8.2** zunächst das Lösungskonzept, mit dem ein durchgängiges und umfassendes Datensammlungs- und Zuführungs-Framework zur Anwendung von DWH-Konzepten auf das Performance-Monitoring und -Tuning bereitgestellt wird, vorgestellt werden. Die beschriebene Monitor-Architektur wendet die in den drei vorangegangenen Kapiteln vorgestellten Konzepte an, um aus einer Menge von verschiedenen operativen Systemen Performance-Daten zu extrahieren, in einem zentralen Performance Warehouse (PWH) konsistent und langfristig zu speichern und zur Unterstützung sowohl der Administratoren als auch des autonomen Datenbank-Tunings adäquat bereitzustellen. **Abschnitt 8.3** umfasst schließlich unseren konkret implementierten Performance-Tuning-Prototypen, den Autonomic Tuning Expert (ATE). Dieser kombiniert zentrale Konzepte des aus der Kontrolltheorie bzw. dem Autonomic Computing bekannten Feedback-Ansatzes mit der Idee der Formalisierung bewährter Tuningpraktiken und der System- bzw. Tuning-unabhängigen Klassifikation der Workload, um ein Datenbank-basiertes Softwaresystem autonom zu tunen. Wir deuten zudem in **Abschnitt 8.4** an, wie das erweiterte Monitoring in die vorhandene ATE-Infrastruktur integriert und den Komponenten bzw. dem Administrator zugänglich gemacht werden kann. Wir legen dar, welche Änderungen und Erweiterungen der bestehenden Infrastruktur für diesen Zweck notwendig erscheinen, um eine fundierte Basis für sowohl sämtliche manuellen als auch autonomen Analysen und Entscheidungen zu schaffen.

8.1 Allgemeine Anforderungen an die Monitoring-Infrastruktur und das Tuning-System

Das Ziel der zu konzipierenden Datensammlungs- und Zuführungs-Infrastruktur (und einer darauf aufbauenden Tuning-Umgebung) liegt in der Modellierung, Gewinnung, Speicherung, Verwaltung (und Anwendung) von unterschiedlichsten Performance-Daten sowie Kontext-Informationen der Ressourcen eines Software-Stack. Die Daten dienen dabei der Analyse, Planung und Entscheidungsfindung als Grundlage für das (automatische) Erken-

nen, Diagnostizieren und anschließende Auflösen potentieller Performance-Probleme ohne nennenswerte Störung des operativen Systems.

Im Folgenden seien die aus unserer Sicht wichtigsten Anforderungen an eine solche Performance-Monitoring-Infrastruktur (**PMI**) als Grundlage des (autonomen) Performance-Tunings zusammengefasst. In den Kapiteln 5, 6 und 7 haben wir in Form von Anforderungen, Konzepten und Produktvorstellungen bereits ausführlich kennengelernt, wie Daten aus verschiedenen Quellen **erfasst**, einheitlich abgebildet und an zentraler Stelle (zugreifbar) **verwaltet** sowie im Rahmen autonomer Prozesse bzw. manuell durch den DBA letztlich sinnvoll zur Laufzeit **eingesetzt** werden können. Die Trennung bzw. Zerlegung in die einzelnen Phasen der Datenentstehung, Datenbeschaffung, Datenhaltung sowie -verwaltung erlaubt eine größere Flexibilität und Effizienz sowie insbesondere die präzise Kontrolle über den mit der Datensammlung und -zulieferung assoziierten Overhead. Es sind somit bspw. Szenarien denkbar, in denen das Abbild über den (Gesundheits-)Zustand des Systems in regelmäßigen Intervallen (z.B. alle 15 Sekunden) gesammelt, aber die entsprechenden Daten - primär aus Aufwandsgründen - nur etwa jede Stunde aggregiert und übertragen werden.

Zu den (teilweise voneinander abhängigen) **Anforderungen**, die eine ideale Performance-Monitoring- und Tuning-Infrastruktur zu erfüllen hat, zählen:

- Monitoring als Fundament für das Tuning: Die PMI soll die **Entscheidungsprozesse** im Rahmen des manuellen und des autonomen Datenbank-Tuning verbessern und beschleunigen. Sie soll damit sowohl die grundlegende Voraussetzung für Analysen durch die Administratoren als auch für einen Autonomic Tuner darstellen.
- **Umfassende Datenbasis**: Als Grundlage einer Performance-Monitoring- und Tuning-Architektur soll eine Datenbasis dienen, in der neben den Performance-Daten über den operativen Betrieb sowohl Informationen zu den Experten-Strategien (Problem-Analyse und Performance-Tuning) hinterlegt als auch beim Ablauf einer Experten-Analyse entstehende Ergebnis-Daten gespeichert werden. Somit kann die Analyse- bzw. Tuning-Ausführung zusammen mit ihren Ergebnissen jederzeit nachvollzogen werden. Zu den Informationen über eine Problem-analysierende bzw. -behebende Tuning-Maßnahme zählen dabei nicht nur Beschreibungen der einzelnen Analyse- bzw. Tuning-Schritte sowie die Namen der von ihnen verwendeten Sensoren und Effektoren, sondern auch die Reihenfolge der Schritte und die Hierarchie ihrer Anordnung. Ebenso müssen in dieser Datenbasis auch Angaben über die Eingabe- und Ausgabe-Objekte einzelner Teilschritte enthalten sein (siehe Abschnitt 4.2.2).
- Mechanismen zum Umgang mit **Komplexität** und Heterogenität: Performance-Daten stammen aus verteilten Umgebungen und deren vielfältigen, komplexen, heterogenen Software- und Hardware-Komponenten. Eine konkrete Architektur soll diese Vielschichtigkeit adäquat abbilden und verwalten können.
- **Automatisierung der wichtigsten Abläufe**: Die Sammlung, Übertragung, (Transformation), Speicherung und vor allem Auswertung der Daten auf verschiedenen Abstraktions-Ebenen soll zur Entlastung (ohne Eingriffe) der Administratoren erfolgen. Diese Forderung schließt nicht aus, dass der Administrator die Zeitpunkte und den Umfang der einzelnen Aktivitäten steuern und (zeitlich) planen kann.
- **Zuverlässigkeit**: Das Monitoring-System mit all seinen Komponenten zur Sammlung bzw. Weiterleitung von Performance-Informationen ist verantwortlich für je-

des Subsystem und für die zuverlässige Bereitstellung der kritischen Informationen an die verantwortlichen Instanzen.

- Einsatz von Standards / **Interoperabilität**: Der Umgang mit unterschiedlichsten Quellen und Formaten in unternehmensweiten System-Architekturen legt den Einsatz verbreiteter Tools, Mechanismen und vor allem Standards zum Datenaustausch und für die Schnittstellen nahe. Interoperabilität sollte nicht nur auf Basis der Datenquellen, sondern auch durch eine einheitliche Kommunikation der einzelnen Monitoring- und Tuning-System-Komponenten untereinander gewährleistet werden.
- **Skalierbarkeit**: Potentiell können tausende Ressourcen bzw. Applikationen überwacht werden, die Daten an tausende von Empfänger-Entitäten liefern können. Die Sammlung und Übertragung der Daten (über das Netzwerk) muss mit der zunehmenden Zahl der Entitäten sowie dem im Laufe der Zeit steigenden Datenwachstum skalieren. Ein effizienter Umgang mit großen Datenvolumina ist demnach unabdingbar. Sowohl das Monitoring als auch das Tuning müssen letztlich mit der hohen Geschwindigkeit bzw. mit der Rate, in der Performance-Daten gesammelt und generiert werden können, umzugehen wissen.
- Geringer **Overhead** (Performance-Einbußen): Der Monitoring- und Speicher-Aufwand sollen gering gehalten werden und idealerweise vorhersehbar bzw. stabil gegenüber Schwankungen sein. Die schnell anwachsenden Daten-Mengen, die durch Daten-Sammlung und Speicherung entstehen, müssen handhabbar gemacht werden. Dafür sind Mechanismen zur **Aggregation** (und **Archivierung)** erforderlich.
- **Hochverfügbarkeit**: Die PMI soll ständig für kontinuierliche Analysen zur Verfügung stehen und auch zu lastreichen Spitzenzeiten nicht ausfallen.
- **Daten-Qualität** und -**Aktualität**: Neben einem Analyse-bezogenen Umgang mit den Daten fordern wir die Sicherstellung einer vorgegebenen Qualität sowie für die Auswertungen notwendige Aktualität bzw. Verfügbarkeit der Daten.
- Umgang mit Dynamik (**Adaptivität/Selbst-Konfiguration**): Bei der Überwachung und Konfiguration von IT-Systemen ist die Berücksichtigung ihrer Dynamik, maßgeblich durch sich ändernde Zugriffsprofile bzw. Arbeitslasten und geänderte Ziele oder Rahmenbedingungen, unabdingbar. Damit die der Überwachung nachfolgende Optimierung jeder Art von Workload gerecht wird und auch im Falle von Störungen bzw. Änderungen in der Umgebung akzeptable Performance garantiert, muss bereits die Überwachungs-Infrastuktur in der Lage sein, sich kontinuierlich an veränderte Rand- und Umgebungsbedingungen anzupassen. Bei geänderter Arbeitslast müssen bspw. neue Kompontenen überwacht, die Häufigkeit der Datensammlung reduziert oder auch neue Tuning-System-Komponenten hinzugeschaltet werden. Zusammengefasst soll die PMI die von ihr gesammelten Daten auch dazu nutzen, sowohl die eigene Ausführung, die zugrunde liegenden (Workload- und System-)Modelle und Regeln, als auch die eigenen Ressourcen im Angesicht der dynamisch ändernden Bedingungen anzupassen.
- **Lernfähigkeit**: Das System soll in der Lage sein, aus den eigenen Erfahrungen und der Historie zu lernen und das eigene Verhalten dadurch anzupassen bzw. zu verbessern. Die Fähigkeit zum Lernen ist als grundsätzliche Voraussetzung für die Selbst-Konfiguration zu sehen.

- **Fehlertolerante System-Komponenten:** Derartige Komponenten erlauben ein automatisches Ausführen adäquater (Reparatur- bzw. Heilungs-)Maßnahmen bei Ausfall von Teil-Komponenten bzw. im Falle von Fehlern, um den stabilen Betrieb dauerhaft zu gewährleisten (Selbst-Heilung). D.h. bei einem Ausfall sollten sich bspw. die Komponenten neu starten, ihre Daten replizieren bzw. die Verbindungen wieder neu aufbauen.
- (Semi-)**Automation & Nutzbarkeit**: Das System soll durch Schnittstellen bzw. intuitive Oberflächen einfach und ohne größere Kenntnis bedienbar sein. Nutzer-Eingaben sind zu reduzieren und nur im wirklich notwendigen bzw. gewünschten Fall einzubeziehen. Darüber hinaus soll über die GUI präsentiertes (Zusatz-)Wissen (insbesondere auf der Analyse-Ebene) einen Lern-Effekt vermitteln und den DBAs bspw. helfen, die Zusammenhänge zwischen den Ressourcen zu verstehen und bei ihren Überlegungen zu berücksichtigen.
- Berücksichtigung von **Verzögerungen**: Ursache und Wirkung einer Aktion fallen häufig zeitlich auseinander und können u.U. nicht mehr ohne weitere Annahmen einander zugeordnet werden. Dies ist insbesondere bei der Sammlung von Metriken und der Ermittlung der (oft verzögerten) Wirkungsweisen von Effektor-Anwendungen von Bedeutung.
- Berücksichtigung von **Policies**: Eine Monitoring- und Tuning-Architektur muss sich Policy-Forderungen anpassen und in ihrer Funktionsweise von den DBA-Richtlinien „steuern" lassen. Es ist denkbar, dass die Policy (nach Interpretation) vorgibt, welche Performance-Metriken welcher Ressourcen in welchen Abständen zu sammeln sind und auch wie bzw. in welchem Rahmen das Tuning zu erfolgen hat.

Wir wollen in den nachfolgenden Abschnitten eine Performance-Monitoring-Infrastruktur aufzeigen, welche diese Anforderungen gezielt berücksichtigt und damit sowohl als Grundlage für manuelle Analysen durch Administratoren als auch für eine autonome Tuning-Architektur gesehen werden kann.

8.2 Eine Performance-Monitoring-Infrastruktur (PMI)

Die PMI soll ihren Zweck sowohl als Kernkomponente der gesamten Autonomic-Tuning-Infrastruktur, als auch als Grundlage für alle manuellen, administrativen Entscheidungen erfüllen. Das integrierte PWH dient hierbei als zentraler Punkt für die Daten-Sammlung und -Verwaltung über verschiedene Ressourcen und Applikationen in einem Datenbank-basierten Software-Stack hinweg.

Die wesentlichen Funktionen einer Performance-Monitoring-Infrastruktur sind dementsprechend:

1. Sammlung und Integration aktueller Werte zu vordefinierten Performance-Indikatoren.
2. Transformation, Berechnung und Aggregation ablauforientierter und statistischer Kennzahlen.
3. Bereitstellung von Werkzeugen zur flexiblen, multidimensionalen Analyse und Navigation innerhalb der Analyse-orientierten Performance-Daten.
4. Verteilung, (grafische) Präsentation und Verwertung der Analyse-Ergebnisse.

Abbildung 8.1 veranschaulicht grob das geplante Vorgehen zur Daten-Vereinheitlichung und -Zentralisierung sowie die daraus resultierenden Performance-Daten-Ebenen in Anlehnung an eine Data-Warehousing-Umgebung (siehe Abschnitt 2.3). Konkret unterscheiden wir (von links nach rechts in Abbildung 8.1) die *operative Ebene*, die *Datengewinungs-Ebene*,

die Ebene der zentralen *Speicherung und Verwaltung* von Performance-Daten sowie die *Analyse*-Ebene.

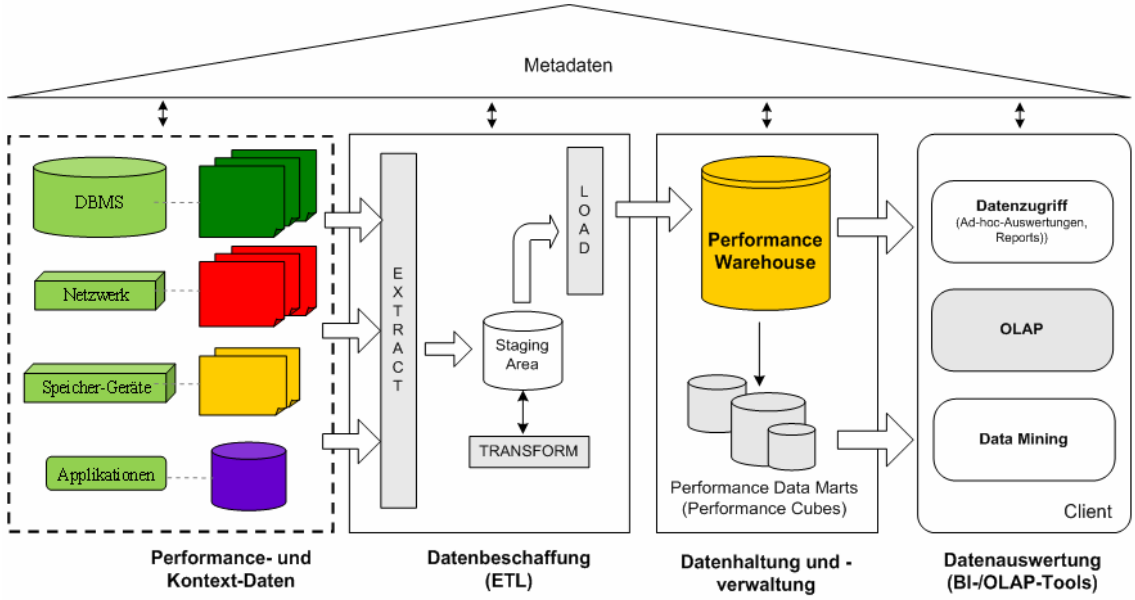


Abbildung 8.1: Schema einer Performance-Monitoring-Infrastruktur

Das Ziel unseres Monitoring-Systems ist die zuverlässige Bereitstellung zeitnaher und akkurater Performance- und Kontext-Informationen ohne nennenswerte Störung des operativen Systems. Die Daten dienen als Grundlage für das (automatische) Erkennen, Diagnostizieren und anschließende Auflösen von potentiellen Performance-Problemen.

Die Adressierung der Komplexität und Heterogenität wird dabei erzielt durch (1) eine adaptive Monitoring-Strategie zur Reduzierung der durch das Monitoring zusätzlich auf dem System erzeugten Last, (2) den Performance-Warehouse-Ansatz zur Langzeit-Speicherung sämtlicher Performance-Daten und zusätzlichen Kontext-Informationen in einer einheitlichen, zentralen Datenbank, (3) ein einheitliches Datenformat zur Formalisierung und Übertragung von Event-Daten sowie (4) dezentrale Monitoring-Agenten zur verteilten Gewinnung der Performance-Daten und Zuführung in das zentrale PWH.

Im Folgenden geben wir einen Architektur-Überblick unserer, den gestellten Anforderungen zu genügenden Performance-Monitoring-Infrastruktur zur Überwachung und Aufzeichnung des Systemzustands bzw. -verhaltens und der (relevanten) Workload. Die vorgestellte Konzipierung dient als abstrakte, von einer konkreten Implementierung unabhängige, Referenz-Architektur. Sie soll lediglich spezifizieren, welche Komponenten auf welche Art und Weise interagieren, um eine integrierte Performance-Monitoring- und -Tuning-Umgebung zu schaffen. Die Wahl geeigneter Daten-, Persistierungs- und Übertragungs-Formate obliegt einer konkreten Implementierung.

Die Autoren glauben, dass die nachfolgend beschriebene Architektur mit akzeptabler Performance und Skalierbarkeit implementiert werden kann. Heterogene Quellen verlangen nach einer einheitlichen Verwaltung von XML-, Datenbanken-, Dateisysteme- und OO-Metadaten. Der Zugriff auf die einzelnen, in unterschiedlichen Formaten vorliegenden Metadaten kann per (asynchronen) Dateiaustausch (MDIS, CDIF, CWM, OIM, XML), (snychronen) Application Programming Interface (ODBC, OLEDB,...) oder aber auch (synchron und asynchron) durch Metadaten-Wrapper, d.h. der Abbildung zwischen verschiedenen Metadaten-Repräsentationen erfolgen. Das Datenformat sollte hinsichtlich

Nutzbarkeit und Kompaktheit ausgewählt werden, um durch Hinzunahme wohldefinierter Schnittstellen und Daten-Übertragungsprotokolle den Einsatz gängiger, am Markt etablierter Tools zu ermöglichen. Ebenso sollten gängige Repository-Standards und Referenz-Architekturen für Repository-Systeme (z.B. IRDS, PCTE) in Betracht gezogen werden. Sowohl in Abschnitt 2.2.5 als auch in den Untersuchungen von [Böt07] finden sich nähergehende Anhaltspunkte für eine konkretisierende Technologiewahl.

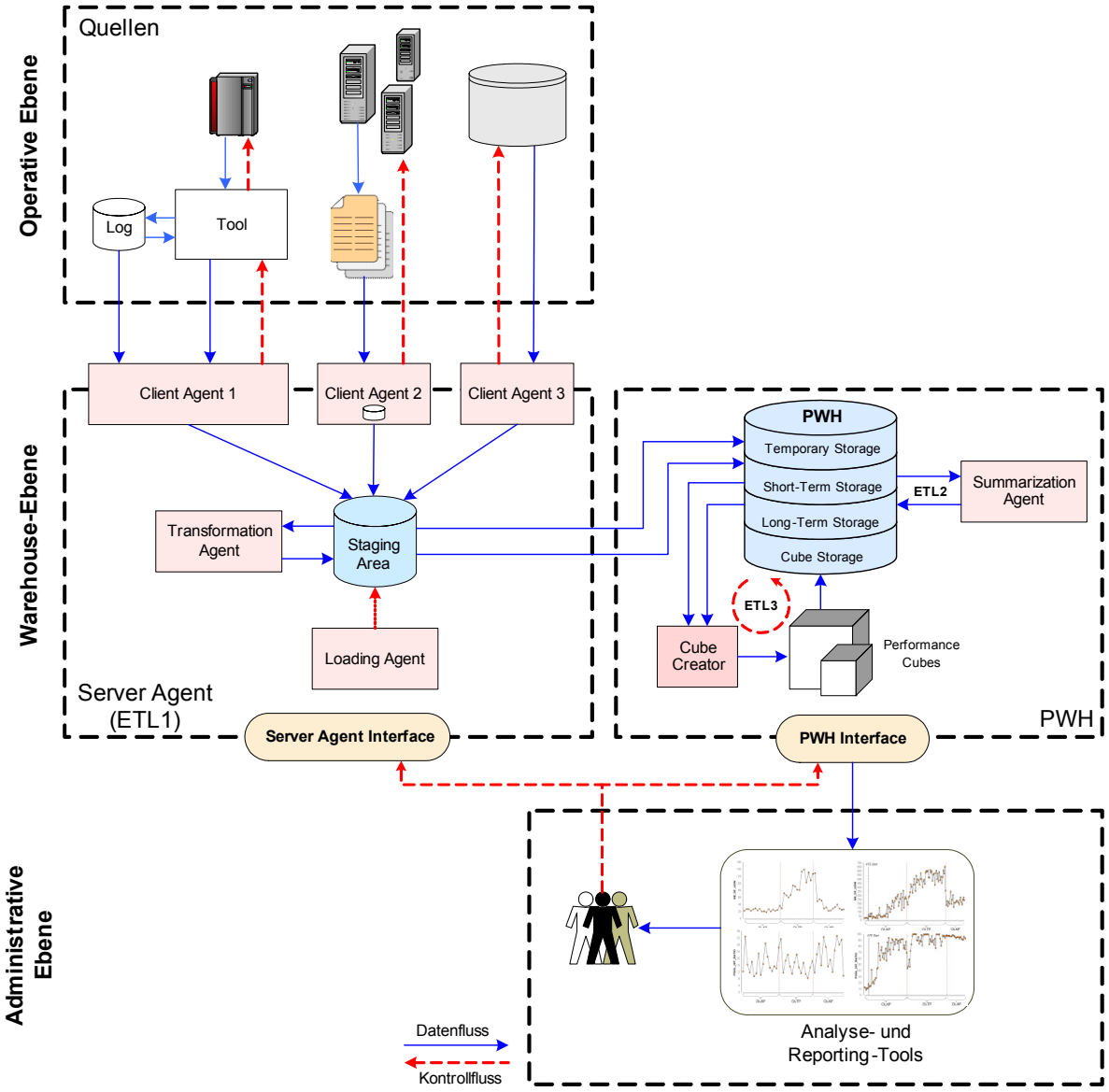


Abbildung 8.2: Unsere Monitoring-Umgebung im Überblick

Die Monitor-Architektur besteht, wie in **Abbildung 8.2** dargestellt, aus drei aufeinander aufbauenden Ebenen und orientiert sich i.W. an unseren Ergebnissen aus [Köh10]. Die **operative Ebene** versorgt das Performance Warehouse, das zentrale Konstrukt der **Warehouse-Ebene**, kontinuierlich mit Performance-Daten. Die **administrative Ebene** ermöglicht über entsprechende Schnittstellen schließlich dem Datenbank-Administrator bzw. den autonomen Mechanismen auf die Performance-Daten des PWH zuzugreifen sowie dessen Komponenten zu konfigurieren. Der Informations-Austausch zwischen den Ebenen und deren jeweiligen Elementen findet dabei auf Kontroll- und Datenflussniveau in die verschiedenen, durch Pfeile grob angedeuteten, Richtungen statt. Im nachfolgenden Verlauf werden wir insbesondere sehen, dass drei vermeintlich verschiedene ETL-Prozesse zur

Extraktion, Transformation und Überführung von Performance-Daten auf den verschiedenen Ebenen der Architektur existieren.

8.2.1 Operative Ebene: Agenten-basiertes Monitoring mittels Client Agents

Grundsätzlich unterscheiden wir auf der operativen Ebene zwischen Performance-Daten-Produzenten und -Konsumenten. Hierbei bilden die verschiedenen Ressourcen bzw. die auf sie aufgesetzten Tools die produzierenden Daten-Quellen für die **Monitoring-Agenten** (Client Agents), den (konsumierenden) Empfängern der Performance-Daten.

Unter einem (Monitoring-)**Agenten** verstehen wir ein Stück Software, das nach einem parametrisierbaren Algorithmus aktiv Daten holt (pull) bzw. auf Daten-Empfang wartet (push) und die gesammelten bzw. temporär zwischengespeicherten Daten in bestimmten Abständen ggf. aufbereitet an eine zentrale Komponente übermittelt. Verschiedene Agenten können gleichzeitig Daten unterschiedlichster Komponenten nach ihren individuellen Richtlinien sammeln und damit die Aufgaben sowie die Last auf dem System angebrachter verteilen. Daten können durch die Agenten direkt an der Quelle vorverarbeitet und auch analysiert werden. Dies kann helfen, den Datenverkehr im Sinne der zu übertragenden Datenmenge über das Netzwerk deutlich zu reduzieren. Sofern man davon ausgeht, dass sich die Agenten untereinander austauschen, steigt dadurch jedoch, wenn auch in geringem Maße, der Kommunikations- und Koordinations-Aufwand.

Zur Daten-Übertragung interagieren die Daten-produzierenden Quellen und die -konsumierenden Agenten mit Hilfe von

- **Notifikationen**: Die Performance-Daten werden, initiiert durch den Produzenten, alle in einem Schritt an den/die entsprechenden Konsumenten geschickt.
- **Query/Response** (Polling): Das Polling entspricht der klassischen Anfrage-Antwort-Strategie. Initiiert durch den Konsumenten werden (typischerweise regelmäßig) aktuelle Daten abgefragt und durch den Produzenten an den Anfrage-Steller als Antwort übermittelt. Polling ist ein weit verbreiteter Ansatz. Er ist einfach umzusetzen und nahezu alle Datenquellen unterstützen das zugrundeliegende Pull-Paradigma durch ein entsprechendes (Sensor-)Interface. Das Hauptproblem besteht jedoch in der Wahl eines geeigneten Polling-Intervalls mit entscheidendem Einfluss auf die Aktualität bzw. Existenz von Metrik-Werten. Die Abfragen sollten im Detailgrad und Umfang, dynamisch, in Abhängigkeit von der Last und dem Zustand des Systems, oder auch basierend auf Richtlinien angepasst werden (Adaptive Monitoring, vgl. Abschnitt 5.4.2). Ein zu einer gegebenen Zeit adäquates Abfrage-Intervall kann zu einem späteren Zeitpunkt, je nach Systemzustand etwa, zu lang oder zu kurz sein, so dass wichtige Informationen entweder verloren gehen oder der gesammelte Detailgrad zu groß ist.
- **Publish/Subscribe**: Mit diesem Verfahren wird es den „Publishern" ermöglicht, ausgewählte Performance-Daten oder, wie in den meisten Fällen, Events an eine große Anzahl beliebig weit verstreuter „Subscriber" (Abonnenten) zu verteilen, die mittels einer Anmeldung (einem Abonnement) ihr Interesse an spezifischen Informationen angezeigt haben. Der Ansatz wird verwendet, wenn eine bestimmte Instanz von einem speziellen Vorfall informiert werden soll. Die Veranlassung dazu erfolgt durch den Produzenten. Dieser stellt fest, dass ein Ereignis (bzw. eine Zustands-Änderung) aufgetreten ist und benachrichtigt daraufhin die zuvor bei ihm registrierten Abonnenten automatisch über eine Aktualisierungs-Schnittstelle. Soll-

ten die Produzenten jedoch über keine Möglichkeit zum Push-basierten Übermitteln der Daten verfügen, so lässt sich nicht auf ein strategisches Polling verzichten. Es ist in jenem Falle sogar nicht ausgeschlossen, dass ein entsprechend intelligenter Agent anhand der überwachten bzw. gesammelten Daten auch selbst Events detektieren und an in der Infrastruktur nachgelagerten Komponenten weiterleiten kann.

Die Hauptaufgabe der Agenten ist die Überwachung, temporäre Speicherung und regelmäßige Übertragung von *Performance-Metriken* und der durch die Tools der operativen Ebene entdeckten (lokalen) *Events*. Die gewonnenen Daten werden dabei auch mit zusätzlichen Informationen, z.B. über die (Topologie) der Quelle und den Zeitpunkt der Sammlung, angereichert und an das zentrale Performance Warehouse übermittelt.

Die Informationen können aus verschiedensten Quellen gewonnen werden: von elementaren Hardware- oder Software-Sensoren, die Performance-Daten in Echtzeit zur Verfügung stellen, über das Query-Interface von Datenbanken zur Aufbietung historischer Daten, bis hin zu komplexen Monitoring-Tools und Event-Engines (vgl. Abschnitt 4.1.2 sowie Abschnitt 5.3). Die Monitoring-Agenten nehmen dabei zwei Rollen ein. Bei der Kommunikation mit der operativen Ebene sind sie als Daten-Konsumenten, bei der Interaktion mit der nachfolgenden Warehouse-Ebene sind sie jedoch als Daten-Produzenten zu verstehen. Ein Monitoring-Agent kann dabei eine oder mehrere Quellen überwachen und die Daten an einen oder beliebig viele Konsumenten zur Auswertung senden. In unserem Szenario soll jeder Performance-Daten-Quelle genau ein Monitor-Agent zugeordnet sein. Die Daten aller Monitore werden in die Staging Area der Warehouse-Ebene extrahiert, dort zwischengespeichert und nach spezifischen Regeln transformiert. Diesen auch in Abbildung 8.2 dargestellten Vorgang bezeichnen wir als ETL1.

Monitoring-Agenten, die *intelligent* agieren, können neben den aufgeführten Basisfunktionalitäten durchaus auch für das Filtern von Events sowie das Caching oder die Zwischenverarbeitung (Bereinigung, Normalisierung, Aggregation) von Performance-Daten verantwortlich sein. Ein solcher Agent könnte bspw. auch derart konzipiert sein, dass er Performance-Daten nur dann weiterleitet, wenn basierend auf der Wertehistorie bestimmter Maßzahlen und einem Vorhersage-Modell mit einer Bereichsverletzung und damit mit einem künftigen Problem zu rechnen ist. Es ist auch denkbar, dass die Agenten selbst über Mechanismen zur Detektion von Events verfügen und diese im Falle des Auftretens an einen globalen Event Handler zur (globalen) Korrelation (lokaler Events) weiterleiten. Die Aufgaben intelligenter Monitoring-Agenten gehen demnach weit über eine einfache Weiterleitung bzw. temporäre Speicherung der Performance-Daten hinaus.

Vorstellbar ist auch, dass die Agenten in der Lage sind, die Übertragung der Daten in Abhängigkeit von der aktuellen Last zu verzögern bzw. anzupassen. Adaptiv können sie auch nur zu bestimmten Zeiten oder auf Anforderung für eine definierte Zeitdauer aktiv werden. Ziel ist neben der Verminderung der Belastung des operativen Systems auch die Reduktion der Last auf dem zentralen Repository sowie der über das Netzwerk zu übertragenen Daten-Mengen.

8.2.2 Warehouse-Ebene: Daten-Gewinnung und -Integration mittels Server Agent

Die Unterteilung der Sammlung der Daten durch die Agenten und die eigentliche Integration in das PWH erlaubt größtmögliche Flexibilität und zahlreiche Datenerfassungs-Szenarien. Um die Vielzahl von heterogenen Quellen auf der operativen Ebene überwachen zu können, benötigt man ein möglichst generisches Tool, welches den Zugriff auf diese Quellen ermöglicht, die erforderlichen Struktur- und Performance-Daten extrahiert, bereinigt,

auf ein einheitliches Format transformiert und anschließend in das Performance Warehouse überträgt.

Der **Server Agent** bildet den zentralen Anlaufpunkt zur Konfiguration, Verwaltung, Überwachung sowie Steuerung der Sammlung und auch der Extraktion (E), Transformation (T) und Zuführung (L, Laden) der durch die Monitoring-Agenten gesammelten Performance-Daten in das zentrale PWH. Diesen durch den Server Agent koordinierten und verwalteten Prozess von der Beschaffung der Performance-Daten bis hin zur anschließenden Integration in das PWH bezeichnen wir als **ETL1-Job**.

Voraussetzung für eine nachfolgende korrekte, korrelierte Analyse ist ein gemeinsames Zeitverständnis über den ermittelten Zustand und das Verhalten der Ressourcen. Der Server Agent übernimmt daher auch die Rolle als zentraler *Zeitgeber* und dient somit der Synchronisation der potentiell verschiedenen Zeiten heterogener Daten-Quellen.

8.2.2.1 Extraktion

Der Client Agent ist für die *Extraktion* im ETL1 zuständig und stellt die Verbindungskomponente zwischen der operativen und der Warehouse-Ebene dar. Da auf der operativen Ebene sehr viele verschiedene Tools und Quellen vorhanden sein können, muss für jede Art von Quelle, über die per Sensor-Schnittstelle Performance-Daten bezogen werden sollen, ein entsprechend konfigurierter Client Agent zur Verfügung stehen. Die Schwierigkeit der Daten-Extraktion besteht daher mitunter in der potentiellen Vielfalt an Sensor-Schnittstellen mit unterschiedlichsten Daten-Formaten und der damit verbundenen notwendigen Flexibilität der Agenten. Über die Extraktion der Daten in die Staging Area findet somit eine erste Homogenisierung und Vereinheitlichung der heterogenen Quellen und Formate statt.

Je nach Datenvolumina, Dynamik der Daten und Anforderungen (insbesondere an die Aktualität) können die Zeitpunkte der Extraktion (und auch des Ladens) unterschiedlich gewählt werden. Analog zur Sammlung kann dies demnach sofort, periodisch, Anfrage- oder auch Ereignis-gesteuert geschehen. Die genaue technische Realisierung hängt dabei stark von der vorherrschenden Hard- bzw. Software sowie den Schnittstellen der Datenquellen ab.

Grundsätzlich soll der Client Agent zumindest zwei Betriebsarten unterscheiden. Hierzu zählen zum einen das periodische Extrahieren von Performance-Daten (*Routine Monitoring*) und zum anderen das gezielte umfangreiche Sammeln bzw. Extrahieren von Daten im akuten Problemfall (*Intensive Monitoring*) über einen kurzen Zeitraum mit definierten Start und End-Zeitpunkt.

Ein Client Agent muss neben der aktiven, adjustierbaren Sammlung bzw. Extraktion von Performance-Daten die Möglichkeit haben, die Datensammlungs-Intervalle der Monitoring-Tools der Quellen über entsprechende Effektor-Schnittstellen zu *rekonfigurieren*. Beispielsweise soll der IBM Performance Expert in unkritischen Zeiten alle 30 Minuten Snapshots der CPU-Auslastung in sein Repository schreiben. Im Falle einer Beeinträchtigung der Leistungsfähigkeit des Systems, also einer kritischen, näher zu beobachtenden Lage, ist dieses Intervall jedoch unzureichend, so dass der Client Agent die Möglichkeit haben muss, den Wert auf 60 Sekunden zu verringern, um somit eine höhere Granularität und adäquatere Analyse-Basis zu erreichen.

Zum Transport der extrahierten Daten in die Staging Area bieten sich die inkrementelle Übertragung der seit der letzten Extraktion geänderten Daten sowie eine vollständige Daten-Übertragung bei jeder Anfrage an. Ersteres muss von den Daten-Produzenten

unterstützt werden. Darüber hinaus verlangt eine vollständige Daten-Übertragung die (Zwischen-)Speicherung der gesammelten bzw. generierten Daten.

8.2.2.2 Transformation

Der Transformation Agent hat im Rahmen des ETL1 die Aufgabe, die heterogenen, mittels Client Agent aus der operativen Ebene extrahierten und in der Staging Area abgelegten Performance-Daten zu bereinigen (cleanen) und in ein einheitliches Format zu transformieren, damit diese schließlich konsistent in dem Performance Warehouse abgelegt werden können.

Der Transformation Agent muss, im Rahmen einer ersten Phase, über geeignete Funktionen zur Beseitigung von Datenkonflikten (unterschiedliche Datentypen, verschiedene Kodierungen etc.) verfügen, die durch bereits vorhandene Tools oder Funktionen des Datenbank-Management-Systems ergänzt werden können. In einer darauf folgenden Phase werden die für unseren Kontext notwendigen Transformationen und Abbildungen auf die Ziel-Schemata vorgenommen.

Die Staging Area kann demnach als Zwischenspeicher verstanden werden, nicht jedoch als Datenquelle, auf die direkt Auswertungen zugreifen können. Aus diesem Grund kann und sollte der Inhalt der Staging Area in regelmäßigen Abständen gelöscht werden.

Neben der Möglichkeit, die Transformation in der Staging Area durchführen und sie damit u.U. zum Flaschenhals ausarten zu lassen, können Client Agents um Transformationsfunktionen angepasst werden, um die transformierten Daten direkt an das PWH weiterzuleiten. Auch eine Menge von dedizierten Transformation Agents innerhalb der Staging Area erscheint in diesem Zusammenhang im Sinne einer möglichen Parallelisierung als sehr zweckdienlich.

8.2.2.3 Laden

Der Loading Agent ist für den Ladevorgang im ETL1 zuständig. Er integriert die bereinigten und transformierten Performance-Daten in den Short-Term Storage des PWH und vollzieht somit eine physische Verschiebung von der Staging Area in die Datenbank des Performance Warehouse.

Eine wichtige Voraussetzung für eine schnelle Daten-Übertragung und die Reduktion des Volumens ist die inkrementelle Extraktion, die durch die Client Agents erfolgt. Dadurch können ausschließlich Änderungen in den Datensätzen bzw. relevanter Ressourcen betrachtet und in das PWH übertragen werden. Um die Einfügelast auf das PWH zu verringern und ein wirtschaftlicheres Bulk-Loading zu ermöglichen, können die transformierten Performance-Daten in der Staging Area auch gesammelt und zu vorher definierten Zeitpunkten bzw. zu lastarmen Zeiten oder nach einer gewissen Anzahl an Extraktionsoperationen übertragen werden.

Die Übertragung der Daten von der Staging Area in das PWH kann demnach synchron bzw. asynchron erfolgen. Neben der sofortigen, synchronen Übertragung von Performance-Daten, die hochaktuell und schnellstmöglich im PWH benötigt werden, erscheint demzufolge auch der asynchrone Transfer von Daten, die nicht zeitnah im PWH vorliegen müssen, als relevant. Synchrones, zu häufiges Übertragen birgt dabei stets ein gewisses Risiko, den aktuellen PWH-Analyse-Betrieb durch die hohe Einfüge-Rate merklich negativ zu beeinflussen. Das asynchrone Übertragen lässt sich in der Regel gut kontrollieren. Allerdings sollte darauf geachtet werden, dass für die Übertragung der Performance-Daten nicht zu lange auf einen lastarmen Zeitraum gewartet wird. Erstrebenswert für den Ladevorgang

ist ein Kompromiss zwischen einer relativ zeitnahen Übertragung und einer möglichst geringen Beeinflussung des Warehouse-Betriebs.

Wenn die Performance-Daten in das PWH übertragen wurden, kann es sinnvoll sein, darauf aufbauende *Performance Cubes* zu aktualisieren. Dies könnte als Information in dem jeweiligen ETL1-Job festgelegt werden. Ebenfalls denkbar wäre eine thematische Priorisierung der Jobs, um besonders zeitkritische Daten bevorzugt zu behandeln.

8.2.3 Warehouse-Ebene: Das Performance Warehouse

Das Performance Warehouse besteht, wie in Abbildung 8.2 ersichtlich, aus folgenden vier Speicherbereichen. Die drei Speicherbereiche Short Term Storage, Long Term Storage und Cube Storage des Performance Warehouse sind ähnlich aufgebaut und unterscheiden sich dabei in Granularität und Umfang der Performance-Daten.

- **Temporary Storage (TS)**
  In diesem Speicherbereich werden alle temporär zu speichernden Daten abgelegt. Hierzu zählen u.a. Datensammlungen aus dem Intensive Monitoring oder auch nicht kontinuierlich gesammelte Daten, die der Entscheidungsfindung durch den ATE dienen und für Symptom- und Event-Erkennung in Echtzeit geeignet sind. Im Grunde sind die Daten des Temporary Storage unabhängig von denen der restlichen drei Speicherbereiche, da sie jene weder beeinflussen noch über Beziehungen miteinander verknüpft sind. Der Speicherplatz-Bedarf für diesen Speicherbereich kann als relativ konstant angesehen werden, da analysierte Datensammlungen in Zukunft nicht mehr benötigt und daher zu gegebener Zeit gelöscht werden können.
- **Short Term Storage (STS)**
  In diesem Speicherbereich werden alle Performance-Daten abgelegt, die durch ETL1-Jobs im Rahmen des Routine Monitoring gesammelt wurden. Die im Short Term Storage vorliegenden Performance-Daten bieten Analysten aufgrund ihrer unaggregierten Rohdaten-Natur den höchsten Detailgrad. Da die Daten des STS in der Regel der kurzfristigen Entscheidungstreffung dienen, können sie in regelmäßigen Abständen verdichtet und in den Long Term Storage des Performance Warehouse verschoben werden (ETL2). Der Speicherplatz-Bedarf für diesen Speicherbereich kann, wie beim Temporary Storage, als relativ konstant angesehen werden. Obgleich kontinuierlich neue Performance-Daten in den Short Term Storage gelangen, so werden im Gegenzug dazu „alte“ Performance-Daten regelmäßig aggregiert in den Long Term Storage verschoben.
- **Long Term Storage** (LTS)
  Das LTS dient der langfristigen Speicherung aggregierter Performance-Informationen mit dem primären Ziel der Trend-Analyse und des gezielten Einsatzes von Data-Mining-Analyse-Techniken. Die Antwortzeiten spielen auf dieser Ebene aufgrund des eher großen Betrachtungszeitraums der Anfragen eine weitaus geringere Rolle als im TS oder STS, sollten aber nicht völlig unbeachtet bleiben. Im Sinne eines ETL2-Prozesses kann das STS hierbei als Quelle und das LTS als Ziel der Aggregation angesehen werden. Der Aufbau des LTS soll sich daher an dem des STS orientieren. Typischerweise bleiben Daten im LTS über einen sehr langen Zeitraum erhalten. Im Gegensatz zum TS oder

auch STS ist der Speicherbedarf dieses Speicherbereichs daher nicht konstant und sollte gelegentlich erweitert werden.

- **Cube Storage** (CS)
  Der Cube Storage versteht sich als eine Ansammlung (materialisierter) Problem-spezifischer Ausschnitte häufig zugegriffener Performance-Daten aus dem STS bzw. dem LTS. Die Idee hinter den Performance Cubes wurde bereits in Abschnitt 6.3 veranschaulicht. Ziel ist es, auf Basis der Cubes, häufig wiederkehrende bzw. ähnliche Problem-Situationen rechtzeitig zu erkennen und mit Hilfe dafür zugeschnittener Performance-Daten eine Ursachen-Analyse durchzuführen.

Die Performance-Daten werden in konfigurierbaren Intervallen durch den Loading Agent in die Staging Area der Warehouse-Ebene (siehe Abbildung 8.2) übertragen, um eine etwaige Bereinigung und Transformation zu durchlaufen. Im Anschluss werden die Daten durch den Loading Agent je nach Kontext entweder in den Temporary Storage oder den Short Term Storage übermittelt.

Um die Daten-Mengen überschaubar zu halten und Top-Down-basierte Analysen zu ermöglichen, ist unser **Summarization Agent** für das regelmäßige Verdichten (Aggregieren) der Performance-Daten vom Short Term Storage in den Long Term Storage sowie das Löschen (Pruning) nicht mehr benötigter Daten zuständig. Dieser Vorgang umfasst, ähnlich wie ETL1, Extraktion, Transformation und Laden und wird daher im Folgenden als **ETL2** bezeichnet.

Im Gegensatz zu ETL1, ist ETL2 deutlich weniger komplex. Für die Extraktion genügt i.d.R. eine SQL-Anfrage an den Short Term Storage des Performance Warehouse. Bei der Transformation kann auf eine Datenbereinigung verzichtet werden, da dies bereits durch den Transformation Agent beim ETL1 erfolgt ist. Da in der Regel Quell- und Zielschema nahezu identisch sind, genügt es dem Summarization Agent sich auf Aggregations-Operationen während der Transformation zu beschränken. Das Laden bei ETL 2 erfolgt ähnlich wie die Extraktion per SQL. Da bei ETL2 permanent Performance-Daten vom STS in den LTS verschoben werden, muss der Summarization Agent zusätzlich die Funktion eines Schedulers übernehmen. Die Aufbewahrungszeiten, Aggregations- sowie Pruning-Intervalle sind überlegt und unter Berücksichtigung des notwendigen Detailgrades anschließender Analysen zu wählen.

Der **Cube Creator** hat die Aufgabe, Daten aus dem Short Term Storage bzw. aus dem Long Term Storage des Performance Warehouse zu extrahieren, in multidimensionale Performance Cubes (siehe Abschnitt 6.3.2) zu transformieren und anschließend im Cube Storage abzulegen. Ein Performance Cube kann mit einer (materialisierten) Sicht verglichen werden und ist demzufolge lediglich ein (materialisierter) problembezogener Ausschnitt aus dem STS bzw. den LTS des Performance Warehouse. Diesen ebenfalls in Abbildung 8.2 visualisierten Prozess bezeichnen wir als **ETL3**. In Abschnitt 6.3 haben wir zwei verschiedene Vorgehensweisen der Daten-Speicherung im PWH angedeutet. Sollten STS und LTS multidimensional strukturiert sein, kann der Cube Storage eine ähnliche (abgeleitete) Struktur wie der LTS bzw. STS mit den ausschließlich die für die Analyse durch Performance-Cubes notwendigen Ausschnitte (Projektion und Selektion)besitzen. Sind hingegen STS und/oder LTS an das in Abschnitt 6.3.1 dargelegte generische Datenmodell orientiert, so gestaltet sich der ETL3-Prozess ein wenig umfangreicher, kann aber, wie wir gesehen haben, zu großen Teilen automatisiert werden.

Grundsätzlich können wir zwischen zwei Arten von Performance Cubes unterscheiden. *Ad-hoc Performance Cubes* werden bei Bedarf befüllt und liegen nicht im Cube Storage des

PWH vor. Sie sollen für weniger häufige Analysen Anwendung finden und müssen somit nicht permanent zur Verfügung gestellt werden. *Persistente Performance Cubes* hingegen werden sehr häufig benötigt und müssen ihre Inhalte schnell zur Verfügung stellen. Sie werden daher im CS persistent abgespeichert (bspw. in Form von materialisierten Sichten auf den Strukturen des PWH). Da die zugrundeliegenden Performance-Daten und damit auch der Kontext eines Performance Cubes dynamisch sind, kann ein persistenter Cube in Analogie zu materialisierten Sichten nochmals unterschieden werden. Manuelle Performance Cubes auf der einen Seite müssen bzw. sollen nicht automatisch aufgefrischt werden. Eine Aktualisierung kann bei Bedarf durch den DBA erfolgen. Automatische Performance Cubes auf der anderen Seite sollten stets aktuell gehalten werden, um zum Analysezeitpunkt möglichst unverzüglich aktuelle Daten bereitzuhalten. Das bedeutet, dass der Performance Cube in periodischen Abständen oder auch Ereignis- bzw. Daten-gesteuert mit aktuellen Performance-Daten gefüllt werden muss.

Da sich die Anzahl der Aufrufe und somit die Wichtigkeit eines Performance Cubes mit der Zeit ändern können, soll der DBA die Möglichkeit besitzen, einen optionalen Schwellwert für den Zeitpunkt der Umwandlung eines manuellen in einen automatischen Performance Cube (oder umgekehrt) anzugeben. Eine Überschreitung des Schwellwerts bedeutet dabei für einen manuellen Performance Cube, dass dieser in einen automatischen umgewandelt wird. Eine Unterschreitung des Schwellwertes hätte somit eine Umwandlung eines automatischen Performance Cube in einen manuellen Performance Cube zur Folge. Der Cube Creator hat bei solchen automatischen Umwandlungen auch die Aufgabe, Oszillations-Effekte präventiv zu erkennen bzw. zu vermeiden.

Der Speicherplatzbedarf des Cube Storage kann über die Zeit hinweg relativ konstant bleiben, sofern ungenutzte materialisierte Performance Cubes aus dem Cube Storage, im Gegensatz zu den (dauerhafteren) Performance-Daten im Short Term Storage bzw. Long Term Storage, nicht aufgehoben und regelmäßig gelöscht werden. Die Schwierigkeit für den DBA ist es, einen guten Kompromiss zwischen Geschwindigkeitszuwachs durch materialisierte Sichten und dem sich dafür „erkauften" benötigten Speicherplatz und ggf. Aktualisierungsaufwand zu finden.

8.2.4 Interaktion

Die im Folgenden beschriebene **Abbildung 8.3** stellt schematisch die Kommunikation zwischen den einzelnen Komponenten der operativen Ebene sowie zwischen dem Server Agent und den am ETL1 beteiligten Client Agents, Transformation Agent und Loading Agent dar.

Die einzelnen Interaktions- und Kommunikations-Schritte zwischen den Client Agents und den Ressourcen bzw. Tools der operativen Ebene sind in Abbildung 8.3 von eins bis acht, die Interaktion zwischen dem Transformation Agent und dem Seher Agent von neun bis elf und die Kommunikation des Loading Agent mit den restlichen Ziffern nummeriert.

Der Server Agent koordiniert und beaufsichtigt sowohl Prozesse und Verhalten der Client Agents (1), als auch des Transformation Agent (9) und des Loading Agent (13).

Der Server Agent kann u.a. das Extraktions-Intervall der ihm zugeordneten Client Agents konfigurieren (1). Darüber hinaus kann ein durch den Server Agent gesteuerter Client Agent die Einstellungen zum Umfang, Art oder auch Zeitpunkte der Daten-Sammlung des ihn mit Performance-Daten beliefernden Tools anpassen und damit über die ihm zur Verfügung stehenden Effektor-Schnittstellen Einfluss auf dessen Verhalten und Kommunikation mit der Daten-Quelle nehmen (2, 4). Der Client Agent kann ebenso direkt auf

seine zugehörige Quelle zugreifen, sie wenn nötig und möglich rekonfigurieren und anpassen (3). Auf eine Daten-Quelle können mehrere Agenten (7), aber auch mehrere Tools zugreifen (5). Letztere extrahieren, ähnlich wie Client Agents Daten aus dieser Quelle (5) und leiten sie von selbst oder auf Anfrage an die Agenten weiter (6). Das vorläufige Zwischenziel der Extraktion durch die Client Agents ist die Staging Area (8), in der die gewonnen Performance-Daten einer anschließenden Transformation unterliegen (10, 11). Der Loading Agent ist schließlich für die Integration der Performance-Daten von der Staging Area in das Performance Warehouse verantwortlich. Performance-Daten, die im Rahmen des Intensive Monitoring (siehe Abschnitt 8.2.2.1) gewonnen wurden, sollen lediglich in dem Temporary Storage des PWH verweilen (13). Sie müssen schnellstmöglich, jedoch nur kurzfristig zur Verfügung stehen, sind nicht zu aggregieren und gehen daher auch nicht in den Long-Term Storage ein. Im Rahmen des Routine Monitoring extrahierte Daten, werden zunächst in den Short-Term Storage des PWH übertragen (14). Sie können später aggregiert und in den Long-Term Storage transferiert werden (15).

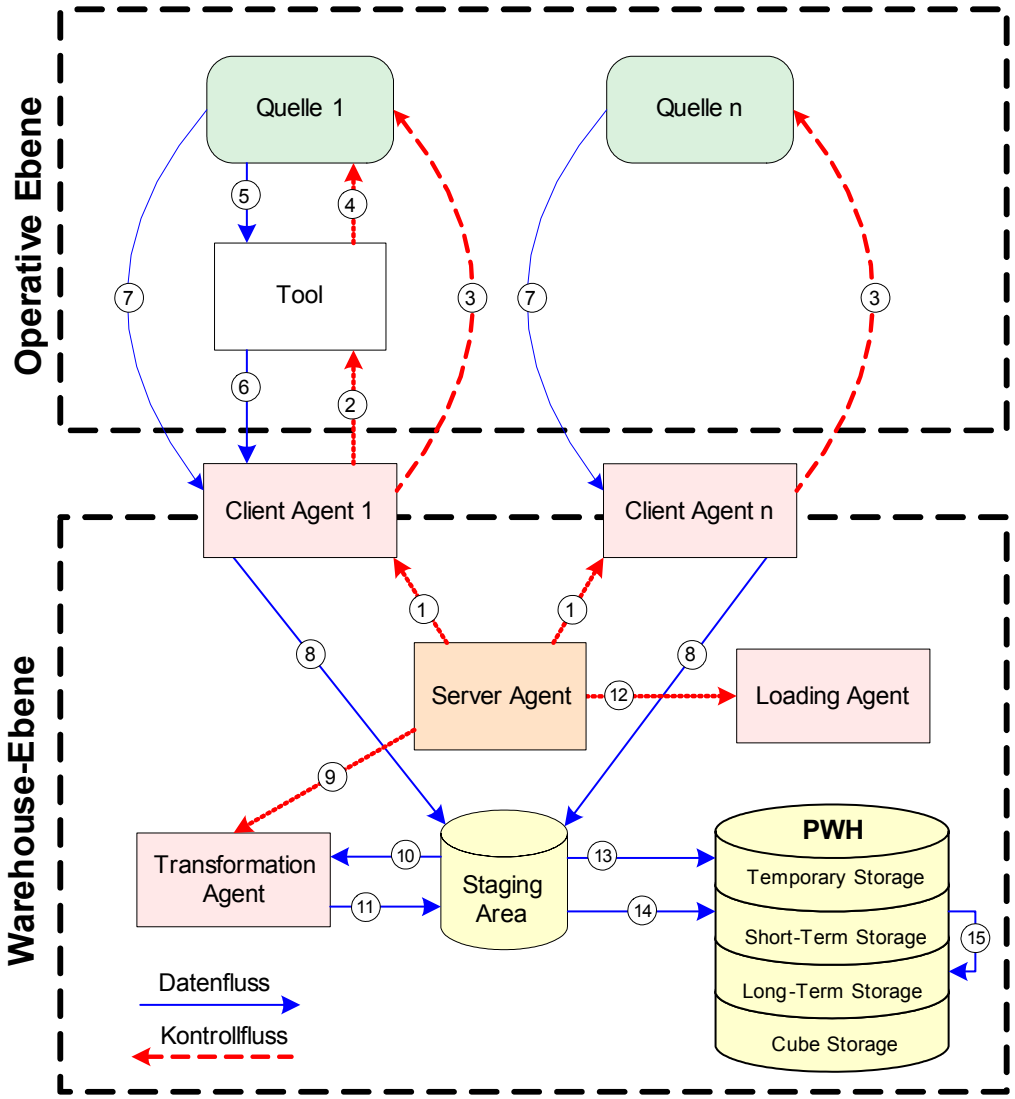


Abbildung 8.3: Zusammenspiel der operativen mit der Warehouse-Ebene (nach [Köh10])

8.2.5 Administrative Ebene

Auf der administrativen Ebene hat der DBA die Möglichkeit, auf die Performance-Daten und die Konfiguration des Warehouse zuzugreifen. Die **DBA GUI** stellt dafür ein *Server Agent Interface* und ein *Performance Warehouse Interface* zur Verfügung.

Über das *PWH Interface* kann der DBA „manuell" oder auch mit Hilfe von Tools auf alle Daten des Performance Warehouse zugreifen, um diese für Analyse- und Problemlöse-Zwecke weiterzuverwenden. Über ein entsprechend zu entwickelndes Reporting-Modul beispielsweise könnten die Daten auf diesem Wege in Form von nutzerfreundlichen Grafiken oder Berichten angezeigt und ausgewertet werden. Über das PWH-Interface können sowohl DBAs als auch ein autonomes System nicht nur auf die Performance-Daten des PWH, sondern auch auf dessen Komponenten (Summarization Agent, Cube Creator) Zugriff erhalten.

Über das *Server Agent Interface* kann der DBA (und ein autonomes System) alle Komponenten in Hoheit des Server Agent (Client Agents, Transformation Agent, Loading Agent) konfigurieren, verwalten, steuern und koordinieren. Dies erfolgt maßgeblich durch das Anlegen, Bearbeiten, Verändern oder Löschen von ETL1-Jobs.

8.3 Performance Tuning mit dem Autonomic Tuning Expert

In Abschnitt 4.2 wurde bereits skizziert, auf welche Art und Weise Tuning-Wissen formalisiert und maschinenlesbar gemacht werden kann, um das Ziel des Nachbaus und der Automatisierung der menschlichen Vorgehensweisen zu erreichen. Maschinell sollen Performance-Probleme effektiver identifiziert, analysiert und gezielter behoben werden können.

Unser im Folgenden vorgestellter Workload- und Workflow-basierter **Autonomic Tuning Expert (ATE)** setzt sich diese Aufgabe zum Ziel [RWRA08, WRRA08, WRRA09]. Der architekturelle Aufbau orientiert sich dabei an den durch das erweiterte Wissensmodell (siehe Abschnitt 4.3) identifizierten Bereichen in Form separater Komponenten bzw. Repositories. Zur Validierung der Referenz-Architektur wurden einige auf DB2 zugeschnittene Tuning-Abläufe mit Datenbank-spezifischen Workflow-Aktivitäten modelliert, in die Architektur eingebettet und unter kontinuierlicher Workload autonom ausgeführt. Die ersten erfolgreichen Evaluations-Ergebnisse bestätigen, dass die Implementierung und Integration des ATE erfolgreich vollzogen werden konnte und die zusätzlichen Überwachungs- und Tuning-Komponenten keinen merklichen Einfluss auf die Funktionsweise des operativen Systems haben.

8.3.1 Architekturbeschreibung

Wie wir bereits in Abschnitt 2.2.4 dargelegt haben, verwalten IT-Systeme mit nach [IBM06e] umgesetzten autonomen Fähigkeiten ihre Ressourcen mit Hilfe so genannter Autonomic Manager. Diese implementieren dazu typischerweise eine MAPE-Schleife. Die Betrachtung der Problembereiche erfolgt lokal. Zur bereichsübergreifenden Problem-Erkennung und -Auflösung ist die Einführung von übergeordneten, in einer Hierarchie angeordneten Autonomic Managers möglich. Vorteile dieser verteilten Lösung sind in der effizienten Aufgabenverteilung und der lokalen, autonomen Abarbeitung von Problemen zu sehen.

Als Kritik am IBM-Autonomic-Computing-Prinzip wird häufig eben jener Aufbau von Systemen als Kombination vieler eigenständiger Regelkreise gesehen, vor allem, da die

Interferenzen zwischen den einzelnen Schleifen noch weitgehend unerforscht sind und gesonderter Betrachtung im Hinblick auf die Abstimmung der einzelnen lokalen Entscheidungen bedürfen. Ein überhöhter Koodinations- und Kommunikationsaufwand zwischen den vielen Managers kann ein weiterer Nachteil sein.

Unser Ansatz weicht von dieser Herangehensweise jedoch ab, da wir nach der goldenen Regel des Tuning Änderungen am System ausschließlich sequentiell („one change at a time“) vornehmen möchten, um somit die Auswirkungen bzw. den Einfluss einzelner Aktionen nachvollziehen und ggf. rückgängig machen zu können. Ein einzelner Tuning-Schritt entspricht dabei einem Schleifendurchlauf. Dadurch reduziert sich der Aufwand zur Erstellungs- und Ausführungszeit, da der Nutzer die benötigte Planungslogik (innerhalb der Tuning-Pläne) nicht selbst umsetzen muss und systemseitig keine Orchestrierung der einzelnen MAPE-Schleifen notwendig ist.

Aber fairerweise sollte nicht unerwähnt bleiben, dass es auch hier nicht nur Vorteile gibt. Eine einzige globale Instanz zur Überwachung aller Subkomponenten, zur Anhäufung des Wissens und zum Koordinieren/Treffen der Entscheidungen kann im eher unwahrscheinlichen worst case zum Flaschenhals des Autonomic Tuner oder gar zum single point of failure ausarten.

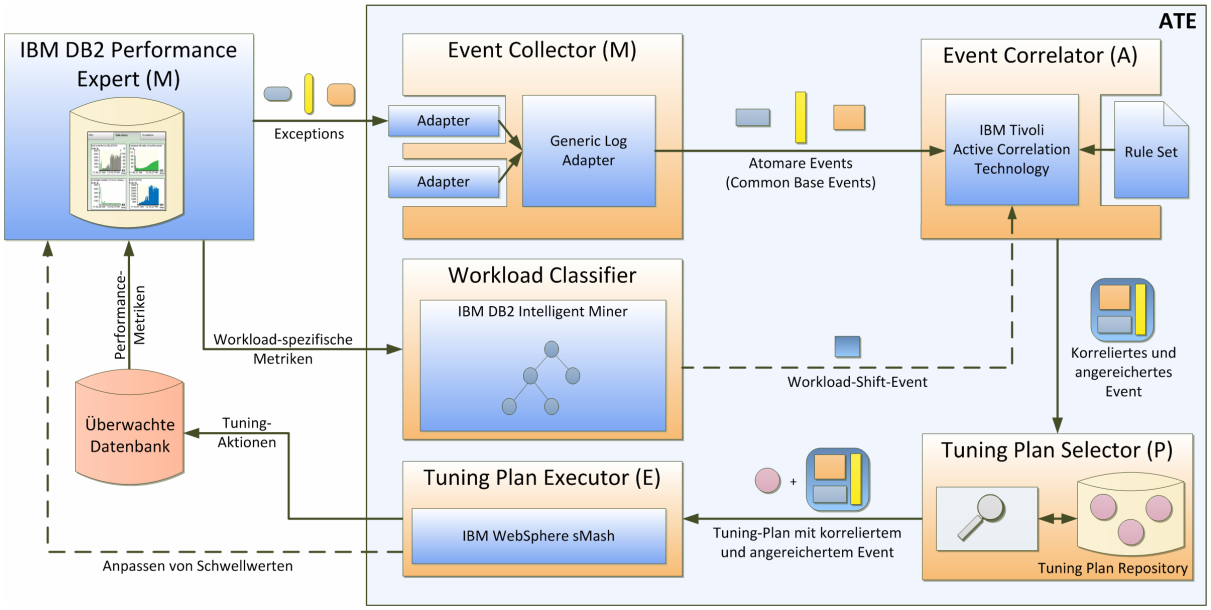


Abbildung 8.4: Architektur des Autonomic Tuning Expert

Das ATE-Framework, unser Architekturvorschlag zum autonomen Datenbank-Tuning, implementiert eine einzige MAPE-Schleife zur Automatisierung des Datenbank-Tunings [RWRA08, WiRa07, WRRA08, WRRA09]. Die Architektur integriert Standardprodukte, wie IBM DB2 Performance Expert (PE, [CBM+06]), IBM WebSphere sMash (WsM, [WsM09]) und IBM DB2 Intelligent Miner [IBM06h] und basiert auf weit verbreiteten Technologien, wie Generic Log Adapter (GLA, [IBM05]), Tivoli Active Correlation Technology (ACT, [BiGa05]) und Common Base Events (CBE, [OKSC04]).

Die MAPE-Phasen werden durch einzelne Komponenten realisiert, die miteinander kooperieren und untereinander von außen bereitgestellte oder generierte, akkumulierte Informationen (Wissen) austauschen um die Funktionalität des MAPE-Regelkreises zu gewährleisten. Es soll auch nicht unerwähnt bleiben, dass sämtliche Komponenten ihre Aktivitäten protokollieren und damit eine Überwachung bzw. Evaluation ihrer selbst bzw. ihrer Wirkungsweise und Effektivität ermöglichen. **Abbildung 8.4** veranschaulicht dabei das Zu-

sammenspiel aller Komponenten des Frameworks, die im Folgenden näher beschrieben werden.

Der **Event Collector**, die Überwachungskomponente des ATE-Frameworks, basiert auf dem Generic Log Adapter (GLA) als Teil der Eclipse Test & Performance Tools Platform (TPTP) und verwendet als integralen Bestandteil den Performance Expert. PE bietet neben der Möglichkeit, aktuelle Performance-Metriken anzuzeigen bzw. Trendanalysen auf Basis von in der Vergangenheit gesammelten Performance-Daten durchzuführen (Performance Warehouse), auch einen Mechanismus zur Notifikation über Schwellwert-basierte Ausnahmesituationen (siehe Abschnitt 5.3.2.1). Daher wird PE nicht nur zur regelmäßigen Erzeugung und Speicherung von Performance-Snapshots herangezogen, sondern ebenfalls zur Definition und Überwachung von Exceptions (atomaren Ereignisse) genutzt. Hierbei wird die Überschreitung von vorher definierten Schwellwerten, sprich Anzeichen für potentielle Performance-Probleme, die einer weiteren Diagnose bzw. Behandlung bedürfen, dem Administrator in Form von Ereignismeldungen visuell präsentiert. Es besteht zudem die Möglichkeit, die Informationen über aufgetretene Exceptions mittels eines User Exit für beliebige Konsumenten maschinenlesbar in XML-Form zu externalisieren.

Der GLA sammelt und verarbeitet kontinuierlich Monitor-Daten, die durch Nutzung verschiedener Adapter aus beliebigen Quellen (u.a. Tabellen, Dateien) stammen können. Die von der Ausnahmebehandlungs-Komponente des PE übermittelten Informationen über aufgetretene Exceptions werden mit Hilfe eines Adapters in das standardisierte, XML-basierte Common-Base-Event-Format transformiert und stehen somit dem Event Correlator zur Verfügung. Dabei werden die CBEs mit Metadaten, wie dem aktuell anliegenden Datenbank-Workload-Typ sowie den beteiligten Ressourcen angereichert und schließlich dem Event Correlator für die Analyse-Phase übergeben. Es ist angedacht weitere Adapter z.B. für die Nutzung des DB2 Diagnostic Log (db2diag.log) bzw. diverser Tivoli-Produkte zu integrieren. Damit kann ATE ohne zusätzlichen Mehraufwand auf beliebige, typischerweise in zu tunenden System-Umgebungen vorhandene Überwachungs-Mechanismen aufsetzen.

Die Analyse-Komponente (**Event Correlator**) bietet Mechanismen zur getriggerten oder auch periodischen Korrelation von CBEs und ermöglicht dadurch die Modellierung komplexer Problem-Situationen. Dabei werden Annahmen über mögliche Probleme getroffen und potentielle Reaktionen überprüft, um dadurch einen Handlungsbedarf abzuleiten. Der Event Correlator macht Gebrauch von der Tivoli Active Correlation Technology zur Korrelation und Verarbeitung von Ereignissen aus unterschiedlichen Quellen (siehe Abschnitt 4.2.2). Eingehende CBEs des GLA werden in einer Queue abgelegt und mit einem vom Benutzer definierten, Workload-abhängigen Satz von Regeln (Mustern) abgeglichen. Diese Regeln definieren welche Folgen bzw. Muster von atomaren Ereignissen ein komplexes Ereignis darstellen und helfen die in einer komplexen System-Umgebung auftretende „Flut“ von Wertebereichs-Überschreitungen zu reduzieren(siehe Abschnitt 4.2.2). Des Weiteren erlaubt dies den Kontext (z.B. andere existente Ereignisse bzw. überschrittene Schwellwerte) eines Ereignisses zu erfassen. Durch Korrelation atomarer Ereignisse lassen sich also Problem-Situationen präziser modellieren. Die entstandenen korrelierten Ereignisse werden dann an die Komponente für die Plan-Phase, den Tuning Plan Selector, weitergegeben.

Die Plan-Komponente (**Tuning Plan Selector**) ermöglicht die Strukturierung von Maßnahmen, die zum Erreichen der von außen vorgegebenen Zielrichtlinien (Policies) erforderlich sind. Aus diesem Anlass hat sie die Auswahl, Priorisierung und Verknüpfung vordefinierter, bewährter Tuning-Praktiken (Tuning-Pläne) zur optimalen Auflösung über Event-

Korrelation erkannter Performance-Probleme zur Aufgabe. Hierfür werden im Vorfeld Tuning-Pläne und deren Zuordnungen zu den durch komplexe Ereignisse repräsentierten Problem-Situationen durch DBA definiert und im Tuning Plan Repository gespeichert (siehe Abschnitt 4.2.3). Zur Laufzeit werden entsprechende Tuning-Pläne aus dem Repository selektiert und an den Tuning Plan Executor zur Ausführung weitergeleitet. Bei der Planung sollen zudem die Systemkomponenten mit allen ihren Abhängigkeiten (siehe Abschnitt 7.5) in Betracht gezogen werden, um das System stabil zu halten und Oszillationen zu vermeiden.

Die Ausführungskomponente (**Tuning Plan Executor**) führt die vom Tuning Plan Selector ausgewählten Tuning-Pläne in der entsprechenden Reihenfolge aus. Realisiert wurde der Tuning Plan Executor mit Hilfe der in den WebSphere sMash Anwendungs-Server integrierten, leichtgewichtigen Workflow-Engine. Die einzelnen Schritte eines Tuning-Plans entsprechen dabei einzelnen Workflow-Aktivitäten. Hierbei kommen sowohl Standard-Workflow-Aktivitäten als auch benutzerdefinierte, Datenbank-Tuning-spezifische Workflow-Aktivitäten zum Einsatz (siehe Abschnitt 4.2.3). Der Tuning Plan Executor protokolliert die Ausführung einzelner Tuning-Schritte und hat die Möglichkeit Ressourcen-(Re-)Allokationen zu überwachen. Dabei können Tuning-Schritte, welche vom DBA definierte Schranken über- bzw. unterschreiten, automatisch angepasst bzw. zurückgewiesen werden. Die Ausführung der Tuning-Pläne verursacht eine Zustands-Änderung der überwachten Datenbank. Diese Zustands-Änderung wird durch die Überwachungskomponente beobachtet und der Kreislauf beginnt erneut.

Der in Abbildung 8.4 dargestellte **Workload Classifier** lässt sich nicht eindeutig einer MAPE-Phase zuordnen. Seine Aufgabe ist, die aktuell anliegende System-Workload zu erkennen, um Workload-spezifische Problem-Erkennung und -auflösung zu ermöglichen (siehe Abschnitt 7.4). Hierbei wird die Workload durch Performance-Metriken, wie „die Anzahl der Sortierungen pro SQL-Statement“ oder „das Verhältnis der Select-Anweisungen zu allen SQL-Anweisungen“ charakterisiert. Sie spiegeln Merkmale und das Verhalten der aktuellen Workload in Form einer Menge von Metrik-Werten dieser Datenbank wider. Die Herausforderung ist dabei einerseits typische Metriken zu finden, die bei der Unterscheidung der zu unterstützenden Workload-Typen von Bedeutung sind. Andererseits spielt im Kontext des automatischen Performance-Tunings der Aspekt der System- und vor allem Tuning-Unabhängigkeit der Metriken eine große Rolle. Nur wenn diese Voraussetzung erfüllt ist, ist eine generische Workload-Detektion unter Tuning des Systems möglich.

Bei erkannter Änderung des Workload-Typs, wird ein Workload-Shift-Event generiert und über den Event Correlator in den MAPE-Kreislauf injiziert (vgl. Abbildung 8.4). Das Wissen über die aktuelle Workload verbreitet sich somit an alle Komponenten und stellt sicher, dass diese nun präziser auf evtl. neue Umstände reagieren können. So wird beispielsweise (1) eine von der aktuellen Workload abhängige Auswahl der Events und der Tuning-Pläne ermöglicht. Die Information der aktuellen Workload kann von beliebigen Tuning-Plänen außerdem genutzt werden, um (2) ihre Ausführung durch die Parametrisierung mit der Workload als Input-Parameter zu verändern. Zudem können (3) beliebige Tuning-Pläne in Folge des Workload-Shift-Events ausgeführt werden. So ist es beispielsweise denkbar die Konfiguration der Datenbank oder des DBMS einmalig für die erkannte Workload proaktiv zu optimieren. Im Rahmen von Meta-Tuning-Plänen können (4) Veränderungen an der Konfiguration des ATE vorgenommen werden. Derart ließen sich etwa die Schwellwerte des PE und damit die atomaren Events anpassen.

Abbildung 8.5 veranschaulicht die Integration des Workload Classifier bzw. des Konfigurationsmoduls in den Autonomic Tuning Expert. Über die GUI besteht die Möglichkeit neue

Modelle zu lernen, die dafür zu verwendeten Metriken zu spezifizieren und bereits erlernte Modelle zur Klassifikation einzusetzen.

Derzeit erlaubt die Workload-Klassifikations-Komponente des ATE die Unterscheidung zweier Workload-Klassen: OLAP und OLTP. Wie wir jedoch in Abschnitt 7.4.6 gezeigt haben, können weitere Workload-Klassen ohne größeren Aufwand eingebracht und im Tuning berücksichtigt werden.

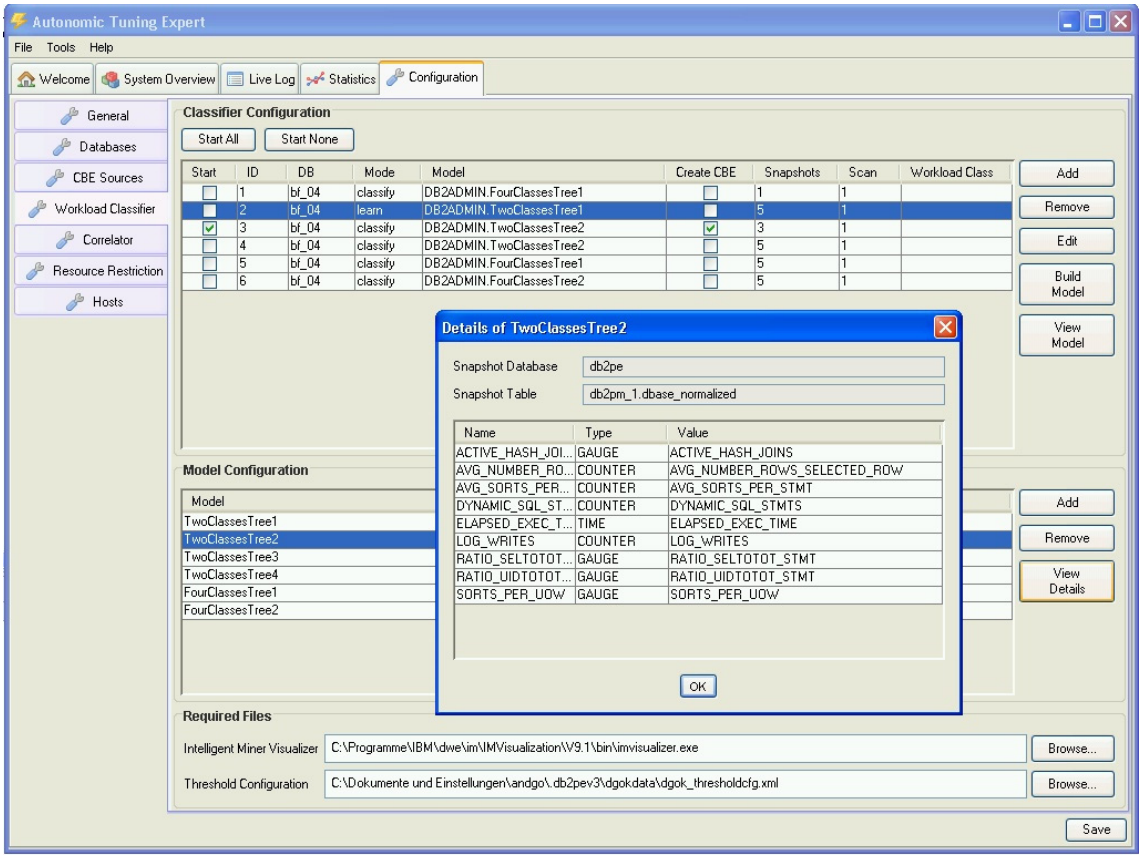


Abbildung 8.5: Integration des Workload Classifier in den ATE

8.3.2 Umgesetzte Tuning-Pläne

Zur Formalisierung und Modellierung von kritischen Situationen greifen wir auf die Exceptions von PE (atomare Ereignisse) bzw. auf die aus Abschnitt 4.2.2 bekannten Mechanismen von ACT zurück. Mittels Tivoli ACT können die atomaren Ereignisse durch Operatoren einer Ereignis-Algebra zu komplexen Ereignissen mit akutem Handlungsbedarf korreliert werden, denen zur Laufzeit durch entsprechende Tuning-Pläne begegnet werden soll.

Als Basis für die Tuning-Plan-Modellierung und Ereignis-gesteuerte Ausführung dient IBM WebSphere sMash, eine flexible Entwicklungs- und Ausführungs-Umgebung für Web 2.0-Anwendungen. Es basiert auf Project Zero, einem Mitte 2007 von IBM gestarteten Entwicklungsprojekt, welches von einer offenen Entwicklergemeinde vorangetrieben wird. Bestandteil von WsM ist ein leichtgewichtiges Workflow-System zur Erstellung von dynamischen Webinhalten mit integriertem webbasierten Workflow-Editor (Flow Assemble Tooling).

Die Basis-Funktionalitäten des Modellierungs-Editors sind das Erstellen, Speichern und Laden eines Workflows. Durch Drag and Drop kann man die Aktivitäten in den Editor hineinziehen, sie bearbeiten und mit anderen Aktivitäten verbinden. Der Editor generiert aus den zusammengestellten Aktivitäten eine „.FLOW"-Datei, die an die Workflow-Engine bzw. zum Server geschickt, dort geparst und anschließend ausgeführt wird. Dieses Workflow-System und der entsprechende Editor sind so konzipiert, dass eine Erweiterung um benutzerdefinierte Aktivitäten mit geringem Aufwand möglich ist.

Durch die Einbindung eines leichtgewichtigen Workflow-Systems in die ATE-Architektur wird es Administratoren ermöglicht, durch Wiederverwendung von vordefinierten Datenbank-Tuning-spezifischen Aktivitäten, ihre Tuning-Praktiken als spezielle Tuning-Pläne in Form von Workflows auf hohem Abstraktionsniveau zu spezifizieren.

Nachfolgend wird eine exemplarische Auswahl der im Rahmen einer Machbarkeits-Studie umgesetzten Ereignisse und Tuning-Pläne beschrieben. Die Tuning-Pläne beschränken sich im Wesentlichen auf einfache Konfigurationsparameter-Anpassungen der elementaren Heaps und Caches von DB2. Der eigentliche Fokus unseres universellen, zum DB2-internen Self-Tuning-Memory-Manager-Mechanismus [SGL+06] komplementären, Ansatzes liegt jedoch auf Black-Box-orientiertem Tuning von Datenbank-basierten Software-Stacks. Dabei erfolgt der Zugriff auf die zu tunenden Komponenten ausschließlich über die nach außen sichtbaren Schnittstellen und erfordert keinen Einblick in die System-Interna.

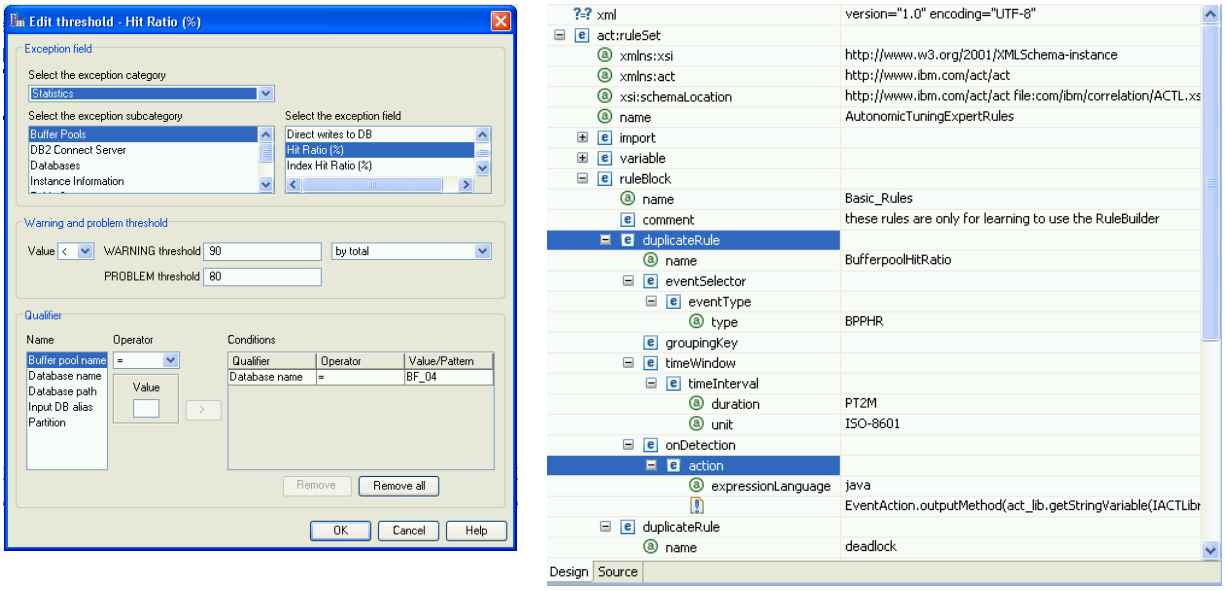
Um möglichst realistische, aber vor allem wiederholbare Workload-Szenarien für aufeinander folgende Tests mit vergleichbaren Ergebnissen zu erzeugen, haben wir uns für Variationen der Standard-Benchmarks TPC-C und TPC-H [TPCC07, TPCH08] entschieden (siehe Abschnitt 7.4.6). Damit basieren sowohl das Training, Testen und auch das anschließende Auswerten der Workload-Modelle auf der Unterscheidung dieser beiden mit OLAP bzw. OLTP verwandten Klassen.

PE-Metrik	PE-Schwellwert für OLTP	PE-Schwellwert für OLAP	Atomares Ereignis	Komplexes Ereignis	Tuning-Plan
bufferpool hit ratio	< 90	< 70	BPHR	Duplicate(BPHR)	Bufferpool-Tuning
avg number of rows read per selected row	> 5	> 25	RRRS	Duplicate(RRRS)	Index-Design-Tuning
post threshold hash joins	> 0	> 4	PTHJ	Duplicate(PTHJ *or* PTS)	Sheapthres-Tuning
post threshold sorts	> 0	> 4	PTS		
overflowed sorts	> 2	> 10	OS	Duplicate(OS *or* HJO *or* HJSO)	Sortheap-Tuning
hash join overflows	> 0	> 4	HJO		
hash join small overflows	> 0	> 4	HJSO		

Tabelle 8.1: Zuordnungen von Metriken, Ereignissen und Tuning-Plänen

Tabelle 8.1 fasst einige der konzipierten Tuning-Pläne sowie die entsprechenden Tuning-Plan-auslösenden Ereignisse zusammen. Die ersten vier Spalten beinhalten die Definitionen und die Namen der atomaren Ereignisse. Zu jedem atomaren Ereignis gehören der Name der Metrik sowie die Workload-abhängigen Schwellwerte mit zugehörigen Operatoren. Spalte 5 enthält die Korrelationsmuster der atomaren Ereignisse, die mittels entsprechender ACT-Regeln umgesetzt und als Auslöser für die korrespondierenden Tuning-Pläne (Spalte 6) verwendet werden. Zudem werden mittels einer speziellen ACT-Filterungsregel (duplicate rule) für jedes komplexe Ereignis Überreaktionen (das sind wiederholte Reaktio-

nen auf ein noch nicht gelöstes Problem) verhindert. Durch die Verwendung dieser Regel lassen sich bei Auftreten eines komplexen Ereignisses für die nachfolgende, vordefinierte Zeitperiode alle eintretenden Ereignisse gleichen Typs ignorieren. **Abbildung 8.6** stellt die Definition atomarer Ereignisse in PE (siehe Abbildung 8.6a) und die Korrelation mittels ACT (siehe Abbildung 8.6b) anhand der BPHR visuell dar. Über die Schwellwert-Definition im PE (Abbildung 8.6a) kann aus einer Menge vorgegebener, kategorisierter Metriken gewählt werden. Für das betrachtete Element werden schließlich eine Schranke zur Warnung und auch zur Indikation eines Problems definiert. Zusätzlich können Kontext-Einschränkungen vorgenommen werden. In dem in Abbildung 8.6a dargestellten Beispiel der Definition von Schwellwerten für die BPHR erfolgt eine Einschränkung auf eine konkrete Datenbank. Abbildung 8.6b zeigt die Definition eines komplexen Events in ACT. Konkret definieren wir hierbei eine Duplicate Rule für die durch den PE erkannte BPHR-Unterschreitung. Mit „PT2M“ wird ausgedrückt, dass nach Reaktion auf ein Ereignis erst zwei Minuten später auf erneut eintretende Ereignisse gleichen Typs reagiert wird.

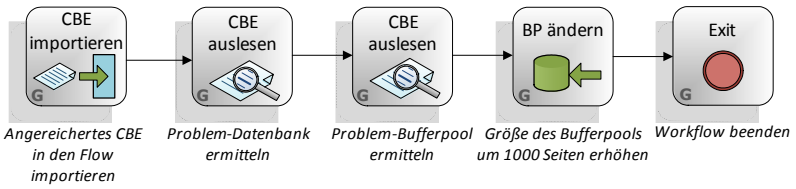


a) Schwellwert-Definition in PE

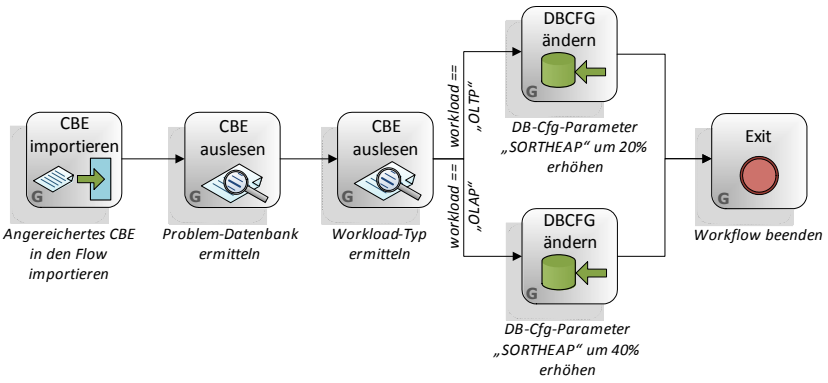
b) Definition komplexer Events in ACT

Abbildung 8.6: Problem-Formalisierung mit ATE

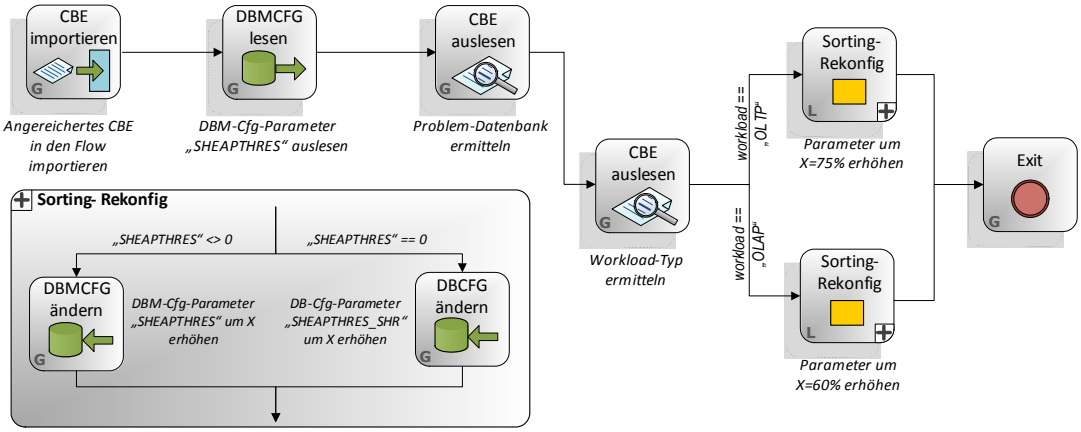
Die folgenden Unterabschnitte mit den Beschreibungen der Tuning-Pläne sollen außerdem die Möglichkeiten der visuellen Workflow-orientierten Modellierung von Tuning-Plänen zeigen. Die Darstellung der Tuning-Pläne orientiert sich dabei am Erscheinungsbild des WsM-Workflow-Editors [WsM09]. Die Workflows enthalten in dieser konzeptuellen Darstellung zur Erleichterung des Verständnisses der WsM-Konzepte zusätzliche Informationen. Dabei stellen gerundete Rechtecke (globale oder lokale) Aktivitäten dar, die sich wiederum zusammensetzen lassen und dann mit einem „+“ gekennzeichnet sind. Globale Aktivitäten sind dabei mit einem „G“ gekennzeichnet, lokale mit einem „L“. Der Typ einer Workflow-Aktivität steht innerhalb seines Rechteckes, eine Kurzbeschreibung seiner Aufgabe außerhalb. Der Kontrollfluss ist durch gerichtete Kanten zwischen den Workflow-Aktivitäten dargestellt. Dabei ist es möglich, Kanten mit Übergangsbedingungen zu versehen, die zur Laufzeit von der Ausführungskomponente ausgewertet werden. Im Falle einer positiven Auswertung erfolgen der Übergang und die Ausführung nachfolgender Aktivitäten.



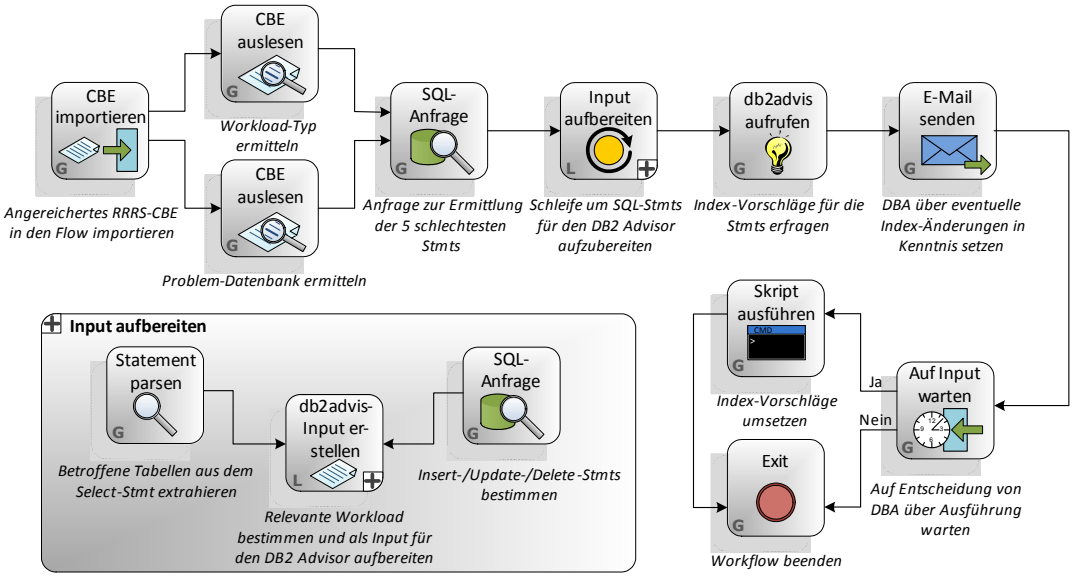
a) Bufferpool-Tuning-Plan



b) Sortheap-Tuning-Plan



c) Sheapthres-Tuning-Plan



d) Index-Design-Tuning-Plan

Abbildung 8.7: Graphische Darstellung ausgewählter Tuning-Pläne

8.3.2.1 Bufferpool-Tuning

Der Bufferpool-Tuning-Plan (**Abbildung 8.7a**) ist für die Bestimmung der optimalen Bufferpool-Größe verantwortlich, um möglichst viele nachfolgende Anfragen aus dem Bufferpool zu bedienen und damit die Zahl der Plattenzugriffe zu reduzieren. In seiner aktuellen Version beschränkt sich der Tuning-Plan jedoch ausschließlich auf die Erhöhung der Bufferpool-Größe. Die dynamische Verringerung ist vorerst nicht Bestandteil unserer Untersuchungen. Ausgelöst wird der Tuning-Plan durch Auftreten eines entsprechenden Ereignisses, das eine Unterschreitung des Workload-abhängigen Schwellwertes für die „bufferpool hit ratio" (BPHR)[47] beschreibt (siehe Tabelle 8.1).

Mit Hilfe der Aktivität „CBE importieren" erfolgt der Zugriff auf die den Problemkontext beschreibenden Metadaten, aus welchen anschließend mittels der globalen Aktivität „CBE auslesen" die Problem-Datenbank sowie der Problem-Bufferpool ausgelesen werden. Damit stehen dem Tuning-Plan die Metadaten sowohl des Event Correlators als auch des Tuning Plan Selectors zur Verfügung.

Anschließend erfolgt die Erhöhung der Bufferpool-Größe um 1000 Seiten und damit in Folge auch die einhergehende Verbesserung der BPHR. Die in den Tuning Plan Executor eingebaute Ressourcen-Beschränkungsfunktionalität verhindert dabei übermäßige Ressourcen-Allokationen durch Beachtung der vorgegebenen benutzerdefinierten Obergrenzen.

8.3.2.2 Tuning des Sortier-Bereichs

DB2 führt Sortier-Operationen durch sobald in dem abgesetzten SQL-Statement ein explizites ORDER BY, GROUP BY, UNION oder DISTINCT vorkommen. Auch bei OLAP- bzw. Aggregationsfunktionen sowie bei der Reorganisation, der Index-Erzeugung oder bei Hash Joins[48] beanspruchen DB2-Agenten Speicher in einem Sortheap für ihre Sortierungen.

Für Multiprozessor-Architekturen bietet DB2 Möglichkeiten zur parallelen Query-Abarbeitung. Mit dem durch einen Konfigurationsparameter aktivierbaren Partitionsinternen Parallelismus werden Anfragen, die sich auf eine einzige Partition erstrecken, durch mehrere Prozessoren gleichzeitig bearbeitet. Alle Sortierungen erfolgen dabei im gemeinsamen Speicher (Database Shared Memory). Man spricht in dem Zusammenhang von sog. Shared Sorts. Im Gegensatz dazu handelt es sich um Private Sorts, wenn dieser Parallelismus inaktiv ist und jeder DB2-Agent zum Sortieren auf seinen privaten Speicherbereich (Agent Private Memory) zugreift.

Der DB-Konfigurationsparameter SORTHEAP bestimmt dabei, wieviel Speicher (Agent Private Memory bzw. Database Shared Memory) für jede einzelne Sortierung verwendet wird. Reicht der durch SORTHEAP definierte Speicher für die Sortierung nicht aus, kommt es zu einem kostspieligen Überlauf (Sort Overflow) und die Sortierung muss in einen Temporary Tablespace ausgelagert werden. Von einem Hash Join Small Overflow wiederum spricht man, wenn der Sortheap für den Hash Join unzureichend ist, aber der Platz nicht, wie bei den Hash Join Overflows, um mehr als 10% überschritten wurde.

Bei mehreren in einer Instanz gleichzeitig stattfindenden Shared Sorts legt der Konfigurationsparameter des Datenbankmanager SHEAPTHRES die obere Grenze für den gemeinsamen Speicherverbrauch fest, so dass keine weiteren Sortheap-Allokationen zulässig sind, wenn

---

[47] Die Bufferpool Hit Ratio beschreibt das Verhältnis von logischen (über Cache) und physischen (über Sekundärspeicher) Datenbankzugriffen und ist damit ein verlässliches Maß für die Performance von (I/O-dominierten) Anfrage-Ausführungen.

[48] Der Hash Join ist eine effiziente Umsetzung von Equi-Joins, die auf der Verwendung einer speicherresidenten Hash-Tabelle beruht.

dies zu einer Überschreitung der Grenze führen würde. Für Private Sorts ist eine Überschreitung dennoch möglich. Dabei wird ein Speicher-Drosselungsmechanismus aktiviert, so dass bei nachfolgenden Sortier-Operationen weniger Speicher als angefordert allokiert wird. Durch eine kontinuierliche Halbierung des zu allokierenden Sortier-Speichers hält sich das Ausmaß der Überschreitung in Grenzen. Die Sortierungen bzw. die Hash Joins, die außerhalb des durch SHEAPTHRES beschränkten Bereichs erfolgen, werden als Post Threshold Sorts bzw. Post Threshold Hash Joins bezeichnet.

Auf Datenbank-Ebene ist eine Speicher-Begrenzung für Sortiervorgänge durch den Wert des DB-Konfigurationsparameters SHEAPTHRES_SHR möglich. Dazu muss der Parameter SHEAPTHRES der Instanz-Ebene auf den Wert 0 gesetzt werden.

Der in **Abbildung 8.7b** dargestellte Sortheap-Tuning-Plan wird abhängig von der aktuellen Workload durch die einen Schwellwert verletzende atomare Ereignisse „overflowed sorts" (OS), „hash join overflows" (HSO) oder „hash join small overflows" (HJSO) ausgelöst. Durch Erhöhen des für nachfolgende Sortierungen allokierbaren Speichers (SORTHEAP) um 20% für OLTP bzw. 40% für OLAP sollen künftig derartige Überläufe vermieden werden.

Der Sheapthres-Tuning-Plan aus **Abbildung 8.7c** wird jedesmal dann ausgeführt, wenn die Schwellwerte für die Metriken „post threshold hash joins" (PTHJ) bzw. „post threshold sorts" (PTS) verletzt wurden. In Abhängigkeit von der aktuellen, durch den Workload Classifier bestimmten Workload wird ermittelt, um wie viel Prozent der Wert des entsprechenden Konfigurationsparameters zu erhöhen gilt. Die globale, zusammengesetzte Aktivität „Sorting-Rekonfig" führt die Parameteranpassung auf Datenbank-Ebene (SHEAPTHRES_SHR) bzw. Instanzebene (SHEAPTHRES) in Abhängigkeit vom aktuellen Wert des Konfigurationsparameters SHEAPTHRES durch.

8.3.2.3 Index-Design

Die Idee hinter diesem auf dem bereits in [WRRA08] vorgestellten Tuning-Plan basierenden und um zusätzliche Funktionalitäten erweiterten Index-Design-Tuning-Plan (IDTP) besteht darin, den IBM DB2 Design Advisor [IBM06c] Ereignis-gesteuert aufzurufen und ihn dabei mit Informationen über die aktuelle Workload zu versorgen (**Abbildung 8.7d**). Die vom DB2 Design Advisor erstellten Vorschläge (zum Anlegen bzw. Löschen von Indexen sowie zum Aktualisieren von Statistiken) sollen anschließend an einen DBA zur Qualitätssicherung (z.B. Prüfung oder Freigabe) per E-Mail weitergeleitet werden. Nach Prüfung der erstellten Vorschläge kann der DBA diese automatisch ausführen lassen oder verwerfen. Der Tuning-Plan soll immer dann ausgeführt werden, wenn aufgrund fehlender Indexe bedeutend mehr Tupel vom DBMS gelesen werden mussten als in der Anfrage-Ergebnismenge enthalten sind. Ein Indikator hierfür ist eine Schwellwertüberschreitung des auf den DB2-Metriken ROWS_READ und ROWS_SELECTED basierenden Verhältnisses „avg number of rows read per selected row" (RRRS).

Zur Bestimmung der relevanten Workload werden zunächst die fünf bezüglich der RRRS schlechtesten Select-Statements der letzten Minuten sowie die Häufigkeiten ihres Auftretens ermittelt. Um die durch Anlegen eines Index entstehenden Folgekosten bei Änderungs-Operationen (Insert, Update, Delete) zu berücksichtigen, sollen zusätzlich die entsprechenden IUD-Statements zusammen mit ihren Häufigkeiten für den Aufruf des DB2 Design Advisors bestimmt werden.

Die globale Aktivität „SQL-Anfrage" erlaubt es, SQL-Anfragen gegen beliebige Datenbanken abzusetzen. Sie wird hierbei verwendet, um historische Performance-Daten über den

Dynamic Statement Cache[49] aus der PE-internen Datenbank abzufragen. Der vorher aus dem CBE ermittelte Datenbankname dient dabei der Einschränkung des Anfrage-Ergebnisses bei der Ermittlung der fünf in Bezug auf die RRRS schlechtesten SQL-Statements innerhalb der letzten Minuten sowie deren Häufigkeiten.

In der zusammengesetzten, lokalen Aktivität „Input aufbereiten" werden in einer Schleife alle von den ermittelten Select-Statements verwendeten Tabellen ausgelesen (lokale Aktivität „Statement parsen") und alle IUD-Statements bestimmt (globale Aktivität „SQL-Anfrage"). Anschließend sorgt die ebenfalls zusammengesetzte, lokale Aktivität „db2advis-Input erstellen" zunächst dafür, dass aus allen IUD-Statements nur jene herausgefiltert werden, die sich auf die betroffenen Tabellen der fünf Select-Statements beziehen. Zudem werden sowohl die Select- als auch die herausgefilterten IUD-Statements zusammen mit ihren Häufigkeiten in einer Eingabedatei für den DB2 Design Advisor abgespeichert.

Die erzeugte Datei mit der relevanten Workload bildet nun den Input für den Aufruf des DB2 Design Advisors durch die globale Aktivität „db2advis aufrufen". Dabei wird die Ausführungszeit des DB2 Design Advisors und damit die erwartete Qualität der Advisor-Ergebnisse in Abhängigkeit von der anliegenden (und erkannten) Workload gesteuert. Für eine OLAP-ähnliche Workload erscheint es sinnvoll, dem Advisor mehr Zeit für seine Berechnungen zur Verfügung zu stellen und ggf. sogar materialisierte Sichten berücksichtigen zu lassen.

Nach Ausführung des DB2 Design Advisors werden dessen Vorschläge durch die bereits von WsM bereitgestellte Aktivität „E-Mail senden" an den DBA per E-Mail in aufbereiteter Form gesendet. Die WsM-Workflow-Engine wartet nun eine vorgegebene Zeitspanne auf eine Rückmeldung des DBA. Umgesetzt wird dies durch die zum Standardumfang von WsM gehörende Aktivität „Auf Input warten". Der DBA kann nun innerhalb einer vorgegeben Zeitspanne entscheiden, ob er die Vorschläge des DB2 Design Advisors akzeptiert oder verwirft. Verwirft er die Vorschläge oder verstreicht die vorgegebene Zeit, so endet der Tuning-Plan ohne Veränderung des Datenbank-Zustands. Andernfalls wird ein vom DB2 Design Advisor erstelltes Skript zur Umsetzung der Design-Entscheidungen ausgeführt (globale Aktivität „Skript ausführen").

In der Zeitspanne zwischen dem Versenden der E-Mail und der Reaktion des DBA sind Workload-Shifts möglich, so dass vorgeschlagene Indexe ggf. keinen Einfluss auf die (mittlerweile) neue Workload haben. Um diesem Problem entgegenzuwirken, ist ein Kompromiss zwischen der Dauer bis zum Timeout, der Häufigkeit des Aufrufens des DB2 Design Advisors und der Größe der Workload-Datei, d.h. der Anzahl der zu betrachtenden Select- und entsprechenden IUD-Statements zu finden.

Im Rahmen künftiger Index-Tuning-Erweiterungen ist zu untersuchen, inwieweit sich einige der in den letzten Jahren entwickelten Ansätze zur Index-Auswahl [SGS03, BrCh06] in Tuning-Pläne integrieren lassen.

8.3.2.4 Weitere Tuning-Pläne

Neben den bereits aufgeführten Tuning-Plänen gibt es noch eine Reihe weiterer Workflows. Diese umfassen in dem Bereich des Locking u.a. Betrachtungen zu Sperr-Eskalationen und Warte-Situationen aufgrund lang gehaltener Sperren bzw. Deadlocks.

---

[49] In dem DB2 Dynamic Statement Cache befinden sich alle zum Abrufzeitpunkt aktiven dynamischen SQL-Statements. Der PE speichert diesen Zustand in periodischen Abständen und erlaubt damit die Bildung einer Historie aller abgesetzten SQL-Statements.

Nach „schwerwiegenden“ Aktionen, die möglicherweise beträchtliche Auswirkungen auf die Anfrageoptimierung haben (z.B. das großflächige Index-Anlegen/Löschen auf Empfehlung des Design Advisor), kann oftmals eine Aktualisierung der Statistiken sinnvoll sein. Dies erfolgt durch einen Tuning-Plan, der automatisch alle betroffenen Tabellen sowie Indexe bestimmt und ein DB2 RUNSTATS ausführt. Der Plan kann durch ein Ereignis gefeuert oder durch einen anderen Tuning-Plan aufgerufen werden.

Tuning-Pläne können auch periodisch ausgeführt werden. Beispielsweise bestimmen wir in bestimmten Zeitabständen die im Sinne der Ausführungszeiten (execution time) und Anzahl an Ausführungen teuersten (TopN) Statements. Das Anfrage-Ergebnis wird materialisiert und im Anschluss dem Administrator zur weitergehenden Problem-Diagnose per E-Mail zugänglich gemacht.

Aber auch auf (Meta-)Events des ATE wird reagiert. Es gibt bspw. einen Tuning-Plan, der im Falle eines erkannten Workload-Wechsels und dem damit erzeugten Workload-Shift-Event ausgeführt wird. Seine Aufgabe ist das Umsetzen einiger wichtiger, an die neue Workload angepasster Initialparameter (wie etwa Grad des Intra-Partitions-Parallelismus, die Anzahl Agenten zum asynchronen Füllen und Leeren des Bufferpool etc.). Das dem Tuning-Plan zugeordnete Workload-Shift-Event feuert im Vergleich zu den „normalen“, auf Exceptions basierenden Events relativ selten. Daher ist es auch denkbar Konfigurationsparameter mit einzubeziehen, die online nicht ohne Weiteres änderbar sind. Bevor jedoch derartige Parameter mit den entsprechenden Beeinflussungen des operativen Betriebs automatisch geändert werden, sollte bei dem DBA eine Bestätigung bzw. ein Termin zur Durchführung der Umkonfiguration eingeholt werden.

Weitere, im Rahmen des Kooperationsprojektes mit der IBM identifizierte, DB2-spezifische Probleme sowie entsprechende System-technische und Anwendungs-bezogene Tuning-Maßnahmen und Check-Listen für einen auf DB2 basierenden Software-Stack können in [RWRA08, WiRa07, WRRA08] eingesehen werden.

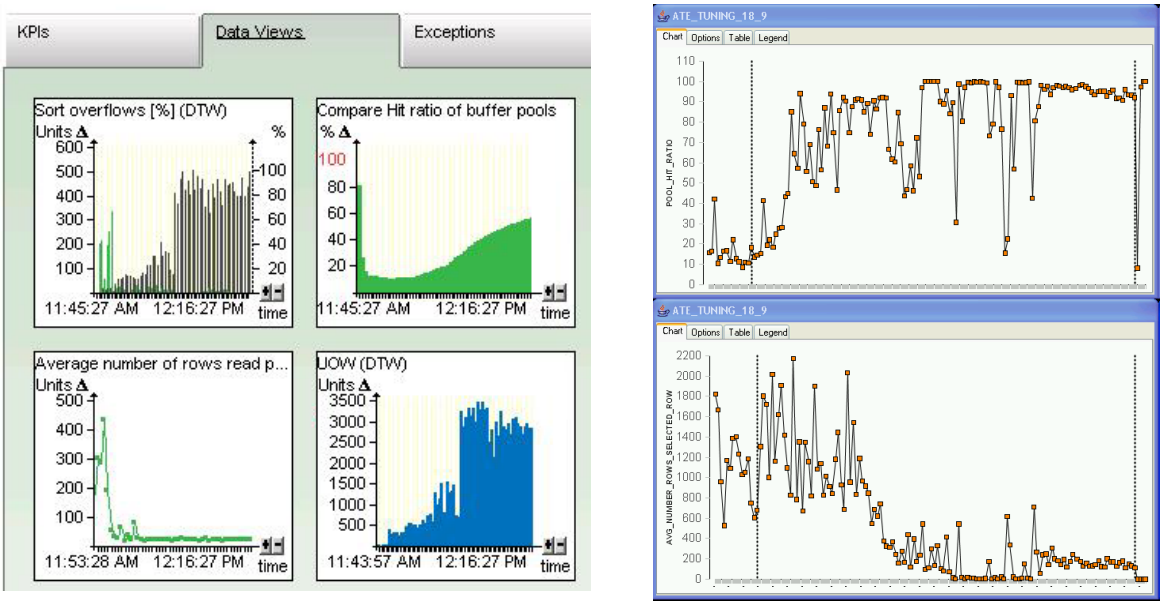
8.3.3 Evaluations-Ergebnisse

Seitens der Implementierung ist es möglich alle beschriebenen Komponenten der ATE-Architektur auf einem Rechner zu integrieren. Wir haben uns im Rahmen der evaluierenden Testläufe jedoch für eine Ressourcen-schonendere drei-schichtige Verteilung entschieden. Die zu optimierende Datenbank befindet auf einem Rechner. Auf einem anderen Rechner wird der Workload-Mix erzeugt und in Form einzelner SQL-Statements per remote-Verbindung an den Datenbank-Server gesendet. Zur Generierung der Workload nutzen wir einen auf unsere Bedürfnisse zugeschnittenen IBM-eigenen Workload Generator (siehe Abschnitt 7.4.6). Die dritte Schicht umfasst den ATE und ist verantwortlich für die Sammlung, Speicherung und Auswertung der Performance-Daten, die Erkennung und Korrelation von Events sowie die automatische Auflösung derartig erkannter Performance-Probleme mittels Tuning-Plänen.

Die Remote-Fähigkeit des auf standardisierten Schnittstellen basierenden ATE-Frameworks erlaubt prinzipiell auch die Auslagerung einzelner ATE-Komponenten auf entfernte Systeme, wodurch der zusätzliche Overhead auf dem Produktivsystem minimiert werden kann. Derzeit erscheint eine solche Maßnahme lediglich für den rechenintensiven Workload Classifier sinnvoll (siehe Abschnitt 7.4).

Zur Laufzeit bzw. in einer retrospektiven Nachbereitungsphase können die relevanten Performance-Indikatoren und wesentliche ATE-Metadaten (wie z.B. ATE-Start bzw. -Stop, Zeitpunkte der Erkennung von Events und Ausführungszeitpunkte der korrespondierenden Tuning-Pläne) mit Hilfe unseres **KPI Visualizer** veranschaulicht und für eine Evalua-

tion herangezogen werden. Das Tool erlaubt den Vergleich von bis zu vier Performance-Metriken und bietet eine übersichtliche Darstellung des zeitlichen Verlaufs der ausgewählten Metriken. Darüber hinaus ist es möglich, interaktiv den Betrachtungs- und Darstellungszeitraum anzupassen und somit einen feingranulareren Werteverlauf darzustellen. **Abbildung 8.8** veranschaulicht die Möglichkeiten zur visuellen Interpretation des Tuning mittels KPI Visualizer und Performance Expert. Die „Data Views"-Komponente des PE erlaubt die Metrikverfolgung über einen gewissen Zeitraum in einem in Darstellungsart anpassbaren Diagramm. Beide Tools können zudem die Werte auch in tabellarischer Form präsentieren.



a) PE system overview **b)** ATE KPI Visualizer tool

Abbildung 8.8: ATE-Tools zur Visualisierung der Performance-Daten

Initial wurde die zu optimierende Datenbank (DBUC) mit einer vom DB2 Configuration Advisor (siehe Abschnitt 3.1.1) vorgeschlagenen Konfiguration erzeugt. Nach dem Deaktivieren der DB2-internen autonomen Mechanismen wurde die neu erzeugte Datenbank mit den entsprechenden OLAP- bzw. OLTP-Daten gefüllt. Um insbesondere die Effektivität des Index-Design-Tuning-Plans besser bewerten zu können, haben wir lediglich die jeweils 9 OLTP- bzw. 8 OLAP-Primärschlüssel-Indexe angelegt und auf weitere verzichtet.

Tuning-Plan (TP)	**Einfluss des Tuning-Plans**	**(Initial-)Wert / Anzahl vor dem Tuning**	**Wert / Anzahl nach dem Tuning**
Bufferpool-TP	Größe des Bufferpools	17354 Seiten	36354 Seiten
Sorting-TP	Größe von SORTHEAP	754 Seiten	40651 Seiten
	Größe von SHEAPTHRES	2917 Seiten	52256 Seiten
Index-Design-TP	OLTP-Indexe	9	21
	OLAP-Indexe	8	33

Tabelle 8.2: Auswahl umgesetzter Tuning-Pläne und deren Wirkung

Beide im Folgenden beschriebenen Test-Szenarien (ohne und mit aktiviertem ATE-Tuning) basieren auf der gleichen Initial-Konfiguration. Durch einen Neustart der DB2-Instanz vor jedem Test-Szenario haben wir dafür Sorge getragen, dass Caches und Heaps

geleert wurden. Die Testläufe sind zur Veranschaulichung mit einer Auswahl an den elementaren Tuning-Plänen aus Abschnitt 8.3.2 durchgeführt wurden. **Tabelle 8.2** stellt eine für die Evaluations-Ergebnisse relevante Auswahl an Tuning-Plänen sowie Art und Umfang ihres Einflusses auf die zu tunende Datenbank dar.

Die Tuning-Pläne beschränken sich im Rahmen der Machbarkeitsstudie zunächst ausschließlich auf Modifikation bzw. Erhöhung von Konfigurationsparameter-Werten[50]. Wir halten die Hardware fix, die Ressourcen (Haupt-/Festspeicher, CPU etc.) des Systems beschränkt und rücken somit die Frage nach ihrer möglichst effizienten Aufteilung in den Mittelpunkt. Die in diesem Zusammenhang potentiell auftretenden Ressourcen-Überallokationen werden durch das Ressourcen-Restriktionskonzept des Tuning Plan Executors unter Beachtung nutzerdefinierter Obergrenzen zurückgewiesen (siehe Abschnitt 8.3.1). Wir gehen darüber hinaus auch davon aus, dass die Ursachen für Probleme nicht im Versagen bzw. Ausfall von Hardware oder Software begründet sind.

Die Effektivität des Tunings wurde an einigen Performance-relevanten Metriken gemessen. Dazu zählen die „units of work", welche den Systemdurchsatz, sprich die Zahl durchgeführter Transaktionen, repräsentiert; die „bufferpool hit ratio; die Anzahl von Sortierungen („total sorts"); sowie das Verhältnis von gelesenen Tupeln zu Ergebnistupeln („rows read / rows selected") als Indikator für die Indexausnutzung [IBM06i].

Jedes Diagramm der **Abbildung 8.9** veranschaulicht die Werteverläufe der entsprechenden Metrik sowohl ohne als auch mit Tuning. Aus Gründen der Übersichtlichkeit haben wir auf die Darstellung gesonderter Geschehnisse (wie ATE-Start, Problem-Erkennung und Tuning-Plan-Ausführung) verzichtet. Die durch die Generierung der Workload auf unserem Testsystem ohne Tuning entstandenen Werteverläufe können als Vergleichsgrundlage für das Szenario mit ATE-Tuning dienen. Beide Testläufe haben eine Dauer von 5 Stunden und 15 Minuten und beinhalten zwei Workload-Shifts: nach einer 105-minütigen OLAP- folgte eine 105-minütige OLTP- und anschließend wieder eine 105-minütige OLAP-Workload-Phase.

Das erste Diagramm stellt die Werteverläufe der Metrik „bufferpool hit ratio" (BPHR) dar. Der unter ATE-Tuning entstandene Werteverlauf der BPHR charakterisiert sich durch kontinuierliches Ansteigen. Unmittelbar nach Start des ATE-Prototypen liegt der Wert der BPHR bei ca. 25%, wodurch der Schwellwert für das Auslösen des atomaren BPHR-Ereignisses verletzt ist. Als Reaktion auf das Unterschreiten des Schwellwertes wird der Bufferpool-Tuning-Plan angestoßen, welcher die Größe des Bufferpools um 1000 Seiten erhöht und somit eine steigende BPHR verursacht. Bereits nach wenigen Ausführungen des Tuning-Plans steigt die BPHR bis auf ca. 85%. Bei jedem kurzfristigen Unterschreiten der 85%-Grenze wird die Bufferpool-Größe erhöht, so dass die BPHR am Ende der ersten OLAP-Phase bei ca. 95% liegt. Durch den Workload-Shift bedingt, fällt die BPHR jedoch auf ca. 30%, da die entsprechenden angefragten Seiten sich zunächst nicht im Bufferpool befinden. Sowohl durch das allmähliche Füllen des Bufferpools als auch durch die kontinuierliche Erhöhung der Bufferpool-Größe steigt die BPHR wieder an, bis sie sich gegen Ende der OLTP-Phase bei ca. 97% befindet. Bei dem nächsten Workload-Wechsel ist wieder ein Einbruch der BPHR zu verzeichnen. Hier profitiert die BPHR jedoch von den in den beiden vorangegangenen Perioden erfolgten Bufferpool-Größen-Erhöhungen und steigt auf über 98% in einen Sättigungsbereich ohne nennenswerte Schwankungen. Im Verlauf des Tunings steigt die Größe des Bufferpools von initial 17354 Seiten auf 36354 Seiten. Es ist jedoch anzumerken, dass in einem realen Anwendungs-Szenario jedem Workload-Typ typischerweise mind. ein eigener Bufferpool zuzuordnen ist.

[50] Erklärungen zu den jeweiligen Konfigurationsparametern finden sich in [IBM06c].

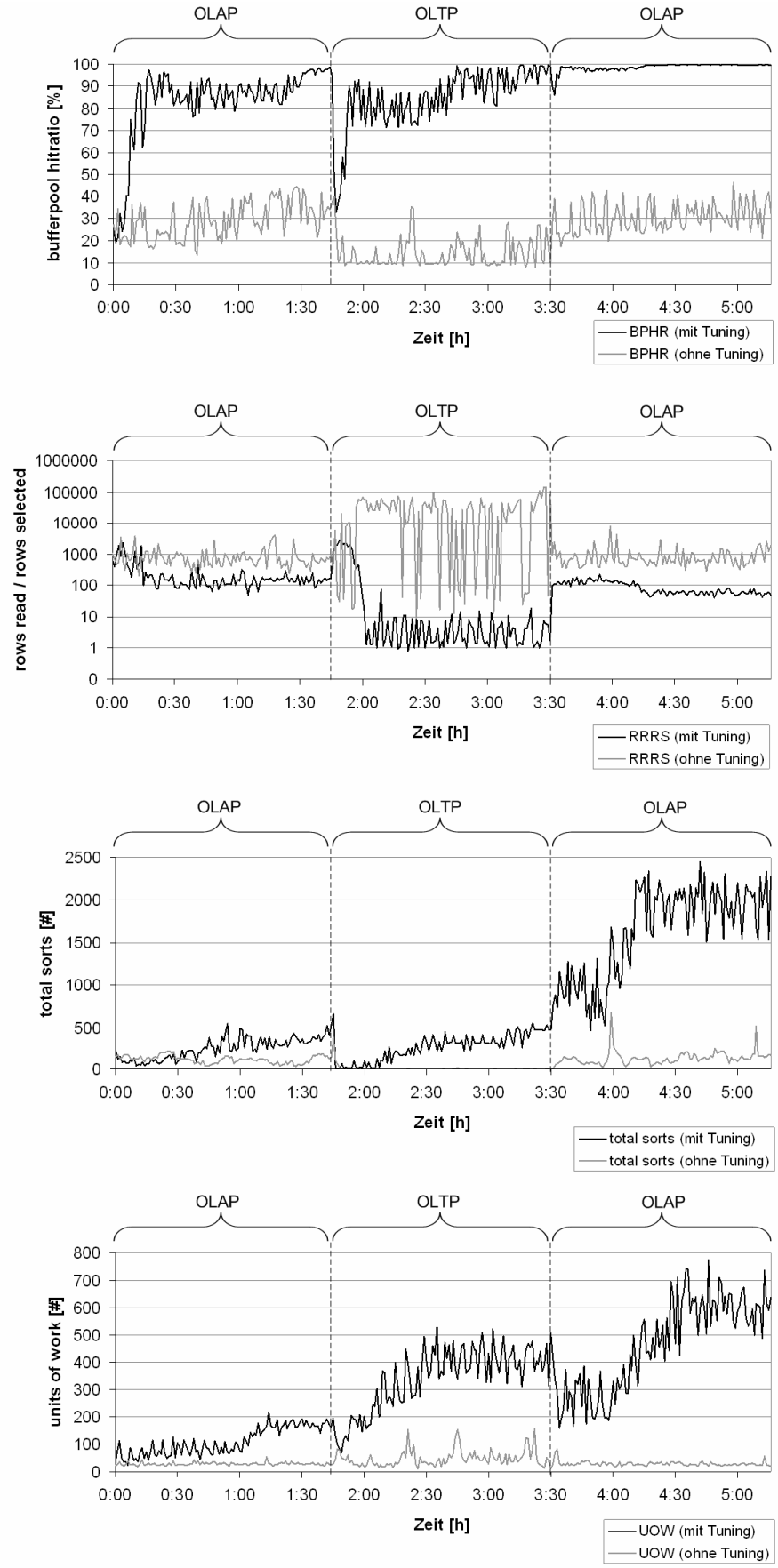


Abbildung 8.9: Ausgewählte Ergebnisse der Testläufe

Die Werteverläufe im zweiten Diagramm stellen das Verhalten der Metrik „rows read / rows selected“ (RRRS) ohne und mit Tuning unter wechselnder Workload dar. Im Gegensatz zu den anderen Diagrammen weist die Ordinate eine logarithmische Skalierung auf. Höhere Werte sind ein Indikator für schlechte Performance bzw. schlechte Indexausnutzung oder gar nicht vorhandene Indexe. Da anfänglich bewusst nur die Primärschlüssel-Indexe auf den Tabellen angelegt wurden, verwundert es nicht, dass RRRS ohne Tuning gänzlich hohe Werte annimmt und somit zu regelmäßigen Ereignis-Auslösungen und damit verbundenen Tuning-Plan-Ausführungen bei aktiviertem ATE-Tuning führt. Jeder Aufruf des DB2 Design Advisors liefert Indexvorschläge, die (zu Testzwecken ohne Nachfrage beim Administrator) unmittelbar umgesetzt werden, was eine allmähliche Steigerung der Performance bzw. die fallende Tendenz der RRRS-Metrik sowohl bei OLAP als auch bei OLTP impliziert. Aufgrund der Natur der OLTP-Statements sind die durch die Index-Erzeugung auftretenden Effekte bereits innerhalb der ersten Minuten deutlich sichtbarer als in den beiden OLAP-Phasen. Die RRRS-Metrik nähert sich dabei durch das Anlegen der 21 Indexe zügig dem Optimum von 1. Ähnlich den anderen Diagrammen setzt der Werteverlauf der zweiten OLAP-Phase dort an, wo der der ersten aufgehört hat.

Auch die Metrik „total sorts“ hat einen ähnlichen Verlauf. Die Anzahl der Sortiervorgänge lässt sich durch das Sort-Tuning im Speziellen und aufgrund des gesteigerten Gesamtdurchsatzes im Allgemeinen kontinuierlich steigern. Nach dem ersten Workload-Shift fällt der Wert der Metrik Workload-bedingt zwar abrupt, steigt jedoch anschließend auch hier kontinuierlich weiter.

Man erkennt unschwer im letzten Diagramm, dass die Metrik „units of work“ (UOW) unmittelbar nach Aktivieren des Tunings kontinuierlich steigt. Der UOW-Verlauf wird dabei durch die Gesamtheit aller Tuning-Pläne geprägt und dient uns daher als Indikator für die Performance des gesamten Systems. Auch dem durch den ersten Workload-Shift entstehenden Einbruch kann der ATE innerhalb weniger Minuten und einiger Tuning-Plan-Ausführungen erfolgreich entgegenwirken. Nach dem zweiten Workload-Shift ist ein erneuter Einbruch zu erkennen. Die Metrik fällt zunächst auf ihr Niveau am Ende der ersten OLAP-Phase zurück, steigt dann aber analog zur Sortiermetrik wieder rapide an.

8.4 Erweiterung der ATE-Infrastruktur

In diesem Abschnitt wird beschrieben, wie die Architektur des Autonomic Tuning Expert angepasst werden kann, um die in Abschnitt 8.2 beschriebene Monitoring-Architektur als Quelle für Performance-Analysen zu integrieren. Für den Zweck der Einführung einer neuen autonomen Ebene gehen wir kurz auf zusätzlich notwendige bzw. Anpassungen bestehender Komponenten und Schnittstellen ein. Des Weiteren wird beschrieben, welche Erweiterungen der Tuning-Pläne bzw. -Schritte sinnvoll und notwendig erscheinen. Für tiefergehende Details sei auf [Köh10] verwiesen.

Die nachfolgende **Abbildung 8.10** stellt eine mögliche Gesamtarchitektur einer derartigen erweiterten autonomen Monitoring- und Tuning-Infrastruktur sowie die noch nicht bereits in Abschnitt 8.2 beschriebenen Daten- und Kontrollflüsse zwischen den beteiligten Komponenten dar. Gestrichelte Linien symbolisieren Kontrollflüsse, durchgezogene Linien hingegen Datenflüsse. Sämtliche neue Monitoring- und ATE-Komponenten sind goldfarben hervorgehoben. Server Agent Interface und Warehouse Interface wurden der Einfachheit halber auch ausgelassen.

8.4.1 Schnittstellen zur Kommunikation zwischen ATE und PWH

Für die Interaktion und Kommunikation zwischen den (operativen) Datenquellen, der Monitoring-Architektur und den Komponenten des ATE sind die beiden bereits in Abschnitt 8.2 für den DBA eingeführten Schnittstellen vorhanden. Wie auch für einen DBA so ermöglicht das **PWH Interface** den Zugriff auf die Performance-Daten des Performance Warehouse sowie die Konfiguration der Komponenten. Das PWH Interface bspw. erlaubt dem Workload Classifier (vgl. Abschnitt 8.3.1 bzw. 7.4) mit Hilfe der Performance-Daten des Performance Warehouse die aktuelle Workload festzustellen. Diese kann anschließend von ihm in den Short Term Storage des PWH geschrieben werden. Auch der Event Collector nutzt diese Schnittstelle, um die von ihm erkannten und in das CBE-Format umgewandelten Events in den Short Term Storage des PWH zu transferieren. Der Event Correlator wiederum kann über das PWH Interface auf den Short Term Storage, Long Term Storage und Cube Storage des PWH zugreifen, um Kurzzeit- und Langzeit-Analysen zur Event-Ermittlung durchführen zu können. Für sehr kritische Events hingegen soll es auch möglich sein, direkt auf die Echtzeit-Performance-Daten der Client Agents auswertend zuzugreifen und ggf. unmittelbar zu reagieren.

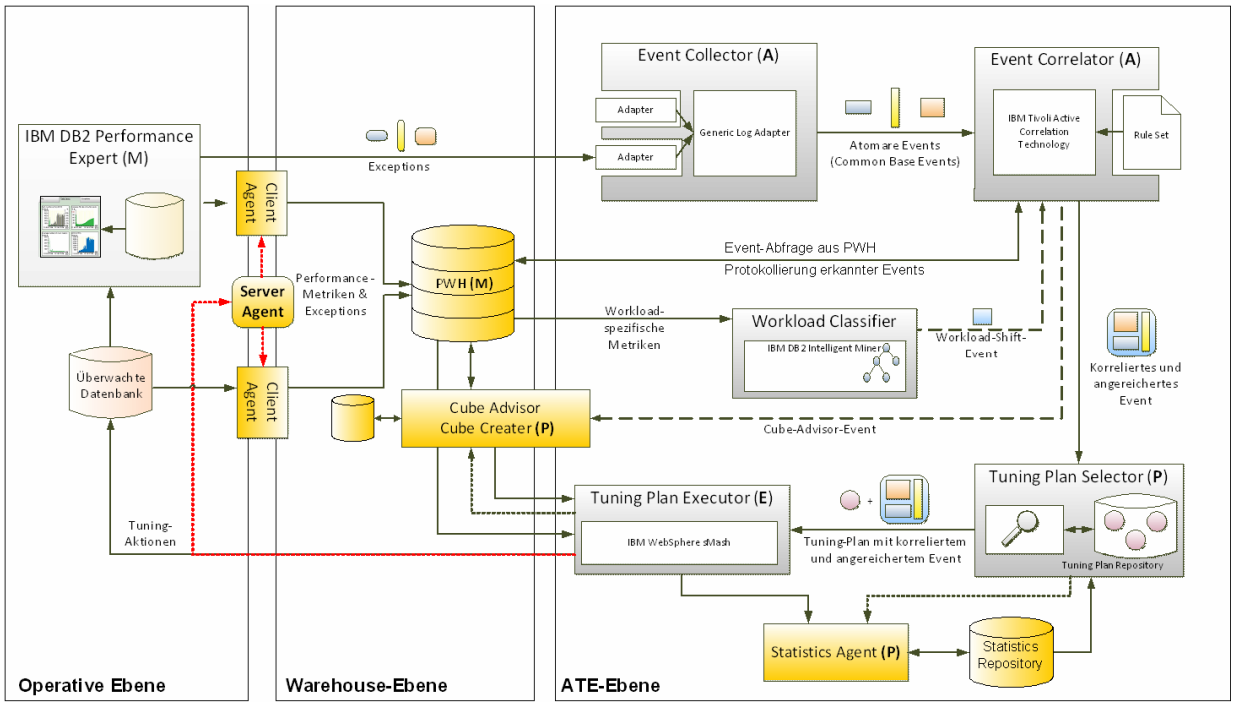


Abbildung 8.10: Anbindung der Warehouse-Ebene an die Autonome ATE-Ebene

Das **Server Agent Interface** erlaubt sowohl dem DBA als auch dem ATE, den Server Agent und dessen Komponenten zu konfigurieren bzw. das Ausführen eines getriggerten ETL-Jobs zu veranlassen. Der Workload Classifier bspw. kann über den Server Agent einen ETL1-Job zur regelmäßigen, gezielten Sammlung von Performance-Daten veranlassen, die anschließend im PWH abgelegt und für die Bestimmung der Workload genutzt werden können. Der Event Collector kann über dieses Interface mit den Client Agents kommunizieren, um die von ihnen empfangenen Events in das CBE-Format umzuwandeln und in der eigenen Warteschlange sowie über das PWH Interface im PWH „dauerhaft“ abzulegen. Ebenso hat er die Möglichkeit, einen ETL1-Job ausführen zu lassen, um auf Basis dieser Daten weitere Events bestimmen zu können.

8.4.2 Zusätzliche Monitoring-spezifische ATE-Komponenten

Neben den bereits in Abschnitt 8.2 vorgestellten ATE-Komponenten sehen wir insbesondere im Zusammenhang einer gesamtheitlichen Monitoring- und Tuning-Infrastruktur die nachfolgenden zusätzlichen Komponenten.

8.4.2.1 Statistics Agent

Durch die Einführung eines **Statistics Agent** als neue Komponente in der ATE-Architektur können mehrer andere Komponenten in ihren Wirkungsweisen profitieren.

Der Statistics Agent soll den **Tuning Plan Selector** bei der Auswahl von Tuning-Plänen unterstützen. Hierzu wird für jeden Tuning-Plan nach der Ausführung und dem Abwarten der Wirkungszeit dessen quantifizierbarer Nutzen bestimmt und in den Metadaten der Tuning-Pläne hinterlegt. Die Analyse kann hierbei automatisch oder manuell durch den DBA erfolgen. Für bestimmte Kategorien von Tuning-Plänen sind eigene Effizienz-Maße einzuführen. Der Tuning Plan Selector kann schließlich die durchschnittliche Effizienz eines Tuning-Plans bei der Priorisierung mehrerer Tuning-Pläne für ein Event berücksichtigen. Die Bewertung kann auch dabei helfen, Tuning-Pläne zu identifizieren und zu deaktivieren, deren Wirkungsgrad nachgelassen hat oder noch nie zufriedenstellend vorhanden war.

Mit Hilfe der **Tuning-Plan-Executor-Historie** kann der Statistics Agent in regelmäßigen Abständen oder auch Ereignis-gesteuert feststellen, wann welcher Tuning-Plan über welche Effektor-Schnittstellen wie lange ausgeführt wurde. Durch Vergleichen des in Form von Performance-Metriken im PWH gespeicherten Ressourcen- bzw. System-Zustands vor und nach der Ausführung kann die Wirkung des Tuning-Plans bzw. der Grad der Verbesserung oder Verschlechterung ermittelt werden.

Über den Zugriff auf das **Tuning Plan Repository** des Tuning Plan Selectors kann die Information über die ermittelte Effizienz eines Tuning-Plan in dessen Metadaten geschrieben und somit dem Tuning Plan Selector zur nachfolgenden Planung und Priorisierung zur Verfügung gestellt werden. Der Statistics Agent hat darüber hinaus die Aufgabe, die anderen Tuning-Plan-Metadaten zur Laufzeit zu aktualisieren und somit für eine akkuratere Planung bereitzustellen.

Der Zugriff auf die im PWH gespeicherten Daten über den System-Zustand, der für die Bestimmung der Effizienz und des Nutzens eines Tuning-Plans benötigt wird, erfolgt über das **PWH Interface**.

Der Statistics Agent schreibt ausführliche Daten für die Ermittlung von Erfolg und Effizienz der Tuning-Pläne in sein **Statistics Repository**. Dazu gehören u.a. die Anzahl der Aufrufe/Ausführungen eines Tuning-Plans, das Verhältnis erfolgreicher zu erfolgloser Ausführungen, die angesprochenen Ressourcen uvm. Eine weitere sehr wichtige Statistik zu Tuning-Plänen sind die dabei entstandenen Kosten. Diese werden üblicherweise in CPU-Zeit und I/O abgerechnet. Alternativ können alles diese Statistiken auch in den Kopf, sprich in die Metadaten, einzelner Tuning-Pläne aufgenommen und stetig aktualisiert werden. Um die Tuning-Pläne nicht aufzublähen, empfiehlt sich jedoch erste Variante.

8.4.2.2 Cube Advisor

Die in Abschnitt 6.5 eingeführte Erfindung behandelt ein Advisor-Tool zur Nutzer-gesteuerten und automatisierten Auswahl und Erzeugung multidimensionaler Datenmodelle (Cubes). Der Advisor hat dabei Wissen über potentielle Performance-Probleme auf dem untersuchten und durch den Performance-Monitor überwachten System. Er hat zudem Kenntnis darüber, welches Performance-Problem mit welchen Cube-Strukturen am geeig-

netsten zu analysieren ist. Das hierfür nötige Wissen kann initial von den Administratoren in das System eingespeist und später durch eine kontinuierliche Beobachtungs- und Lernphase aktuell gehalten und erweitert werden. Darüber hinaus ist der Advisor in der Lage, die zur erkannten Situation passenden Cubes automatisch zu erstellen und mit den Daten aus dem PWH zu füllen. Sind die Strukturen erzeugt und befüllt, können die DBAs mit der Analyse des Performance-Problem via interaktiver BI-Analyse-Tools oder OLAP-Sprachkonstrukten beginnen.

Der Cube Advisor basiert im Kern auf einer Wissensbasis (**Performance Cube Knowledge Base**), die im Grunde die folgenden Informationen bereithält:

- Die Struktur der potentiellen Performance Cubes (cube meta model)
- Das Wissen zur Befüllung dieser Cubes aus dem relationalen Performance-Daten-Bestand (mapping meta model)
- Die Abhängigkeiten zwischen Performance-Problemen und assoziierten Cubes
- Eine Historie über die Zahl der Vorschläge, die jeweilige Nutzung und den Erfolgsfaktor der Cubes für die Problem-Analyse

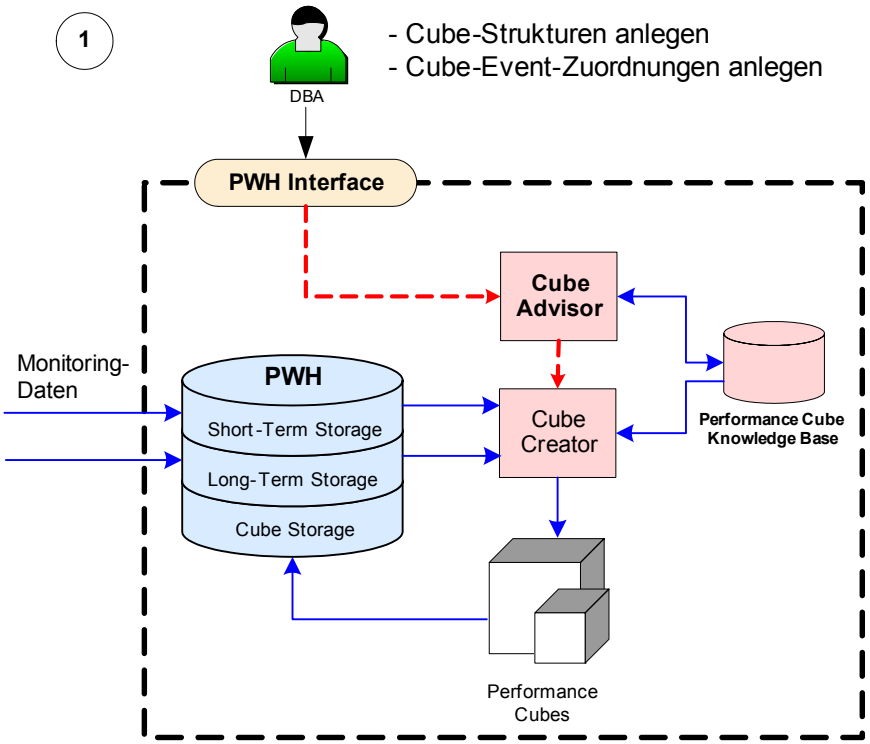
Die Cube-Historie wird verwendet, um ein Ranking der Cube-Kandidaten zu erstellen und ein adaptives Lernen zu ermöglichen. Das Ranking, anhand der Cube Success Ratio (CSR), wird hierbei aus dem Quotienten der Anzahl der Nutzungen eines Cubes (count C) und der Anzahl an erfolgreichen Verwendungen zur Lösung/Analyse eines Problems (successful count SC) gebildet. Die CSR ist demnach die aus der bisherigen Nutzung ermittelte Wahrscheinlichkeit, dass ein Cube-Vorschlag auch in Zukunft zu einer erfolgreichen Problem-Analyse führt und keine weiteren Cubes benötigt werden. Die Rekalkulation der CSR basierend auf den aktuellen Beobachtungen führt zu einem individuellen, adaptiven Cube Advisor.

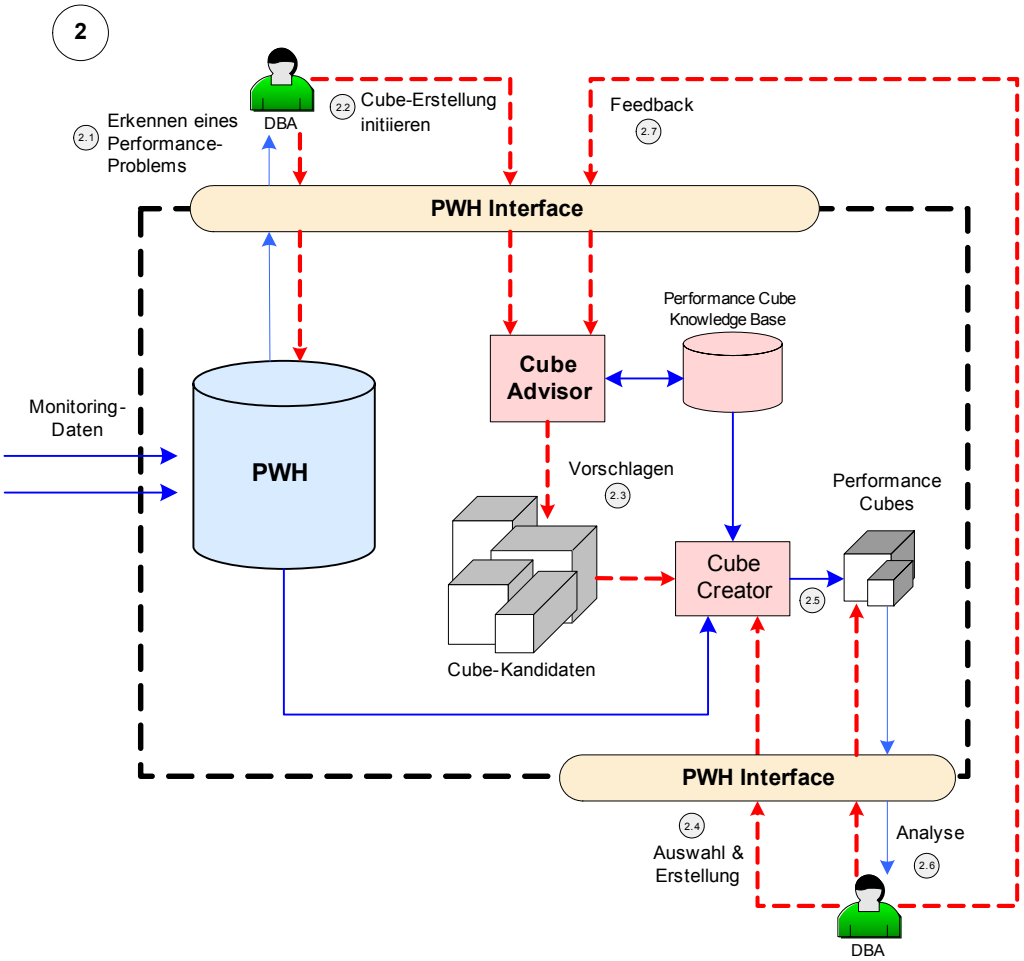
Der Cube Advisor kann als Wegweiser zur Vereinfachung der Auswahl multidimensionaler Cubes für die Problem-Analyse gesehen werden. In der initialen **Lernphase** verwendet der DBA den Advisor interaktiv. Der Cube Advisor befragt den DBA über existierende Performance-Probleme und schlägt eine Menge an geeignet scheinenden Cubes für die interaktive Investigation der Probleme sowie zur Ergründung deren Ursache(n) vor. Der Nutzer kann aus der Liste der Kandidaten dann die (Teil-)Menge der in Folge automatisch angelegten[51] und gefüllten Cubes wählen. Währenddessen wird inkrementell die Cube-Historie zusammen mit den entsprechenden CSR aktualisiert. Sobald die CSR eines Cubes eine nutzerdefinierte Grenze überschreitet, d.h. sobald eine Menge von Performance-Problemen mit dem zur Analyse geeignetsten Cube für die Lösung ermittelt wurde, kann der Advisor eine Erzeugung und anschließende Befüllung automatisch, ohne zusätzliche Nutzer-Interaktionen entweder selbst oder mit Hilfe des Cube Creator vornehmen.

Die Kombination eines gut trainierten/angelernten Cube Advisors mit automatischer **Ereignis-Erkennung** verspricht weiteres Potential. Beispielsweise kann eine beunruhigend niedrige und einen vordefinierten Schwellwert unterschreitende BPHR den Cube Advisor automatisch anstoßen und zur Erzeugung passender Cubes sowie zum Transfer der relevanten Daten aus dem PWH führen. Sobald der/die Cube/s zur Verfügung stehen, können die DBA informiert werden und mit der manuellen bzw. GUI-basierten Analyse beginnen.

---

[51] Der ETL-Vorgang kann im Übrigen auch zu einem späteren, lastarmen Zeitpunkt durch den Cube Creator geplant bzw. ausgeführt werden.


1
DBA
- Cube-Strukturen anlegen
- Cube-Event-Zuordnungen anlegen
PWH Interface
Cube Advisor
Monitoring-Daten
PWH
Short-Term Storage
Long-Term Storage
Cube Storage
Cube Creator
Performance Cube Knowledge Base
Performance Cubes


2
DBA
2.1 Erkennen eines Performance-Problems
2.2 Cube-Erstellung initiieren
Feedback
2.7
PWH Interface
Performance Cube Knowledge Base
Cube Advisor
Monitoring-Daten
PWH
Vorschlagen
2.3
Performance Cubes
Cube Creator
2.5
Cube-Kandidaten
PWH Interface
2.4 Auswahl & Erstellung
Analyse
2.6
DBA

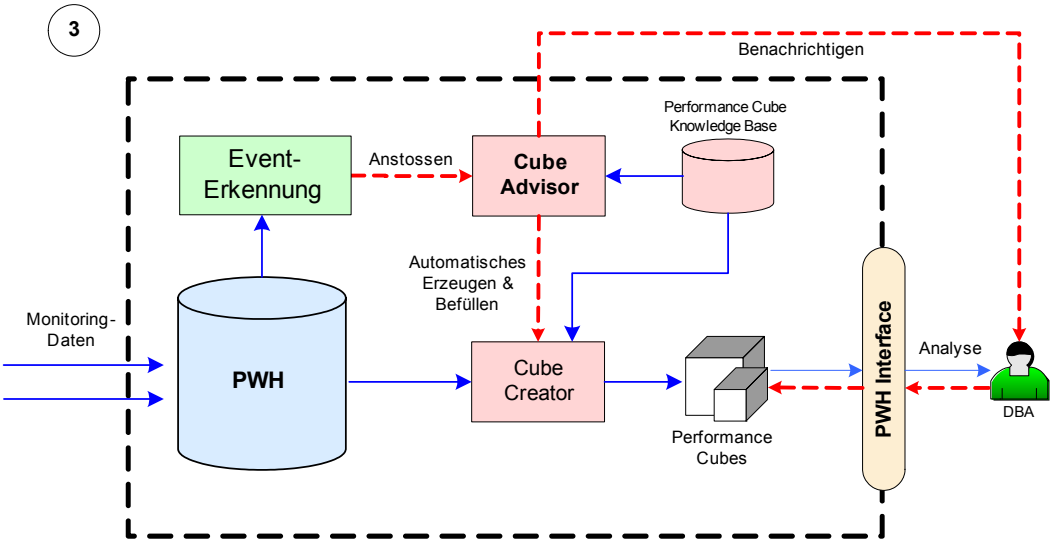


Abbildung 8.11: Details zur Integration und Funktionsweise des Cube Advisor

Darüber hinaus kann der Cube Advisor anhand der Kosten der Befüllung bzw. Benutzung eines Performance Cubes ermitteln, ob es nützlicher wäre, einen Performance-Cube persistent im Cube Storage des PWH abzulegen oder erst bei Bedarf zu füllen. Basierend auf den detektierten Beziehungen zwischen Events und Performance Cubes kann die automatische Befüllung derer veranlasst werden, sobald das in Korrelation stehende Event auftritt. Darüber hinaus lässt sich anhand der Nutzungs-Statistiken der Cubes sowie der Erfolgs-Statistiken der darauf basierenden Tuning-Pläne eine Aussage über den Nutzen und die Effizienz von Performance Cubes treffen. Sämtliches Kontextwissen sowie die dadurch und anhand der Laufzeitdaten generierten Cube-Statisken werden in der separaten Performance Cube Knowledge Base protokolliert.

Abbildung 8.11 umreißt abschließend einerseits die Einbettung des Cube Advisor in die PMI sowie andererseits die ideale Nutzung des Cube Advisor anhand dreier selbsterklärender Teil-Abbildungen:

1. Erzeugung und Initialisierung der Performance Cube Knowledge Base mit generellen Informationen über mögliche Probleme, Cubes und deren Assoziationen
2. Trainieren des Cube Advisor durch interaktive Nutzung des Tools (Lern-Modus)
3. Anwendung des Cube Advisor im autonomen Modus

8.4.3 Anpassung bestehender ATE-Komponenten

Bei der Eingliederung der Analyse-Komponenten sehen wir zwei Möglichkeiten. Zum einen kann der **Event Collector** als Subkomponente des Client Agents verwendet werden. Letzterer empfängt atomare Events der untergeordneten Tools bzw. Datenquellen, übergibt diese zur Transformation in das CBE-Format an den Event Collector, so dass sie von dort schließlich aktiv zur System-übergreifenden Analyse an den **Event Correlator** übertragen werden können. Neben der Filterung und Korrelation ist der Event Correlator auch dafür zuständig sowohl die aufgetretenen atomaren als auch die zusammengesetzten Events im Short Term Storage des PWH zur Historisierung und etwaigen späteren Ursachen-Analyse zu hinterlegen. Zum anderen könnten die Client Agents für sich agierend die empfangenen Events direkt in das PWH ablegen und damit dem Event Collector zur darauf folgenden Transformation und Weiterleitung zur Verfügung zu stellen. Die Transformation in das CBE-Format kann dabei natürlich auch schon vor der Ablage in das

PWH erfolgen, um den Aufwand zur Analyse zu verringern und ein homogenes Format zu etablieren. Beide Varianten haben ihre Vor- und Nachteile. In der ersten Variante können Events zeitnah erkannt und Aktionen eingeleitet werden. Die letzte Variante birgt den Vorteil, dass keine doppelte Weiterleitung erfolgen muss und dadurch der Event Collector bzw. der Event Correlator entlastet wird.

Durch die erweiterten Möglichkeiten der Daten-Sammlung und -Speicherung sollte der Event Correlator auch um die Fähigkeit des Anstoßens von Daten-Sammlungen erweitert werden. Denkbar wäre etwa ein komplexes Event, welches nur dann eintritt, wenn die Datenlage es erforderlich macht. In diesem Fall würde die Event Engine den ersten Teil des komplexen Events erkennen, eine Anforderung an die Monitoring-Komponente senden und nach einer bestimmten Wartezeit die gewonnen Daten unmittelbar auswerten, um festzustellen ob das Event nun einzutreten und eine Reaktion zur Folge hat oder nicht.

Die Einbettung des **Workload Classifier** bedarf nur weniger Anpassungen. Aber auch hier sind verschiedene Herangehensweisen denkbar. Einerseits besitzt der DBA die Option einen ETL1-Job zu definieren, der die für eine Workload-Klassifikation nötigen Daten sammelt und im Short Term Storage des PWH ablegt. Auf diese Daten kann der Workload Classifier in regelmäßigen Abständen zugreifen, um die aktuelle Workload zu ermitteln. Die Workload-Daten können demnach wie reguläre System-Performance-Daten behandelt und entsprechend analysiert werden. Andererseits kann die Workload-Ermittlung auch auf Basis spezifischer, von den Client Agents an den Workload Classifier übermittelten Performance-Daten erfolgen. Da die Erstellung eines ETL1-Jobs deutlich weniger komplex als eine Anpassung der Client Agents und des Klassifikators ist, sollte die erste Variante zur Workload-Bestimmung genutzt werden. Ein weiterer Nutzen ergäbe sich auch, sofern die Workload-Daten im PWH für Zwecke der langfristigen bzw. Trend-Analyse aggregiert werden. Es ist in diesem Zusammenhang vorstellbar, dass sich der DBA einen individuellen Workload-Cube anlegt, der ihm genauere Einsicht über den Workload-Verlauf auf dem überwachten System ermöglicht.

Sofern der Workload Classifier die erkannte Workload wie gehabt in Form eines neuen Events an den Event Correlator weiterleitet und im Sinne der lückenlosen Langzeit-Protokollierung im PWH hinterlegt (bzw. durch den Event Correlator hinterlegen lässt), halten sich die Anpassungen des Event Correlators in Grenzen, da die von ihm benötigten Daten im Short Term Storage des PWH vorliegen.

In der Regel beruht die Analyse-Phase auf der Auswertung kurzfristiger Performance-Daten bzw. häufig wiederkehrender Events. Um die Analyse-Möglichkeiten auszudehnen, kann der Event Correlator bzw. auch eine separate Plan-Komponente (siehe Abschnitt 9.2) um eine Trend-Erkennung und -Vorhersage erweitert werden. Hierzu kann der DBA über entsprechende ETL-Jobs Performance Cubes definieren, auf welche die Analyse- bzw. Planungskomponenten des ATE schließlich zu gegebener Zeit zugreifen und damit im Idealfall proaktiv effektiver reagieren zu können, als dies bei der kurzfristigen Analyse der Fall ist.

Auch die Definition gesonderter Performance Cubes für häufig wiederkehrende Events mittels geeigneter ETL2-Jobs kann zum Zeitpunkt des Problem-Auftretens für zügigere Analyse-Ergebnisse und darauf basierende Entscheidungen sorgen. Um häufig benötigte Performance-Cube-Analysen möglichst schnell ausführen zu können, ist es ratsam, die Performance Cubes persistent im Cube Storage des PWH verfügbar zu halten.

Sollte ein bestimmtes Event erkannt werden, kann es sinnvoll sein, unmittelbar die Befüllung bzw. Aktualisierung eines Performance Cubes oder auch die Sammlung zusätzlicher Daten anzustoßen, um später auf diese umfassenden Informationen zum Zeitpunkt des

Problem-Auftretens analysierend zurückzugreifen. Über den Cube Advisor lassen sich auf diesem Weg Events zu Performance Cubes zuordnen. In den meisten Fällen kündigen sich Probleme jedoch allmählich durch eine Reihe von einzelnen atomaren Events, sprich Indizien, an. Eine Trend- bzw. Muster-Erkennung kann hierbei behilflich sein, derartige sich ins negative abzeichnende Performance-Metrik-Verläufe zu erkennen und ggf. beiläufig Performance Cubes mit Kontext- und Hintergrund-Informationen zu füllen, die später bei dem eigentlichen Auftreten des Problems zum Ermitteln der Ursache zu Rate gezogen werden können.

Der **Tuning Plan Selector** benötigt keine spezielle Anbindung an die Monitoring-Architektur, da die Bestimmung des Tuning-Plans anhand der übermittelten Events vom Event Correlator erfolgen kann. Sein Funktionsumfang wird allerdings aufgrund der Bewertungen durch den Statistics Agent erweitert. Sollten bspw. für ein in Form eines Events erkannten Performance-Problem mehrere Tuning-Pläne in Frage kommen, so können die zusätzlichen Meta-Informationen über die Effizienz in den Tuning-Plänen helfen eine Sortier-Reihenfolge herzustellen und somit eine bessere Auswahl zu treffen. Des Weiteren hat der Tuning Plan Selector Zugriff auf das Statistics-Repository des Statistics Agent sowie auf die Performance Knowledge Base des Cube Advisor, um ggf. an weitere, übergreifende Tuning-Plan- oder Performance-Cube-Statistiken zu gelangen. Eine weitere Möglichkeit die Auswahl eines Tuning-Plans einzuschränken, ist die Festlegung von Policies, wie sie in Abschnitt 4.2.1 beschrieben wurden. Zur Realisierung werden bspw. zur Laufzeit jedem Tuning-Plan bestimmte, Policy-abhängige Prioritäten zugeordnet. Wir sehen eine weitere mögliche Erweiterung in der Zuhilfenahme von Policies sowie der im Ressourcen-Modell enthaltenen Abhängigkeiten zur Verfeinerung der Auswahl der möglichen Cubes bzw. zur Beschränkung der zu betrachtenden Dimensionen.

Um die Monitoring-Infrastruktur in den ATE zu integrieren, müssen die Kommunikations-Möglichkeiten des **Tuning Plan Executors** sowie die Tuning-Schritte um die Möglichkeit des Zugriffs auf die Performance-Daten des PWH erweitert werden. In [Köh10] beschreiben wir ausführlich welche Neuerungen und Änderungen an den bereits in Abschnitt 4.2.3 eingeführten und in Kategorien eingeteilten Tuning-Schritten im Rahmen der Monitoring-Infrastruktur-Integration erforderlich scheinen.

Die Einbettung der Monitoring-Umgebung erlaubt eine generelle Einteilung in drei Klassen, deren angepasste bzw. neue Funktionalitäten durch entsprechende Schnittstellen zur Verfügung gestellt werden müssen:

- **Sammlung:** Ad-hoc-Datensammlungen können durch das Ausführen von ETL1-Jobs mittels des Server Agent angestoßen werden. Die resultierenden, flüchtigen Performance-Daten werden anschließend im Temporary Storage des PWH abgelegt, um von dort dem Tuning-Schritt, der für die Ausführung verantwortlich ist, zur Verfügung gestellt zu werden. Zur Daten-Sammlung lassen sich jedoch auch die regelmäßigen, automatischen Tuning-Schritte zur Extraktion, Transformation und dem letztlichen Laden der PWH-Strukturen, das Befüllen der Performance Cubes und auch das Aggregieren der Performance-Daten zählen.
- **Datenzugriff und -Analyse**: Tuning-Schritte zur Entscheidungsunterstützung sollen den ATE mit Daten versorgen, die er für das Treffen von Entscheidungen benötigt. Diese sollen dank der Monitor-Architektur direkt aus dem PWH abgefragt werden können. Darüber hinaus sind Trend-Analysen und -Vorhersagen durch Anfragen an den Long Term Storage bzw. das Analysieren von im Cube Storage des PWH vorliegenden Performance Cubes denkbar. Neben der Möglichkeit auf die

Strukturen des PWH lesend und in Form von Analysen zuzugreifen, werden auch Tuning-Schritte benötigt, welche den Zugriff auf die Metadaten und die Konfiguration der einzelnen Komponenten des Server Agent sowie des PWH ermöglichen. Dazu gehören zum Beispiel ETL1/2/3-Jobs sowie die Statistiken des Statistics Agent, die im Statistics Storage vorliegen.

- **Konfiguration:** Neben den Daten-auslesenden bedarf es auch -änderenden Tuning-Schritten zum Hinzufügen bzw. Konfigurieren der Komponenten oder auch dem Bearbeiten der Metadaten der Monitor-Architektur. Beispielsweise seien hier das Hinzufügen/Konfigurieren/Löschen von Daten-Quellen oder Agenten, die Definition der Schema Mappings oder auch das Erzeugen bzw. Befüllen von Performance-Cube-Strukturen genannt. Eine solche Aktualisierung eines Performance Cubes ist nötig, wenn für die Analyse durch den Tuning-Schritt aktuelle Performance-Daten vorliegen müssen. Es ist zudem denkbar, die Datensammlungs-Intervalle und weitere Parameter (Sammlungsdauer, Datenquellen) in ETL1-Jobs, die Prioritäten von ETL2-Jobs oder auch Schwellwerte der Performance Cubes zu bearbeiten. Ebenso kann es nötig sein einen ETL1-Job vorübergehend zu deaktivieren. Zu den Konfigurations-Tuning-Schritten zählen überdies Datenbank-Design-Schritte für die Monitor-Architektur mit dem Ziel Zugriffsstrukturen, wie Indexe, Partitionen, Tablespaces, Bufferpools, usw. für das PWH bzw. die Staging Area anzulegen. Das Erstellen, Bearbeiten oder Löschen von materialisierten Sichten, die für die persistente Speicherung von Performance Cubes verantwortlich sind, kann auch dieser Kategorie zugeordnet werden.

Der Tuning Plan Executor erhält über das PWH Interface lesenden Zugriff auf die Performance-Daten des PWH. Hier kann er die Ergebnisse durchgeführter Daten-Sammlungen abrufen, die im Temporary Storage des PWH abgelegt wurden. Ebenso können Daten zur Entscheidungsunterstützung aus dem Short Term Storage, Long Term Storage und in Form von Performance Cubes aus dem Cube Storage abgerufen werden.

Über das PWH Interface und das Server Agent Interface erhält der Tuning Plan Executor zudem die Möglichkeit des ändernden/konfigurierenden Zugriffs auf die Komponenten des PWH (TS, CS, STS, LTS, Summarization Agent, Cube Creator), auf die des Server Agents (Client Agents, Transformation Agent, Loading Agent und Staging Area) sowie auf den Statistics Agent und den Cube Advisor.

Kapitel 9

Zusammenfassung und Ausblick

Die zunehmende Komplexität von IT-Systemen macht deren Administration immer aufwendiger und kostspieliger. Die täglichen Arbeitsroutinen zum Betrieb umfassen die Fehlerbehandlung in Echtzeit, das Abarbeiten von Alarm-Meldungen, das Lokalisieren und Analysieren der Fehler (und Ursachen) sowie das Durchführen von „Reparaturen".

Ansätze, wie das Autonomic Computing helfen der steigenden Komplexität heutiger Systeme entgegenzuwirken, indem sie die komplexen Verwaltungs- und Konfigurations-Aufgaben automatisieren und an die Systeme selbst übertragen. Voraussetzung dafür sind Konzepte und Verfahrensweisen, um zu erkennende Problem-Situationen und darauf basierende -lösungs-Strategien maschinenauswertbar zu beschreiben, den Zustand sowie das Verhalten von Ressourcen permanent zu überwachen und schließlich zur Laufzeit adäquat automatisiert zu reagieren.

Der in dieser Arbeit vorgeschlagene Ansatz setzt sich, in Folge einer obligatorischen Einführung der für das Verständnis notwendigen Grundlagen (Kapitel2) und der Einführung in das Autonomic Database Tuning (Kapitel 3), die schrittweise Erarbeitung einer ganzheitlichen Datenbank-Performance-Monitoring und -Tuning-Architektur zum Ziel. Letztere verknüpft Techniken des Autonomic Computing mit denen des Data Warehousing und der an die medizinische Informatik angelehnten Formalisierung von best-practices-basierten Problemlösungs-Strategien. Elementare Grundlage bilden die Klassifikation und Formalisierung (Kapitel 4), (agenten-basierte) Gewinnung bzw. Generierung (Kapitel 5), Verwaltung (Kapitel 6) und Nutzung (Kapitel 7) von Performance- und Metadaten über den Systemzustand, das Systemverhalten und die Arbeitslast. Durch die Daten-Sammlung- und -Verwaltung-Architektur (Kapitel 8) gewonnene Performance- und Metadaten können sowohl im Rahmen des manuellen als auch des autonomen Datenbank-Performance-Tuning zur automatisierten Analyse, Planung und Entscheidungsfindung Anwendung finden.

9.1 Ergebnisse der Arbeit

Das im Rahmen des universitären Kooperationsprojekts „Autonomic DB2 Performance Tuning" mit der IBM entstandene ATE-Basis-Framework demonstriert prototypisch die automatisierte Analyse, Planung und Entscheidungsfindung im Umfeld des DB2-Datenbank-Tuning. Durch die automatisierte Abarbeitung formalisierter bewährter Standard-Tuning-Praktiken können, so auch exemplarisch nachgewiesen, Problem-Situationen zur Laufzeit erkannt und in Reaktion darauf durch geplantes bzw. koordiniertes Ausführen von formalisiertem, auf Erfahrungen basiertem Tuning-Wissen adressiert werden.

Der in diesem Schriftstück vorgestellte Ansatz zur Arbeitslast-Klassifikation und zur Formalisierung sowie Problem-gesteuerten Ausführung von Tuning-Wissen ist nicht speziali-

siert auf Datenbanksysteme. Er zeichnet sich durch seine Universalität, Modularität und Erweiterbarkeit auf beliebige Hard- und Software-Architekturen aus. Es wird lediglich die Umsetzung von Autonomic-Computing-Konzepten in einer realen, Datenbank-basierten (DB2-)Produktumgebung gezeigt. Die Überwachung und Administration des zu tunenden (Datenbankmanagement-) Systems erfolgt ausschließlich über definierte Schnittstellen. Wissen über System-Interna wird prinzipiell nicht benötigt, kann aber laut Abschnitt 7.5 äußerst förderlich sein.

Diese Arbeit hat gezeigt, welche Schritte und Hürden zum Errichten einer integrativen Performance-Monitoring und -Tuning-Architektur bewältigt werden müssen, bei welcher die Datenbank sowohl als „Gegenstand des (autonomen) Tuning" als auch als Hilfsmittel hierfür angesehen werden kann. Dazu zählt die Entwicklung von Konzepten zur Formalisierung, Modellierung und Verwaltung von Tuning-Aktionen unter Ausführungssteuerung durch Tuning-Richtlinien. Durch die Formalisierung von Tuning-Wissen werden des Weiteren die Integration, der Austausch und die automatische Workload-abhängige Anwendung bewährter Datenbank-Tuning-Praktiken zur Laufzeit ermöglicht. Die formalisierten Problemlösungs-Strategien werden in einer spezialisierten Workflow-Sprache (z.B. XML-basiert) hinterlegt und können zur Laufzeit Ereignis-gesteuert, prinzipiell durch eine beliebige Workflow Engine ausgeführt werden. Nähere Details können der Arbeit von [Rab11] entnommen werden.

Andererseits müssen Mechanismen zum Erfassen, Verwalten und Anwenden von Performance- und Workload-(Meta-)Daten wohl durchdacht sein. Für die Extraktion der Performance-Daten aus den heterogenen Quellen und das anschließende Transformieren und (Be-)Laden, müssen Organisation und Aufbau des Performance Warehouse an den Kontext angepasst werden. System-, Workload- und Performance-Daten werden multidimensional an zentraler Stelle verwaltet und können sämtlichen Prozessen sowie Komponenten zur Laufzeit adäquat bereitgestellt werden. Hierbei lassen sich sowohl zur Speicherung und Verwaltung als auch zur finalen Analyse multidimensionaler Daten am Markt etablierte Warehousing- und OLAP-Tools einsetzen.

Die multidimensionalen, an die Analyse- und Nutzer-Bedürfnisse angepassten Performance-Daten können im Idealfall flexibel und intuitiv analysiert werden, ohne eine neue Sprache zu lernen. Durch entsprechende OLAP-Methodiken zur Organisation und Präsentation der Daten für die Analysen können Experten produktiver arbeiten und Anfänger schnellere Erfolge erzielen. OLAP bietet eine Reihe an Features zur Vereinfachung der Entscheidungsfindung. Durch die Speicherung der Entscheidungs-unterstützenden Daten in Spreadsheet-ähnlichen, multidimensionalen Datenstrukturen, können die DBAs auf ihren umfangreichen (historischen) Datenbestand in einfacher und verständlicher Weise zugreifen: nach den Dimensionen bzw. dem Kontext der operativen Quellen. Dimensionen stehen in engem Zusammenhang mit dem topologischen und architekturellen Aufbau des zu überwachenden Software-Stack und befähigen zu einer sehr intuitiven Art der Organisation und Selektion von Daten zum Zugriff und der Analyse.

Die ausführlichsten Performance-Daten bieten keinen Mehrwert, wenn darauf nicht schnell zugegriffen werden kann. Es muss also ein guter Kompromiss zwischen der Menge an verfügbaren Daten und der Zugriffszeit gefunden werden. Zur langfristigen Speicherung bietet sich eine Verdichtung der Daten zur Reduzierung des Datenvolumens mittels Methodiken der Komprimierung und der Aggregation sowie zum Löschen bzw. Archivieren nicht mehr benötigter Daten an. Die Themen- bzw. Problem-spezifischen Performance Data Marts bieten darüber hinaus eine mögliche Basis für Reports, die voraggregiert für die spätere (Wieder-)Verwendung gespeichert werden. Auf dem Performance Repository aufbauende Data Marts können, sofern entsprechend eingesetzt und nach einer an die

Bedürfnisse und Bedingungen angepassten Aktualisierungs- bzw. Materialisierungs-Strategie verwaltet, zudem der I/O-Entlastung dienen.

Die in dieser Arbeit zusammengetragenen und teilweise anhand des ATE implementierten bzw. evaluierten Konzepte sollen als Vorlage für eine schlussendlich Implementierung einer auf dem Konzept eines multidimensionalen Performance Warehouse basierenden, selbst-optimierenden Monitoring- und Tuning-Architektur dienen und die Grundlage schaffen, weitere konzeptionelle und prototypische Untersuchungen zum autonomen Datenbank-Tuning anzustellen. Dabei wird die finale Umsetzung noch einmal zusätzliche Zeit und Mühe in Anspruch nehmen. Insbesondere da viele der hier angestellten Überlegungen rein theoretischer Natur sind, werden sich im Zuge der praktischen Umsetzung noch weitere, hier nicht erwähnte, Probleme und Lösungen ergeben.

Im nachfolgenden Abschnitt seien einige der sich an die Betrachtungen dieser Arbeit anschließenden, bereits zum jetzigen Zeitpunkt absehbaren Themen aufgegriffen und in Kürze vorgestellt.

9.2 Weiterführende Arbeiten

Die nachfolgenden Unterabschnitte sollen die aus unserer Sicht wesentlichen Ansatzpunkte der Fortführungsthematiken zur Erhöhung des Autonomie-Grades (Abschnitt 9.2.1), zum Policy-basierten Monitoring und Tuning (Abschnitt 9.2.2), zur Verteilung und zum Austausch von Tuning-Wissen (Abschnitt 9.2.3) und schließlich zur Integration von Standards (Abschnitt 9.2.4) kurz aufführen.

9.2.1 Intelligente Planung

Der in Abschnitt 8.3 vorgestellte Prototyp soll dahingehend erweitert werden, dass er basierend auf vergangenen, aktuellen und prognostizierten Informationen intelligente Entscheidungen unter Minimierung menschlicher Eingriffe selbsttätig treffen und sich daran orientierend handeln kann. Mit einer intelligenteren Plankomponente können, im Gegensatz zu den von uns exemplarisch vorgestellten, weitaus tiefgreifendere Tuning-Pläne umgesetzt sowie zur Laufzeit zunächst priorisiert und schließlich ausgeführt werden.

Zur Gewährleistung der gewünschten Adaptivität, muss sich ein Autonomic Tuner seiner selbst sowie seiner Aktionen in der Umwelt bewusst sein. Er sollte über eigene Sensor- und Effektor-Schnittstellen verfügen, die er idealerweise selbst kontrolliert und dem Kontext sowie den wechselnden Umgebungsbedingungen zur Laufzeit anpasst.

Gerade mit einer Zunahme der Adaptivität und der Autonomie durch intelligente Mechanismen muss sichergestellt werden, dass das System gegen einen stabilen Endzustand konvergiert und nicht oszillierend zwischen vermeintlichen optimalen Zuständen springt. In diesem Zusammenhang sollten auch Überlegungen in Richtung von robusten Tuning-Plänen angestellt werden. Solche Pläne, die selbst im Falle von Unsicherheiten (z.B. Kardinalitäts-Abschätzungen) eine vorhersagbare Performance garantieren, sind aus anderer Sicht optimalen Plänen zu bevorzugen.

Prinzipien zur Adaptivität können nicht allein auf das Monitoring beschränkt, sondern auch auf das Tuning angewendet werden. Wir sehen hierbei Optimierungsbedarf bei der Filterung, Priorisierung und kontextbasierten Einschränkung von Tuning-Plänen. Abhängig von der Historie, der aktuellen Umgebung und Last auf dem System kann die Menge der anwendbaren Events und Tuning-Pläne zur Laufzeit adaptiv angepasst werden. Auch die Übertragung des aus DB2 bekannten Utility Throttling (siehe Abschnitt 3.1.3) auf die Ausführung von Tuning-Plänen ist denkbar. Je nach Kontext und operativer bzw. Tuning-

Aktivität auf dem System kann der Beginn der Problemlösung verzögert oder die Ausführung Ressourcen-schonend hinten angestellt werden.

Ein nahe liegender Ansatz zur Erhöhung des Autonomiegrades liegt in der Untersuchung typischer Vorgehensweisen von DBAs zur Problem-Erkennung und -Auflösung. Das System könnte das „Verhalten" der DBAs automatisiert aufzeichnen, über einen Lernprozess verinnerlichen und in Folge selbst analog agieren. Daneben können in einer Trainingsphase auch Events erlernt werden, so dass in Folge kritische Schranken selbstständig erkannt und angepasst werden können.

Autonomes Tuning erfordert Wissen über die Ressourcen(-hierarchien) und ihre Abhängigkeiten. Als eine der möglichen zukünftigen Anwendungen dieses Wissens sehen wir die automatische Umverteilung der Ressourcen-Allokation. Ähnlich den autonomen Features zur automatischen Speicherverwaltung in DB2 v9 wollen wir dem System von außen Schranken (z.B. den Gesamtspeicher) vorgeben und es innerhalb dieser Schranken autonom, Workload-adäquat agieren lassen. Dabei sollen alle Ober- und/oder Untergrenzen und der eigentliche Ressourcenverbrauch für alle untereinander abhängige bzw. in einer Hierarchie angeordnete Speicherkonsumenten automatisch angepasst (umverteilt) werden. Leitgedanken sind hierbei zum einen das automatische Erkennen der Notwendigkeit einer internen Ressourcen-Umverteilung (z.B. durch eine Workload-Analyse bzw. auch -Vorhersage oder durch „getriggerte", Problem-anzeigende Events). Zum anderen sollen vorgegebene Policies bei der Ressourcen-Aufteilung Berücksichtigung finden. In ihnen können bindende Untergrenzen, oder auch tatsächliche Ressourcenverbräuche für kritische, dedizierte Ressourcen fest vorgegeben oder im besten Fall automatische Re-Allokations-Aktionen abgeleitet werden. Besonderes Augenmerk ist auf die Beachtung der Ressourcen-Hierarchie-Dynamik, wie bspw. dem Einfügen bzw. Entfernen von Ressourcen, zu legen.

Von weiterem Interesse bleiben auch alternative zu den im Abschnitt 7.4.7.2 umrissenen Möglichkeiten zum Lernen und der Vorhersage der künftig zu erwartenden Workload bzw. des Systemverhaltens. Da es sich bei konkreten SQL-Statements nicht um numerische Daten handelt, kann eine Regressions-Analyse nicht auf die Rohdaten angewendet werden. Stattdessen müssen diese Daten entweder zuvor entsprechend transformiert oder andere Verfahren hierfür ausgewählt werden. Insbesondere die Korrelation des Wissens um die prognostizierten Anfragen mit den von ihnen adressierten Ressourcen erscheint uns im Sinne einer intelligenten Plankomponente und der Bestimmung des erwarteten Systemzustandes nutzbringend. Weitergehende Arbeiten haben demzufolge zu untersuchen, wie unsere prototypische Architektur um eine derartige Vorhersage effektiv erweitert und zur Laufzeit genutzt werden kann. Die Erweiterung des ATE um zunächst einen *System State Evaluator* zur Ermittlung bzw. Vorhersage der Entwicklung des aktuellen bzw. absehbaren Systemzustands in Relation zu einem definierten Sollzustand ist ratsam. Zudem kann ein *Trend Analyzer*, als separate oder darin integrierte Komponente, proaktive Trend-Exceptions generieren, die in Folge weitere Monitoring-Maßnahmen oder auch ein „leichtgewichtiges", störfreies Tuning auslösen, um dem Problem mit wenigen, einfachen, leichtgängigen Mitteln im Voraus entgegenzuwirken.

9.2.2 Policies

Wir sehen ebenfalls viel Potential in der erweiterten Policy-Steuerung auf Tuning-Komponenten- bzw. auch Ressourcen-Ebene. Voraussetzung für ein derartiges Policy-basiertes Monitoring und Tuning ist neben der Erweiterung der einzelnen Komponenten auch die integrierte, übergreifende und systematische Handhabung der Policies in ihrer Gesamtheit. So wäre es bspw. denkbar, ausgehend von einer abstrakten, übergreifenden

High-Level-Policy automatisch Low-Level-(Komponenten-)Policies abzuleiten und die abhängigen Komponenten bzw. Prozesse damit zu „lenken".

Zur Realisierung einer solchen übergreifenden Policy-Behandlung sind primär Identifikations-, Planungs- und Entwurfs-Arbeiten zu leisten. Darunter fallen bspw. die Sammlung verschiedener Zielvorstellungen und Richtlinien, der Entwurf einer Policy-Hierarchie mit entsprechenden Ableitungsregeln sowie Prinzipien mit denen die wechselseitige Verträglichkeit von Policies geprüft werden kann. In [Rei08] finden sich erste Ansätze einer um eine Policy-Steuerung erweiterte, autonome Tuning-Infrastruktur. Zentrales Element ist hierbei der Tuning Coordinator. Diese Komponente könnte in Zukunft derart erweitert werden, dass sie gemessen an den Policy-Vorgaben den Grad der Zielerreichung protokolliert, in einer Historie verfolgt und bei einer späteren Entscheidung zur Tuning-Plan-Auswahl berücksichtigt. Eine speziellere Form der Erweiterung wäre die Möglichkeit, die maximalen Ausführungskosten zur Durchsetzung bestimmter Policies anzugeben. Basierend auf einem zu entwickelnden Kosten- und Nutzenmodell kann somit auch die Selektion von Tuning-Maßnahmen gesteuert werden. In Verbindung mit dem Ressourcen-Modell können Tuning-Pläne Policy-basiert, dynamisch generiert bzw. mittels der Tuning-Aktion-Historie intelligent zusammengesetzt werden.

Der (Soll-)Systemzustand definiert sich im Grunde durch die Menge der in den Policies definierten Metriken. Es ist demzufolge zu untersuchen, ob und inwieweit man von einer Policy ausgehend nicht nur die anwendbaren Tuning-Aktionen sondern auch die zu überwachenden Performance-Metriken sowie die unmittelbar darauf basierenden (Policy-verletzenden) Events automatisch ableiten kann.

Neben der bereits erwähnten Beeinflussung des Monitoring und des Tuning können Policies auch zur automatischen Bestimmung möglicher, den Richtlinien entsprechender Performance Cubes bzw. zur Beschränkung der zu betrachtenden Dimensionen Einsatz finden.

9.2.3 Tuning-Plan-Community

Die Formalisierung und Materialisierung des Performance-Monitoring- und -Tuning-Wissens ermöglicht nicht nur eine automatische Problem-Erkennung sowie -auflösung sondern auch den Austausch dieses Fachwissens innerhalb einer globalen DBA-Community. Das Tuning-spezifische Fachwissen kann auf einer zentralen Community-Plattform abgelegt, über eine intelligente Suche durch sämtliche (registrierte) Nutzer aufgefunden, heruntergeladen und in das jeweils lokale Produktivsystem „installiert" werden. Auf diese Weise kann Wissen unkompliziert und schnell von erfahrenen an unerfahrene Administratoren weitergegeben werden.

Zu dem austauschbaren Wissen können bspw. vordefinierte Problem Determination Performance Cubes, Mappings bekannter Datensammel-Tools auf ein spezifisches PWH-Schema, atomare bzw. komplexe Events oder auch ganze Tuning-Pläne gezählt werden. Das Bilden von Übertragungsgranulaten „zusammengehöriger" Objekte auf Basis von Metadaten über Abhängigkeiten zwischen Ressourcen, Events und Tuning-Plänen wurde bereits in Abschnitt 4.4.2 verdeutlicht.

Problem-Erkennungs- und -Lösungs-Strategien können von Dritten entweder einfach automatisch ausgeführt oder mit Änderungen versehen und an die eigenen Bedürfnisse angepasst werden. Evolutionäre Änderungen an Struktur und/oder Inhalt von Tuning- bzw. Monitoring-Vorgehensweisen können durch Techniken der Varianten- und Hierarchiebildung sowie Versionierung protokolliert, nachvollzogen und bei Bedarf rückgängig gemacht werden.

Mit der Einführung einer potentiell stetig anwachsenden Community steigt der Bedarf intelligenter Differenzierungs- bzw. Suchmechanismen. Tuning-Pläne können über geeignete Kategorien, Hierarchien oder auch Verschlagwortung (Tagging) sinnvoll geordnet und indexiert werden. Auch Techniken zur Bewertung von Community-Inhalten können neben den traditionellen Beschreibungs-Metadaten bei der letzlichen Auswahl geeigneter Objekte behilflich sein.

In [Kar08] sowie [Hof09] sind die ersten vielversprechenden Konzipierungen eines Web-basierten Assistenten für die Verwaltung, Anwendung und den Austausch von Tuning-Wissen vorgenommen und evaluiert wurden. Über einen eigenen Interpreter kann die Tuning-Infrastruktur eine Modellierung und Abarbeitung von Workflow-basierten Tuning-Plänen unterstützen. Der Kunde, Dienstleister oder Drittanbieter hat über eine entsprechende GUI die Möglichkeit intuitiv Problemlösungs-Strategien zu erstellen und in dem Community Repository zur Verfügung zu stellen. Die Arbeit von [Krü09] behandelt in Anlehnung verwandter Forschungen den Entwurf sowie die Implementierung eines Versionskontroll- und Verwaltungssystems für Tuning-Ablaufpläne. Für diesen Zweck werden bestehende Tuning-Pläne in ihrer Struktur analysiert, ein geeignetes Datenmodell zur Modell-zentrischen Abbildung erstellt, Prinzipien der Versionierung für das Tuning-Framework zur Verfügung gestellt und beispielhafte Vorgehensweisen zur Speicherung und Gewinnung von Tuning-Plänen aus dem Model vorgestellt. In [Rab11] sollen diese Untersuchungen fachlich vertieft und technisch anhand eines gesamtheitlichen Prototyps realisiert werden.

9.2.4 Schnittstellen und Datenformate

In Abschnitt 2.2.5 wurde bereits in Kürze auf derzeitige und künftig benötigte Standards in heterogenen, komplexen System-Umgebungen eingegangen. Vor allem Bestrebungen speziell für das Autonomic Computing rücken hierbei in den Vordergrund. Die Aufgabe der Wissensverwaltung ist es, die Komplexität des System- und Expertenwissens auf eine regelhafte Struktur abzubilden und autonomen Systemkomponenten sowie dem Nutzer in einem intelligenten Informationssystem bedarfsgerecht zu präsentieren.

An die Untersuchungen anzuschließen sind aus unserer Sicht Betrachtungen zur Modellierung, Beschreibung und zum Austausch von Metadaten in Data-Warehouse-Systemen. Ziel ist es, Interoperabilität zwischen verschiedenen System-Komponenten und -Werkzeugen eines Data Warehouse zu ermöglichen. Adäquate Beschreibungen bzw. automatische Bestimmungen der im ETL-Prozess verwendeten Datenschemata von Quell- und Zielsystemen fördern im Idealfall die zwischen diesen stattfindende automatische Transformation sowie Übertragung von Daten.

Erstellte Metadaten-Beschreibungen können bspw. mittels XML einfachst ausgetauscht oder über einen Zwischenschritt verschiedenen Komponenten oder Programmiersprachen zugänglich gemacht werden.

Gerade beim Einsatz mehrerer autonomer Komponenten, die miteinander kommunizieren und eigene Wissensbasen haben, ist ein gemeinheitliches, übergreifendes Wissensmodell nötig. Ontologien können helfen, diesen Weg einzuschlagen und nicht explizit formuliertes, „verstecktes“ Wissen durch Ableitung und einer Menge von Regeln aufzudecken bzw. darzustellen. Es ist daher im Rahmen weiterer Untersuchungen neben der Analyse aktueller Autonomic-Tuning-Initiativen zu prüfen, ob die Architektur unseres Prototyps auf eine Ontologie-basierte Wissenskomponente umgestellt werden sollte.

9.3 Ausblick

Forschungen auf dem Gebiet des Autonomic Computing und die Adaption auf das autonome Datenbank-Tuning sind sehr viel versprechend.

Die Überführung heutiger DBMS-Architekturen in vollständige autonome Systeme ist ein langwieriger Prozess. Von adaptiven bzw. autonomen Systemen kann man zu diesem Zeitpunkt noch nicht sprechen. Große Fortschritte in Forschung und Entwicklung wurden bereits durch Tools und Techniken zur automatischen Speicherverwaltung, zum physischen Datenbankentwurf sowie zur Verwaltung von Statistiken erzielt. Insbesondere die drei Marktführer im Bereich Datenbanken decken primär die Bereiche der Selbst-Optimierung und Selbst-Konfiguration ab und haben erste große Schritte zur Selbst-Verwaltung ihrer Produkte getan. Grundsätzlich sind die derzeitigen Tools aber noch sehr auf den manuellen Eingriff angewiesen.

Aktuelle Entwicklungen zeigen, dass man noch weit davon entfernt ist, den Menschen abzuschaffen. Aber nach dem heutigen Stand der Technik und der Komplexität, die Administratoren gegenüberstehen, bringt bereits eine partielle, semi-automatische Lösung einen nicht unerheblichen Mehrwert.

Generell ist festzustellen, dass der überwiegende Anteil der derzeitigen Produkte im Bereich Autonomic Computing primär auf die Realisierung einzelner Teil-Aspekte des Paradigmas abzielt. Künftige Bestrebungen zur Weiter-Entwicklung von Autonomic-Computing-Systemen sollten dahingehend unternommen werden, die Interoperabilität unterschiedlicher, bereits fortgeschrittener Einzel-Systeme durch eine effiziente Kommunikation untereinander zu gewährleisten und damit zu einem umfassenden Ganzen zu verknüpfen. Die Vereinheitlichung proprietärer Daten- und Übertragungs-Formate kann aus unserer Sicht maßgeblich durch den Einsatz und die Förderung offener Standards (wie z.B. WSDM), gemeinsamer Schnittstellen sowie eine Fokussierung auf den Open-Source-Gedanken erzielt werden.

Zu den wesentlichen technischen Herausforderungen bei der Entwicklung und dem Einsatz autonomer Systeme, wie sie in Paul Horns Autonomic Computing Manifesto beschrieben werden, zählen besonders jene Komponenten, deren Funktionen auf dem Treffen dynamischer Entscheidungen beruhen. Hier sehen wir die Forschungsfelder der Muster-Erkennung und der künstlichen Intelligenz in der maßgeblichen Verantwortung zur Bereitstellung von Strukturen, Systemen und Verfahrensweisen zum Umgang mit der Komplexität in derzeitigen und sich entwickelnden heterogenen IT-Landschaften.

Abgesehen vom noch jungen Stand der Technik werden erfahrene und gut ausgebildete Spezialisten unserer Meinung nach niemals komplett durch intelligente, autonome Tools ersetzt werden können. Eine vollständige Automatisierung erscheint nicht realistisch. Vielmehr verschieben sich die Aufgabenschwerpunkte weg von reaktiven, zeitkritischen Routinetätigkeiten hin zu langfristigen, strategischen Aufgaben, wie der bewussten Planung und Festlegung von Richtlinien, die nicht alleine aus dem aktuellen Systemverhalten geschlussfolgert werden können.

Obgleich das Autonomic Computing beispielsweise noch in den Kinderschuhen steckt, erzielen mit heutigem Wissen und unter zahlreichen Einschränkungen entwickelte Systeme bereits beachtliche Ergebnisse. Fakt ist, dass das autonome Datenbank-Tuning in Zukunft immer mehr an Bedeutung gewinnen wird, da die wenigen Fachkräfte nicht in der Lage sein werden, die immer größer werdenden Ansprüche eines Unternehmens an seine Datenbank(en) alleine zu bewältigen. Zukünftige Entwicklungen können also mit Spannung erwartet werden.

Literaturverzeichnis

[AaHe02] W.v.d. Aalst, K. Hee: Workflow Management - Models, Methods, and Systems. The MIT Press, Cambridge, Massachusetts, 2002.

[ABZ07] M. Agostino, G. Baklarz, P. Zikopoulos: DB2 9 for Linux, Unix, and Windows: DBA Guide, Reference, and Exam Prep. IBM Press, 2007.

[ACK+04] S. Agrawal, S. Chaudhuri, L. Kollar, A. Marathe, V. Narasayya, M. Syamala: Database tuning advisor for Microsoft SQL Server 2005. In Proceedings of the 30th International Conference on Very Large Databases, Toronto, Canada, 2004.

[ACN00] S. Agrawal, S. Chaudhuri, V. Narasayya: Automated selection of materialized views and indexes for SQL databases. In Proceedings of the 26th International Conference on Very Large Databases, Cairo, Ägypten, September 2000.

[Ahu06] R. Ahuja: Introducing DB2 9, Part 4: Autonomic and other enhancements in DB2 9. IBM Developerworks Article, Juni 2006. http://www.ibm.com/-developerworks/data/library/techarticle/dm-0606ahuja2/index.html

[Alg10] E. Algaa: Formale Beschreibung und Verwaltung von Ressourcen im Autonomen Datenbank-Performance-Tuning. Studienarbeit, Institut für Informatik, Friedrich-Schiller-Universität Jena, Februar 2010.

[Alt02] W. Alt: Nichtlineare Optimierung - Eine Einführung in Theorie, Verfahren und Anwendung. Vieweg, Braunschweig, 2002.

[AlNi00] P. Alpar, J. Niederreichholz: Data Mining im praktischen Einsatz: Verfahren und Anwendungsfälle für Marketing, Vertrieb, Controlling und Kunden-unterstützung, Vieweg Verlag, Wiesbaden 2000.

[AnMu97] S. Anahory, D. Murray: Data Warehouse, Addison-Wesley, 1997.

[Arb95] Arbor Software: Relational OLAP - Expectations & Reality. White Paper, http://www.arborsoft.com/papers/, 1995.

[Aut07] L. Auth: Evaluierung autonomer Mechanismen in Oracle 10g. Diplomarbeit, Institut für Informatik, Friedrich-Schiller-Universität Jena, Mai 2007.

[BaGu01] A. Bauer, H. Günzel: Data Warehouse Systeme - Architektur, Entwicklung, Anwendung. Dpunkt, Heidelberg 2000.

[Ban04] U. Bankhofer: Data Mining und seine betriebswirtschaftliche Relevanz. In: Betriebswirtschaftliche Forschung und Praxis (BFuP), Jg. 56, Heft 4, S. 395 - 412, 2004.

[BCC+07] B. Baryshnikov, C. Clinciu, C. Cunningham, L. Giakoumakis, S. Oks, S. Stefani: Managing query compilation memory consumption to improve dbms throughput. In 3rd Biennial Conference on Innovative Data Systems Research (CIDR), Seiten 275–283, 2007.

[BCL96] K. Brown, M. Carey, M. Livny: Goal-Oriented Buffer Management Revisited. Proc. ACM SIGMOD conf., 1996.

[BeHa03] M. Berthold, D. Hand: Intelligent Data Analysis, An Introduction. 2nd editon. Springer, 2003.

[Ben03] D. Benoit: Automatic Diagnosis of Performance Problems in Database Management Systems, Dissertation, Juni 2003.

[Ben05] D. Benoit. Automatic Diagnosis of Performance Problems in Database Management Systems. In Proceedings of the Second International Conference on Autonomic Computing (ICAC 05), Seattle, 2005.

[BFGG04] J. Brooke, D. Fellows, K. Garwood, C. Goble: Semantic Matching of Grid Resource Descriptions. UoM. 2nd European Across-Grids Conference (AxGrids 2004), January 2004, Cyprus, http://www.grid-interoperability.org/semres.pdf

[BHS00] J.P. Bigus, J.L. Hellerstein, M.S. Squillante: Auto Tune: A Generic Agent for Automated Performance Tuning. In Proceedings of the International Conference on Practical Application of Intelligent Agents and Multi-Agents, 2000.

[BiGa05] A. Biazet, K. Gajda: Achieving complex event processing with active correlation technology (Online unter: http://www-128.ibm.com/developerworks/autonomic/library/ac-acact/), 2005.

[BMC+94] K.P. Brown, M. Mehta, M.J. Carey, et al.: Towards Automated Performance Tuning for Complex Workloads. In Proceedings of VLDB, 1994.

[BMM+03] D. F. Bantz, S. Mastrianni, C. Mohindra, D. Shea, J. P. Vanover: Autonomic personal computing. IBM SYSTEMS JOURNAL, 42:165–176, 2003.

[BoEn00] M. Boehnlein, A. Ulbrich-vom Ende: Grundlagen des Data Warehousing - Modellierung und Architektur. Otto-Friedrich-Univ., Bamberg 2000.

[Bol96] T. Bollinger: Assiziationsregeln - Analyse eines Data Mining Verfahrens. Informatik Spektrum, 19(5):257–261, 1996.

[Böt07] S. Böttcher: Web Services-oriented Autonomic Management. Studienarbeit. Institut für Informatik, Friedrich-Schiller-Universität Jena, Dezember 2007.

[Böt08] S. Böttcher: Evaluierung und prototypische Umsetzung von Techniken des Data Mining zum Lernen und zur Vorhersage im autonomen Datenbank-Performance-Tuning. Diplomarbeit. Institut für Informatik, Friedrich-Schiller-Universität Jena, August 2008.

[BrCh06] N. Bruno, S. Chaudhuri: To Tune or not to Tune? A Lightweight Physical Design Alerter. In Proceedings of VLDB, 2006.

[Bre94] E. Brewer: Portable high-performance supercomputing: high-level platform-dependent optimization. Ph.D. thesis, Massachusetts Institute of Technology, September 1994.

[Bre95] E. Brewer: High-level optimization via automated statistical modeling. In Proceedings of the Fifth Symposium on Principles and Practice of Parallel Programming (PPoPP '95), Santa Barbara, CA, July 1995.

[Bro95] K. Brown: Goal-oriented memory allocation in database management systems. Ph.D. thesis, University of Wisconsin, Madison, 1995 (available as technical report CS-TR-1995-1288).

[BCL96] K. Brown, M. Carey, M. Livny: Goal-oriented buffer management revisited. In Proceedings of the 1996 ACM SIGMOD International Conference on Management of Data, Madison, WI, June 1996.

[Bro08] B. Broll: Modeling, prediction and synthetic generation of database workloads. Diplomarbeit, Januar 2008.

[BrSe93] I. Bronstein, K. Semendjajew: Taschenbuch der Mathematik 1993. Verlag Harri Deutsch, Frankfurt, 1993.

[CBM+06] W.J. Chen, U. Baumbach, M. Miskimen, et al.: DB2 Performance Expert for Multiplatforms V2.2, März 2006.

[CCG+99] S. Chaudhuri, E. Christensen, G. Graefe, V. Narasayya, M. Zwilling: Self-Tuning Technology in Microsoft SQL Server. Data Engineering Journal 22(2):20-26, June 1999.

[CCK+00] P. Chapman, J. Clinton, R. Kerber, T. Khabaza, T. Reinartz, C. Shearer, R. Wirth: CRISP-DM 1.0: Step-by-Step Data Mining Guide. CRISP-DM consortium: NCR Systems Engineering Copenhagen (USA and Denmark) DaimlerChrysler AG (Germany), SPSS Inc. (USA) and OHRA Verzekeringen en Bank Groep B.V (The Netherlands), 2000.

[CFF+04] K. Czajkowski, D.F Ferguson, I. Foster, J. Frey, S. Graham, I. Sedukhin, D. Snelling, S. Tuecke, W. Vambenepe: The WS-Resource Framework. http://www.globus.org/wsrf/specs/ws-wsrf.pdf, Mai 2004.

[ChNa97] S. Chaudhuri, V. Narasayya: An efficient cost-driven index selection tool for Microsoft SQL Server. In Proceedings of the 23rd International Conference on Very Large Databases (VLDB '97), Athens, Greece, August 1997.

[ChBu97] P. Chamoni, C. Budde: Methoden und Verfahren des Data Mining. Diskussionsbeiträge des Fachbereichs Wirtschaftswissenschaft der Gerhard-Mercator-Universität Gesamthochschule Duisburg, Nr. 232. Duisburg 1997.

[ChDa97] S. Chaudhuri, U. Dayal: An Overview of Data Warehousing and OLAP Technology, SIGMOD Record, 26(1): 65-74, 1997.

[ChGl98] P. Chamoni, P. Gluchowski: On-Line Analytical Processing (OLAP), In:Muksch H., Behme W. (Hrsg.): Das Data Warehouse-Konzept - Architektur, Datenmodelle, Anwendungen, 3. Auflage, Gabler, Wiesbaden, 1998.

[CHS+98] P. Cabena, P. Hadjinian, R. Stadler, J. Verheers, A. Zanasi: Discovering Data Mining: From Concept to Implementation. Prentice Hall, Upper Saddle River, NJ, 1998.

[ChSt98a] P. Chamoni, S. Stock. Temporale Daten in Management Support Systemen, Wirtschaftsinformatik 40(1998)6, S.513-519, 1998.

[ChSt98b] P. Chamoni, S. Stock. Modellierung temporaler multidimensionaler Daten in Analytischen Informationssystemen, In: Proceedings GI-Workshop „Data Mining und Data Warehousing", Technischer Bericht Universität Magdeburg Preprint Nr. 14, 1998.

[CHY96] M. Chen, J. Han, P. Yu: Data mining: An overview from a database perspective. IEEE Transactions on Knowledge and Data Engineering, Dezember 1996.

[CKAK94] S. Chakravarthy, V. Krishnaprasad, E. Anwar, S.-K. Kim: Composite events for active databases: Semantics, contexts and detection. In Proceedings of the 20th International Conference on Very Large Data Bases, Seiten 606 - 617, 1994.

[CKPS95] S. Chaudhuri, R. Krishnamurthy, S. Potamianos, K. Shim: Optimizing Queries with Materialized Views, in: Proceedings of the 11th International Conference on Data Engineering (ICDE 1995, Taipei, Taiwan, 6.-10. März), 1995, S. 190-200.

[Cla98] N. Clausen: OLAP. Multidimensionale Datenbanken. Produkte, Markt, Funktionsweise und Implementierung, Addison-Wesley, 1998.

[Cli06] K. Cline et al.: Toward Converging Web Services Standards for Resources, Events, and Management. Ein gemeinsames White Paper von HP, Intel, IBM und Microsoft. März 2006.

[CCS93] E. Codd, S. Codd, C. Salley: Providing OLAP (On-line Analytical Processing) to User-Analysts: An IT Mandate, E.F. Codd & Associates, White Paper, 1993.

[CMD62] F. J. Corbato, M. Merwin-Daggett, R. C. Daley. An experimental time-sharing system. In Proceedings of the AFIPS Fall Joint Computer Conference, 1962.

[Col96] G. Colliat: OLAP, Relational, and Multidimensional Database Systems, SIGMOD Record, 25(3): 64-69, 1996.

[CSS93] E.F. Codd, S.B. Codd und C.T. Salley Providing OLAP to User-Analysts: An IT Mandate, White Paper, http://www.arborsoft.com/papers/, 1993.

[DaDi06] B. Dageville, K. Dias: Oracle's Self-Tuning Architecture and Solutions. In Bulletin of the Technical Committee on Data Engineering. Special Issue on Self-Managing Database Systems, Band 29. IEEE Computer Society, September 2006.

[Dem99] S. Demmel: Design und prototypische Implementierung eines integrierten Dienstmanagements für Nomadische Systeme in Intranets, Institut für Informatik der Ludwig Maximilian Universität München, Diplomarbeit, November 1999.

[DGG95] K.R. Dittrich, S. Gatziu, A: Geppert. The active database management system manifesto. In Rules in Database Systems, Seiten 3 -17, 1995.

[DGI+03] B. Darmawan, G. Groenewald, A. Irving, S. Henrique Soares Monteiro, K. M. Snedeker: Database Performance Tuning on AIX, 2003.

[DHK+05] Y. Diao, J. L. Hellerstein, G. Kaiser, S. Parekh, D. Phung: Self-Managing Systems: A Control Theory Foundation. IEEE Second conference on Engineering of Autonomic Systems, 2005.

[DiGa00] K. Dittrich, S. Gatziu: Aktive Datenbanksysteme: Konzepte und Mechanismen. Dpunkt-verlag, 2000.

[DMTF99] Distributed Management Task Force: Common Information Model (CIM) Specification, Version 2.2. Distributed Management Task Force, 1999, http://www.dmtf.org/standards/documents/CIM/DSP0004.pdf

[Dor06] S. Dorendorf. Reorganisation von Datenbanken: Auslöser, Verfahren, Nutzenermittlung. Dissertation. Fakultät für Mathematik und Informatik, Friedrich-Schiller-Universität Jena, Oktober 2006.

[DoRa00] H.H. Do, E. Rahm: On Metadata Interoperability in Data Warehouses. Technical Report 01-2000, Inst. für Informatik, Univ. of Leipzig, März 2000.

[DSHB98] B. Dinter, C. Sapia, G. Höfling, M. Blaschka: The OLAP Market: State of the Art and Research Issues, in: Proceedings of the ACM First International Workshop on Data Warehousing and OLAP (DOLAP 1998, Washington, D.C., USA, 7. November), 1998.

[Eat04] C. Eaton: Monitoring your database with just SQL scripts. IBM DB2 Information Management - Technical Conference. Las Vegas, Nevada. September 2004.

[Eat07a] C. Eaton: DB2PD - Great Diagnostoc Tool. http://blogs.ittoolbox.com/-database/technology/archives/db2pd-greatdiagnostic-tool-14089, Januar 2007.

[Eat07b] C. Eaton: DB2PD to monitor buffer pools. http://blogs.ittoolbox.com/-database/technology/archives/db2pd-to-monitorbufferpools- 14219, Januar 2007.

[Eat07c] C. Eaton: DB2PD to monitor buffer pools (part 2). http://blogs.ittoolbox.com/database/technology/archives/db2pd-to-monitorbufferpools-part-2-15464, April 2007.

[Eat07d] C. Eaton: DB2PD to monitor locks. http://blogs.ittoolbox.com/database/technology/archives/db2pd-to-monitorlocks-15228, März 2007.

[Eat07e] C. Eaton: DB2PD to monitor log utilization. http://blogs.ittoolbox.com/database/technology/archives/db2pd-to-monitorlog-utilization-14318, Februar 2007.

[Eat07f] C. Eaton: DB2PD to monitor table access. http://blogs.ittoolbox.com/database/technology/archives/db2pd-to-monitortable-access-14967, März 2007.

[Ech90] K. Echtle: Fehlertoleranzverfahren. Springer-Verlag, 1990.

[EdEd97] H. A. Edelstein, H. C. Edelstein: Building, Using, and Managing the Data Warehouse, Warehouse (Data Warehousing Institute Series from Prentice Hall Ptr, 1997.

[Ehl07] M. Ehlert: Beschreibung und Verwaltung von Tuningstrategien für ein autonomes Datenbanksystem. Studienarbeit, Institut für Informatik, Friedrich-Schiller-Universität Jena, Januar 2007.

[Ell10] E. Ellguth. Performance-Daten-Gewinnung und -Verwaltung in autonomen Datenbank-Tuning-Systemen. Studienarbeit, Institut für Informatik, Friedrich-Schiller-Universität Jena, April 2010.

[Eln04] S. Elnaffar: Towards Workload-Aware DBMSs: Identifying Workload Type and Predicting its Change. PhD-Arbeit, Queen's University, Canada, 2004.

[Enc08] MSN Encarta: Vegetatives Nervensystem. http://de.encarta.msn.com/, Juli 2008.

[EPBM03] S. Elnaffar, W. Powley, D. Benoit und P. Martin. Today's DBMSs: How Autonomic Are They? First International Workshop on Autonomic Computing Systems, Prag, Tschechische Republik, 2003.

[Exn07] T. Exner: Techniken zur Approximation und Optimierung von DB2-Performance-Funktionen. Diplomarbeit, Institut für Informatik, Friedrich-Schiller-Universität Jena, April 2007.

[Fab03] D. Faber: Utility computing: What killed HP's UDC? 2004. online, http://www.zdnet.com/news/utility-computing-what-killed-hps-udc/138727. Letzter Zugriff: Mai, 2010.

[FeNa99] D. Feitelson, M. Naaman. Self-tuning systems. IEEE Software 16(2):52-60, March/April 1999.

[Fin95] R. Findelstein: Understanding the Need for On-Line Analytical Servers, White Paper, http://www.arborsoft.com/papers/, 1995.

[FNGD93] D. Ferguson, C. Nikolaou, L. Georgiadis, K. Davies: Satisfying Response Time Goals in Transaction Processing Systems. Proc. 2nd Int. Conf. on Parallel and Distributed Information Systems (PDIS-93), 138-147, 1993.

[FPSS96a] U. Fayyad, G. Piatetsky-Shapiro, P. Smyth: From data mining to knowledge discovery in databases. AI Magazine, 1996.

[FPSS96b] U. Fayyad, G. Piatetsky-Shapiro, P. Smyth: From data mining to knowledge discovery: An overview. In Advances in Knowledge Discovery and Data Mining, Seiten 1-34. 1996. The MIT Press (1996) : Menlo Park [u. a.], S. 1 – 34.

[FPSSU96] U. Fayyad, G. Piatetsky-Shapiro, P. Smyth, R. Uthurusamy. Advances in Knowledge Discovery and Data Mining. AAAI/MIT Press, 1996.

[FPE01] G. Franklin, J. Powell, A. Emami-Naeini. Feedback Control of Dynamic Systems, fourth edition. Pearson Education, Upper Saddle River, NJ, 2001.

[GaCo03] A. Ganek, A. Corbi: The Dawning of the Autonomic Computing Era. IBM Systems Journal, 42 Vol 1:5–18, 2003.

[Gan06] K. Ganskow: Autonomic Database Performance Tuning: Formale System-Modellierung am Bsp. von IBM DB2 UDB. Diplomarbeit. Institut für Informatik, Friedrich-Schiller-Universität Jena, September 2006.

[Gao04] M. Gao: IBM DB2 Information Management - Technical Conference. In DB2 UDB Java Application Performance Tuning, Las Vegas, Nevada, September 2004.

[GHRU97] H. Gupta, V. Harinarayan, A. Rajaraman, J.D. Ullman: Index Selection for OLAP, in: Proceedings of the 13th International Conference on Data Engineering (ICDE 1997, Birmingham, U.K., 7.-11. April), 1997.

[GHQ95] A. Gupta, V. Harinarayan, D. Quass: Aggregate-Query Processing in Data Warehousing Environments, in: Proceedings of the 21th International Conference on Very Large Data Bases (VLDB 1995, Zürich, Schweiz, 11.-15. September), 1995.

[Göb08] A. Göbel: Konzeptuelle und prototypische Evaluierung der Möglichkeiten des Intelligent Miner im Autonomen Datenbank-Tuning. Studienarbeit. Juli 2008.

[GrRu02] J. Grabmeier, A. Rudolph: Techniques of Cluster Algorithms in Data Mining. Data Mining Knowledge Discovery, 2002.

[GrLe02] M. Gruninger, J. Lee: Ontology - applications and design. In Communications of ACM 45(2), Seiten 39-41, 2002.

[Gue97] D. Guerrero: Network Management & Monitoring with Linux. http://www.davidguerrero.com/papers/snmp/, Juni 1997.

[HaGu06] S. Hayes, P. Gunning: Tuning Up for OLTP and Data Warehousing, 2006. Online, Letzter Abruf 19.08.2009, http://www.dbazine.com/db2/db2-disarticles/hayes3.

[HaLi04] D. Hanselman, B. Littlefield: Mastering MATLAB 7. Prentice Hall. 2004.

[HäRa01] T. Härder, E. Rahm: Datenbanksysteme: Konzepte und Techniken der Implementierung. Springer-Verlag, 2. Auflage, 2001.

[Hay05] S. Hayes. Idug 2005 - Europe. In 8 Years of UDB Performance Solutions: Life Lessons from the field, Oktober 2005.

[Hel93] P. Helmich: Allgemeinmedizin - Grundlagen hausärztlichen Handelns. 1993.

[Hen07] C. Hennig: Klassifikation und Formalisierung von DB-Performance-Problemen, deren Diagnose und Auflösung anhand von Best-Practice-Methoden am Beispiel von IBM DB2 UDB. Studienarbeit, Institut für Informatik, Friedrich-Schiller-Universität Jena, August 2007.

[Herd99] O. Herden: Temporale Daten im Data Warehouse und Temporales OLAP. Rundbrief der GI-Fachgruppe 5.10 Informationssystem-Architekturen 1999(2): 1999.

[HHY99] D. Hart, J.L. Hellerstein, P.C. Yue: Automated drill down: An approach to automated problem isolation for performance management. In: Proceedings of the Computer Measurement Group. Tivoli Systems and IBM TJ Watson Research Center, 1999.

[HKMW01] H. Hippner, U. Küsters, M. Meyer, K Wilde: Handbuch Data Mining im Marketing. Friedr. Vieweg & Sohn Verlagsgesellschaft mbH, 2001.

[HMS01] D. Hand, H. Mannila, P. Smyth: Principles of Data Mining. MIT Press, Cambridge, MA, 2001.

[Hof09] A. Hofmeister: Konzeptionierung und prototypische Umsetzung eines Systems zur Verwaltung und zum Austausch von Tuningplänen. Diplomarbeit, Institut für Informatik, Friedrich-Schiller-Universität Jena, Februar 2009.

[Hor01] P. Horn: Autonomic computing: IBM's perspective on the state of information technology. IBM Research, 2001.

[HoRi07] M. Holze, N. Ritter: Towards Workload Shift Detection and Prediction for Autonomic Databases. In Proceedings of the ACM first Ph.D. workshop in CIKM, Lissabon, Portugal, 2007.

[HP03] Hewlett-Packard: HP Utility Data Center: Enabling Enhanced Datacenter Agility. Mai 2003. online, Abruf Mai 2010, http://h71028.www7.hp.com/-enterprise/downloads/udc_enabling.pdf

[HRU96] V. Harinarayan, A. Rajaraman, J.D. Ullman: Implementing data cubes efficiently. Proceedings of the 1996 ACM SIGMOD international conference on Management of Data (SIGMOD'96). ACM Press, June 1996.

[HuGa06] Huang, Y., D. Gannon: A Comparative Study of Web Services-based Event Notification Specifications. ICPP Workshop on Web Services-based Grid Applications, 2006.

[IBM00] IBM Coorperation: IBM DB2 UDB v7.1 - Performance Tuning Guide, Dezember 2000.

[IBM01] IBM Coorperation: Autonomic Computing Manifesto, http://www.research.ibm.com/autonomic/manifesto/autonomic_computing.pdf, 2001.

[IBM03a] IBM Coorperation: An architectural blueprint for autonomic computing (1st edition). IBM White Paper, 2003.

[IBM03b] N. Alur, A. Falos, A. Lau, S. Lindquist, M. Varghese: DB2 UDB/WebSphere Performance Tuning Guide. Redbook. IBM Corp, 2003.

[IBM04a] IBM Coorperation: DB2 II: Performance Monitoring, Tuning and Capacity Planning Guide, November 2004.

[IBM04b] IBM Coorperation: DB2 UDB V8 and WebSphere V5 Performance Tuning and Operation Guide, März 2004.

[IBM05] IBM Coorperation: Stand-alone Generic Log Adapter v4.2. User's Guide. 2005. http://www.eclipse.org/tptp/home/downloads/installguide/gla_42/-gla_users_guide.html

[IBM06a] IBM Coorperation. IBM DB2 Version 9 for Linux, Unix and Windows - Administration Guide: Implementation, August 2006.

[IBM06b] IBM Coorperation. IBM DB2 Version 9 for Linux, Unix and Windows - Administration Guide: Planning, August 2006.

[IBM06c] IBM Coorperation: IBM DB2 Version 9 for Linux, Unix and Windows - Performance Guide, August 2006.

[IBM06d] IBM Coorperation: DB2 Data Warehouse Edition Version 9.1.1 - Intelligent Miner Modeling: Administration and Programming Guide, 2006.

[IBM06e] IBM Coorperation: An architectural blueprint for autonomic computing (4th edition). IBM White Paper, 2006. http://www-03.ibm.com/autonomic-/pdfs/AC_Blueprint_White_Paper_V7.pdf

[IBM06f] IBM Coorperation: Active correlation technology: Act specification language, IBM confidential, 2006.

[IBM06g] IBM Coorperation: Active correlation technology: Rule writer's guide and reference, IBM confidential, 2006.

[IBM06h] IBM Coorperation: DB2 Data Warehouse Edition Version 9.1.1 - Intelligent Miner Modeling: Administration and Programming Guide, 2006.

[IBM06i] IBM Coorperation: IBM DB2 Version 9 for Linux, Unix and Windows - System Monitor Guide and Reference, August 2006.

[IBM06j] IBM Coorperation: IBM DB2 Version 9 for Linux, Unix and Windows - Visual Explain Tutorial, August 2006.

[IBM06k] IBM Coorperation: IBM DB2 Version 9 for Linux, Unix and Windows - Troubleshooting Guide, Juli 2006.

[IBM06l] IBM Coorperation: IBM DB2 Version 9 for Linux, Unix and Windows - Developing Java Applications, August 2006.

[IBM06m] IBM Coorperation: IBM DB2 Version 9 for Linux, Unix and Windows - Command Reference, August 2006.

[IBM06n] IBM Corporation: IT Service Management Standards: A Reference Model for Open Standards-Based ITSM Solutions, April 2006, ftp://ftp.software.ibm.com/software/tivoli/pdf/itsmstandardsreferencemodel.pdf

[IBM07] IBM Coorperation: New to Autonomic computing. http://www-128.ibm.com/developerworks/ autonomic/newto/, Oktober 2007.

[IBM08] IBM Coorperation: DB2 Performance Expert - Information Center, 2008 - letzter Zugriff: 10.05.2010.

[IBM08a] IBM Coorperation: IBM Tivoli Monitoring - Administrator's Guide , Version 6.2.1, SC32-9408-02, November 2008.

[IBM08b] IBM Coorperation: IBM Tivoli Monitoring - User Guide , Version 6.2.0, SC12-3588-01, Februar 2008.

[IBM09] IBM Coorperation: IBM Tivoli Monitoring - Information Center , http://publib.boulder.ibm.com/infocenter/tivihelp/v15r1/topic/com.ibm.itmfd.doc/welcome.htm , 2009 - letzter Zugriff: 11.5.2010.

[IBM10] IBM Coorperation: IBM DB2 Database for Linux, UNIX, and Windows Online Information Center. http://publib.boulder.ibm.com/infocenter-/db2luw/v9r5/index.jsp. Letzter Zugriff: April 2010.

[Inf96] Informix: Designing the Data Warehouse on Relational Databases. 1996.

[Inm92] W. Inmon: Building the Data Warehouse. 1. Auflage, Wiley, New York 1992.

[Jac88] V. Jacobson. Congestion avoidance and control. In Proceedings of the 1988 ACM Symposium on Communications Architectures and Protocols (SIGCOMM '88), Stanford, CA, August 1988.

[Jäg07] A. Jäger. Evaluierung von Konzepten, Problemen und Lösungsansätzen aktiver Datenbanksysteme und Untersuchung der Übertragbarkeit auf autonome Datenbanksysteme. Studienarbeit, Institut für Informatik, Friedrich-Schiller-Universität Jena, Dezember 2007.

[JBS97] S. Jablonski, M. Böhm, W. Schulze: Workflowmanagement: Entwicklung von Anwendungen und Systemen – Facetten einer neuen Technologie. dpunkt Verlag, Heidelberg 1997.

[Kap08] A. Kaplick: Erzeugen von kundenbezogenen Produktempfehlungen im Versandhandel unter Einsatz von Data-Mining-Technologien. Diplomarbeit, Institut für Informatik, Friedrich-Schiller-Universität Jena, April 2008.

[Kar08] T. Karabel: Entwurf und Implementierung eines web-basierten Assistenten zur Erstellung, Ausführung und Verwaltung von Datenbank-Tuning-Strategien. Bachelorarbeit. IBM Deutschland Entwicklung GmbH, Böblingen und Hochschule für Technik, Stuttgart; 2008.

[Kay03] R. Kay: QuickStudy: Event Correlation. Computerworld online Artikel unter http://www.computerworld.com/s/article/83396/Event_Correlation. Juli 2003.

[KeCh03] J. O. Kephart, D. M. Chess: The Vision of Autonomic Computing. IEEE Computer, Jan. 2003, pp. 41-50.

[Kes91] S. Keshav: A control-theoretic approach to flow control. In Proceedings of the 1991 ACM Symposium on Communications Architectures and Protocols (SIGCOMM '91), Zurich, Switzerland, September 1991.

[KeSo06] K. Kent, M. Souppaya: Guide to Computer Security Log Management, National Institute of Standards and Technology (NIST), September 2006. Online unter http://csrc.nist.gov/publications/nistpubs/800-92/SP800-92.pdf

[Kim96] R. Kimball: The Data Warehouse Toolkit. New York : John Wiley & SonsInc., 1996.

[KiRo02] R. Kimball, M. Ross: The Data Warehouse Toolkit. The Complete Guide to Dimensional Modeling. 2. Auflage. John Wiley & Sons, New York. 2002.

[Köh10] C. Köhler: Eine Monitoring-Infrastruktur für das (autonome) Datenbank-Performance-Tuning. Studienarbeit, Institut für Informatik, Friedrich-Schiller-Universität Jena, März 2010.

[Kra08] W. Krause: Workloadklassifikation zur Unterstützung des autonomen Datenbank-Performance-Tunings. Studienarbeit. Institut für Informatik, Friedrich-Schiller-Universität Jena, September 2008.

[Kri00] H.-P. Kriegel: Datenbanktechniken zur Unterstützung des Wissenserwerbs. Oldenbourg Wissenschaftsverlag GmbH, München, Wien, Oldenbourg, 2000.

[KRRT98] R. Kimball, L. Reeves, W. Thornthwaite, M. Ross: The Data Warehouse Lifecycle Toolkit, John Wiley & Sons, Inc., 1998.

[Krü09] J. Krüger: Verwaltung und Versionierung von „WebSphere sMash"-basierten Tuningplänen. Diplomarbeit. Institut für Informatik, Friedrich-Schiller-Universität Jena, Dezember 2009.

[KrSt05] H. Kreger, T. Studwell: Autonomic Computing And Web Services Distributed Management. http://www.ibm.com/developerworks/autonomic/library/ac-architect/, Juni 2005.

[Küs01] U. Küsters: Data Mining Methoden: Einordnung und Überblick. In: Hippner, Hajo ; Küsters, Ulrich ; Meyer, Matthias ; Wilde, Klaus (Hrsg.): Handbuch Data Mining im Marketing, Knowledge Discovery in Marketing Databases, S. 95 - 130. Vieweg : Braunschweig, Wiesbaden 2001.

[KZ02] W. Klösgen, J. M. Zytkow: Handbook of data mining and knowledge discovery. Oxford University Press, Inc., New York, NY, USA, 2002.

[Lar05] D. T. Larose: Discovering knowlege in data - An Introduction to Data Mining. John Wiley & Sons, Inc., Hoboken, New Jersey, 2005.

[Lar06] D. T. Larose. Data mining methods and models. John Wiley & Sons, Inc., Hoboken, New Jersey, 2006.

[Laz84] E. D. Lazowska, J. Zahorjan, G. S. Graham, K. C. Sevcik: Quantitative System Performance: Computer System Analysis Using Queueing Network Models. Prentice-Hall, Upper Saddle River, NJ, 1984.

[Leh03] W. Lehner: Datenbanktechnologie für Data-Warehouse-Systeme. Heidelberg: dpunkt.verlag 2003.

[LeMS95] A.Y. Levy, A. Mendelzon, Y. Sagiv: Answering Queries Using Views, in: Proceedings of the 14th ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems (PODS 1995, San Jose, USA, 22.-25. Mai), 1995.

[Li08] H. Li: Workload Characterization, Modeling, and Prediction in Grid Computing. PhD-Arbeit. ASCI Graduate School, Universität Leiden, Januar 2008.

[LiPl10] M. Liebisch, M. Plietz: Performance-Analysen für Realisierungsansätze im Kontext der Aspektorientierten Datenhaltung. Technischer Bericht, Friedrich-Schiller-Universität Jena, November 2010.

[Liu99] J. Liu: Data Mart - Next step in data storage. Term paper. University of Colorado 1999.

[LKK07] K. Lawrence, S. Kudyba, R. Klimberg. Data Mining Methods and Applications (Discrete Mathematics & Its Applications). Auerbach Publications, Boston, MA, USA, 2007.

[MaMi04] F. Manola, E. Miller: RDF Primer. http://www.w3.org/TR/rdf-primer/, Februar 2004.

[Mas90] H. Massalin, P. Calton: Fine-grain adaptive scheduling using feedback. Computing Systems 3(1):139-173, Winter 1990.

[Mat04] Matlab Optimization Toolbox: User's Guide. The MathWorks Inc., 2004.

[MaTo90] S. Martello, P. Toth: Knapsack Problems: Algorithms and Computer Implementations. Wiley Interscience Series in Discrete Mathematics and Optimization. John Wiley & Sons, New York, 1990. http://www.or.deis.unibo.it/knapsack.html.

[MCG+02] M. Möller, B. Callahan, V. Gucer, J. Hollis, S. Weber: Introducing Tivoli Distributed Monitoring Workbench 4.1 (IBM Redbook, 2002), http://www.redbooks.ibm.com/pubs/pdfs/redbooks/sg246534.pdf

[MeEi01] J. Melton, A. Eisenberg: SQL multimedia and application packages (SQL/MM). SIGMOD Rec., 30(4):97–102, 2001.

[MBD01] D. Menascé, D. Barbará, R. Dodge: Preserving QoS of ecommerce sites through self-tuning: A performance model approach. In Proceedings of the 2001 ACM Conference on Economic Commerce (EC '01), Tampa, FL, October 2001.

[MeWi00] P. Mertens, H.W. Wieczorrek: Data X Strategien - Data Warehouse, Data Mining und operationale Systeme für die Praxis. Springer : Berlin, Heidelberg 2000.

[MiHo03] C. Millsap, Jeff Holt: Optimizing Oracle Performance. O'Reilly & Associates, Inc., Sebastopol, CA, 2003.

[Mil05a] B. Miller: The autonomic computing edge: Can you CHOP up autonomic computing? Stand: 16.01.08. www.ibm.com/developerworks/autonomic/library/ac-edge4/, 2005.

[Mil05b] B. Miller: The autonomic computing edge: The role of knowledge in autonomic systems (http://www-128.ibm.com/developerworks/autonomic/library/ac-edge6/), 2005

[Mit97] T. Mitchell: Machine Learning. McGraw-Hill, 1997.

[Mog05] P. Mogin: OLAP Queries and SQL1999. Issues in Database and Information Systems. Victoria University of Wellington, 2005.

[MOKW06] H.A. Mueller, L. O'Brien, M. Klein, B Wood: Autonomic computing. http://handle.dtic.mil/100.2/ADA448227, 2006.

[MoSl93] J. Moffett, M. Sloman: Policy Hierarchies for Distributed System Management. In: IEEE JSAC Special Issue on Network Management 11 (1993), November, Nr. 9. http://citeseer.ist.psu.edu/article/moffett93policy.html

[MRC+97] J.N. Matthews, D. Roselli, A.M.. Costello, R. Wang, T. Anderson: Improving the performance of log-structured file systems with adaptive methods. In Proceedings of the 16th ACM Symposium on Operating System Principles (SOSP '97), Saint Malo, France, October 1997.

[MS10] MS SQL Server: Online Product Documentation. http://msdn.microsoft.com-/ens/library/bb418440%28v=SQL.10%29.aspx. Letzter Zugriff: April 2010.

[Mur04] R. Murch: Autonomic Computing. IBM Press, 2004.

[Neu06] S. Neumann: Use of the CIM Ontology. Conference Proceedings of DistribuTech 2006, February 7-9, 2006 - Tampa, Florida.

[NFC92] C. Nikolaou, D. Ferguson, P. Constantopoulus: Towards Goal-Oriented Resource Management. IBM Research Report RC 17919, IBM T.J. Watson Research Center, Yorktown Heights, 1992

[Nie99] R. J. Niemiec: Oracle Performance Tuning Tips and Techniques; McGrawHill, 1999.

[OAS06a] OASIS Web Services Distributed Management TC, Technical Committee: WSDM : Management Using Web Services (MUWS 1.1) Part1. http://www.oasisopen.org/committees/download.php/20576/wsdm-muws1-1.1-spec-os-01.pdf, August 2006.

[OAS06b] OASIS Web Services Distributed Management TC, Technical Committee: Web Services Distributed Management: Management of Web Services (WSDM-MOWS) 1.1. http://www.oasis-open.org/committees/download.php/20574/-wsdm-mows-1.1-spec-os-01.pdf, August 2006.

[OKSC04] D. Ogle, H. Kreger, A. Salahshour, J. Cornpropst: Canonical situation data format: The common base event v1.0.1 specification. Eclipse.org whitepaper. 2004. http://www.eclipse.org/tptp/platform/documents/resources/cbe101spec/CommonBaseEvent_SituationData_V1.0.1.pdf

[ONGr95] P. O'Neil, G. Graefe: Multi-Table Joins through Bitmapped Join Indeces, SIGMOD Record, Volume 24, Number 3, September 1995, S. 8-11.

[ONQu97] P. O'Neil, D. Quass: Improved Query Performance with Variant Indices, in: Proceedings ACM SIGMOD International Conference on Management of Data (SIGMOD 1997, Tucson, USA, 13.-15. Mai), 1997, S. 38-49.

[ORA10] Oracle Database: Online Documentation 11g Release 1. http://www.oracle.com/pls/db111/homepage. Letzter Zugriff: April 2010.

[Perk04] A. Perkins: Data Warehouse Architecture - A Blueprint for Success. http://www.dmreview.com/whitepaper/paper_sub.cfm?whitepaperId=10086, 2004.

[Pet05] H. Petersohn: Data Mining - Verfahren, Prozesse, Anwendungsarchitektur. Oldenbourg : München 2005.

[PH05] M. Parashar, S. Hariri: Autonomic Computing: An Overview. Technischer Bericht, INIST-CNRS, Cote INIST, 2005.

[PoMa06] W. Powley, P. Martin: A Reflective Database-Oriented Framework for Autonomic Managers. In Proceedings of International Conference on Autonomic Systems (ICAS'06), San Jose, CA, USA, 2006.

[PTB+03] M. Peleg, S. Tu, J. Bury, et al: Comparing Computer-interpretable Guideline Models: a Case Study Approach. Journal of the American Medical Informatics Association 10(1), 2003.

[PTV05] N. G. Pavlidis, D. K. Tasoulis, M. N. Vrahatis: Time series forecasting methodology for multiple-step-ahead prediction. In M. H. Hamza (Hrsg.), Computational Intelligence, Seiten 456-461. IASTED/ACTA Press, 2005.

[Rab06] G. Rabinovitch: Technologien und Konzepte zur autonomen Verwaltung von IT-Systemen. In Tagungsband zum 18. Workshop "18. Workshop über Grundlagen von Datenbanken", Institut für Informatik, Martin-Luther-Universität Halle-Wittenberg, Seiten 120-124, Wittenberg, Juni 2006.

[Rab09] G. Rabinovitch: Policy-based coordination of best-practice oriented autonomic database tuning. The First International Conference on Adaptive and Self-adaptive Systems and Applications (ADAPTIVE 2009), Athens, Greece, 15.-20. November, 2009.

[Rab11] G. Rabinovitch: Eine Beschreibungs- und Verwaltungsumgebung für autonomes Datenbank-Tuning. Dissertation, Institut für Informatik, Friedrich-Schiller-Universität Jena, *in Bearbeitung*.

[Ras04] S. Raspl. An overview of pmml version 3.0. In KDD-2004 Workshop on Data Mining Standards, Services and Platforms (DM-SSP 04), KDD-2004 The Tenth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, 2004.

[RaWi07] G. Rabinovitch, D. Wiese: Non-linear Optimization of Performance Functions for Autonomic Database Performance Tuning. In Proceedings of The Third International Conference on Autonomic and Autonomous Systems, Athen, Griechenland, 2007.

[RAWR08] M. Reichert, S. Arenswald, D. Wiese, G. Rabinovitch: Workload Aware Exception Detection. Patent Proposal, September 2008.

[Ree04] S. Rees: Zen and the Art of Database Performance. IBM DB2 Information Management - Technical Conference, Las Vegas, Nevada, September 2004.

[Ree05] S. Rees. Advanced Performance Diagnostics in DB2 UDB for L/U/W v8.2.x. Idug 2005 - Europe. Oktober 2005.

[Rei07] S. Reinisch: Evaluierung autonomer Mechanismen in MS SQL Server. Diplomarbeit, Institut für Informatik, Friedrich-Schiller-Universität Jena, Juni 2007.

[Rei08] M. Reichenbach: Entwicklung eines Modells zur Definition von Tuning-Richtlinien zur Steuerung des autonomen Datenbank-Tunings. Diplomarbeit, Institut für Informatik, Friedrich-Schiller-Universität Jena, Juli 2008.

[Rei81] D. Reiner, T. Pinkerton: A method for adaptive performance improvement of operating systems. In Proceedings of the 1981 ACM Conference on the Measurement and Modeling of Computer Systems (SIGMETRICS '81), Las Vegas, September 1981.

[RMH+09] B. Raza, A. Mateen, T. Hussain, M.M. Awais: Autonomic Success in Database Management Systems. ICIS, pp.439-444, 2009 Eigth IEEE/ACIS International Conference on Computer and Information Science (icis 2009), 2009.

[Roe08] K. Roeder: Konzipierung und prototypische Umsetzung einer Workload-Komponente zur gezielten Erzeugung von DB2-spezifischen Sperrproblemen im Umfeld des Autonomen Datenbank-Tunings. Studienarbeit, Institut für Informatik, Friedrich-Schiller-Universität Jena, August 2008.

[RWH06] M. Reichert, D. Wiese, N. Heck: Automatic Problem-Oriented Transformation of Database Performance Data. Februar 2006. Patent-Anmeldung (USPTO Application #: 20070185913).

[RWRA08] G. Rabinovitch, D. Wiese, M. Reichert, S. Arenswald: Datenbank-Tuning mit dem Autonomic Tuning Expert. Datenbank-Spektrum, Heft 27, Seiten 18-26. Dezember 2008.

[SaCa09] K. Sattler, S. Conrad: Vorlesung Data-Warehouse-Technologien, 2009. http://wwwiti.cs.uni-magdeburg.de/iti_db/lehre/dw/dw0001/dw04.pdf, Letzter Zugriff: 25. 03. 2010.

[Sap99] C. Sapia: On Modeling and Predicting Query Behavior in OLAP Systems. Proceedings of the International Workshop on Design and Management of Data Warehouses. Heidelberg, Germany, 14. - 15. 6. 1999.

[SaPr07] M. Saraswatipura, S. Prasad: Understanding the advantages of DB2 9 autonomic computing features. IBM Developerworks Article. November 2007. http://www.ibm.com/developerworks/data/library/techarticle/dm-0709saraswatipura/

[Sat07] K. Sattler: Self-*-Techniken in Datenbank-Management-Systemen: Grundlagen, Techniken, Systemüberblick. Tutorial der 11. BTW-Tutorientage 2007, Aachen.

[Sch99] R. Schaarschmidt: Konzept und Sprache für die Archivierung in Datenbanksystemen. Dissertation. Fakultät für Mathematik und Informatik, Friedrich-Schiller-Universität Jena, Dezember 1999.

[Sch03] R. Schumacher: Monitoring Databases the Right Way with Embarcadero Performance Analyst. Technischer Bericht, Embarcadero Technologies, Inc., Juni 2003.

[SCH04a] H. Schwarz: Data-Warehouse-, Data-Mining- und OLAP-Technologien: Chapter 3 - Data Warehouse Design. Lecture notes. University of Stuttgart, 2004.

[SCH04b] H. Schwarz: Data-Warehouse-, Data-Mining- und OLAP-Technologien: Chapter 5 - Online Anlytic Processing. Lecture notes. University of Stuttgart 2004.

[Sch05] C. Schaal: SQL Trendanalyse mit Hilfe mathematischer Verfahren. Diplomarbeit, Universität Stuttgart und IBM Böblingen, 2005.

[Sel97] M. Seltzer, C. Small: Self-monitoring and self-adapting operating systems. In Proceedings of the Sixth Workshop on Hot Topics on Operating Systems (HotOS-VI), Chatham, May 1997.

[SGG+99] D. C. Steere, A. Goel, J. Gruenberg, D. McNamee, C. Pu, J. Walpole: A feedback-driven proportion allocator for real-rate scheduling. In Proceedings of the Fourth Symposium on Operating Systems Design and Implementation (OSDI '99), New Orleans, LA, February 1999.

[SGL+06] A.J. Storm, C. Garcia-Arellano, S. Lightstone, et al.: Adaptive self-tuning memory in DB2. In Proceedings of VLDB, 2006.

[SGS03] K. Sattler, I. Geist, E. Schallen: QUIET: Continuous Query-driven Index Tuning. In Proceedings of VLDB, 2003.

[Shos82] A. Shoshani: Statistical Databases: Characteristics, Problems and some Solutions, in: Proceedings of the 8th International Conference on Very Large Data Bases (VLDB 1982, Mexico City, Mexico, 8.-10. Sept.), S. 208-2221982.

[Skat06] S. Skatulla: Speicherung und Indexierung komplexer Objekte in objektrelationalen Datenbank-Management-Systemen. Promotion, Friedrich-Schiller Universität Jena, 2006.

[Smi04] H. Smith: IBM DB2 Information Management - Technical Conference. In DB2/Java Performance Analysis and Tuning Considerations, September 2004. Las Vegas, Nevada.

[Sne03] B. Sneed: Performance Forensics - SMI Performance and Availability Engineering (PAE). Technischer Bericht, Sun Microsystems, Inc., Dezember 2003.

[SRR+06] M. Sedlmayr, T. Rose, R. Röhring, et al: A workflow approach towards GLIF execution. In Pro-ceedings on Workshop AI Techniques in Healthcare: Evidence-based Guidelines and Protocols. European Conference on Artificial Intelligence (ECAI), Trient, Italien, 2006.

[StBu03] R. Sterritt, D. Bustard: Autonomic Computing-a Means of Achieving Dependability? In Proceedings of IEEE International Conference on the Engineering of Computer Based Systems (ECBS'03), Seiten 247–251, April 2003.

[StHi05] R. Sterritt, M. Hinchey: Autonomic Computing Panacea or Poppycock?, Proceedings of the 12th IEEE International Conference and Workshops on the Engineering of Computer-Based Systems (ECBS05).

[Sto04] Stojanovic et al.: The role of ontologies in autonomic computing systems, IBM Systems Journal, 2004.

[Stö01] U. Störl: Backup und Recovery in Datenbanksystemen. Teubner 2001.

[Sul03] D. Sullivan: Using probabilistic reasoning to automate software tuning. Ph.D. Thesis, Harvard University, September 2003.

[SUN05] Sun Microsystems: Sun N1 System Manager 1.2 Introduction. 2005. http://dlc.sun.com/pdf/819-4140/819-4140.pdf

[TeUl97] M. Teschke, A. Ulbrich vom Ende: Using Materialized Views to Speed Up Data Warehousing, Technical Report, IMMD 6, Universität Erlangen-Nürnberg, 1997.

[Tho97] E. Thomsen: OLAP Solutions - Building Multidimensional Information Systems, Wiley Computer Publishing, 1997.

[TM06] V. Tewari, M. Milenkovic: Standards for autonomic computing. Intel Technology Journal, 2006.

[TPCC07] TPC Benchmark* C: Standard Specification. Revision 5.9. Transaction Processing Performance Council, 2007.

[TPCH08] TPC Benchmark* H: Standard Specification Revision 2.7.0. Transaction Processing Performance Council, 2008.

[Tre08] M. Treiber: Vorlesungsskript Statistik 1. Technische Universität Dresden, SS 2008.

[TuAr98] E. Turban, J.A. Aronson: Decision Support Systems and Intelligent Systems, Prentice Hall, 1998.

[VGD99] A. Vavouras, S. Gatziu, K. Dittrich: The SIRIUS Approach for Refreshing Data Warehouses Incrementally. In: BTW. Freiburg, 1999.

[YaLa87] H.Z. Yang, P.A. Larson: Query Transformations for PSJ Queries, in: Proceedings of 13th International Conference on Very Large Data Bases (VLDB 1987, Brighton, England, 1.-4. September), S. 245-254, 1987.

[Wei97] T. Weik: Terminierung und Konfluenz in einer aktiven objektorientierten Datenbank, Band 37. Infix Verlag, 1997.

[Wei10] A. Weiß: Gewinnung von Performance-Daten im Umfeld des Autonomen Datenbank-Tuning. Studienarbeit, Institut für Informatik, Friedrich-Schiller-Universität Jena, Februar 2010.

[WHMZ94] G. Weikum, C. Hasse, A. Möckeberg, P. Zabback: The Comfort Automatic Tuning Projekt, Information Systems 19(5), April 1994.

[Wie05] D. Wiese: Framework for Data Mart design and implementation in DB2 Performance Expert. Externe, vertrauliche Diplomarbeit in englischer Sprache für die IBM Deutschland Entwicklung GmbH. Böblingen, März 2005.

[Wie06] D. Wiese: Utilizing Data Marts for Performance Analysis. In Tagungsband zum 18. Workshop „Grundlagen von Datenbanken", Institut für Informatik, Martin-Luther-Universität Halle-Wittenberg, Seiten 150-15. Wittenberg, Juni 2006.

[Wie09] D. Wiese: Multidimensional Performance Management. International Conference on Computer Systems Design Engineering (ICCSDE 2009) der World Academy of Science, Engineering and Technology. Venice, Italy, 28.-30.Oktober 2009.

[WiRa07] D. Wiese, G. Rabinovitch: DB2 Autonomic Performance Management. IBM CAS Technical Report, IBM confidential, Juni 2007.

[WiRa09] D. Wiese, G. Rabinovitch: Knowledge Management in Autonomic Database Performance Tuning. The Fifth International Conference on Autonomic and Autonomous Systems (ICAS 2009), Valencia, Spanien, April 2009.

[WMHZ02] G. Weikum, A. Möckeberg, C. Hasse, P. Zabback: Self-tuning Database Technology and Information Services: from Wishful Thinking to Viable Engineering, 28th VLDB Conference, pp. 20-31, Hongkong, China, 2002.

[WRRA08] D. Wiese, G. Rabinovitch, M. Reichert, S. Arenswald: Autonomic Tuning Expert - A framework for best-practice oriented autonomic database tuning. In Proceedings of Centre for Advanced Studies on Collaborative Research (CASCON 2008). Ontario, Canada, 2008.

[WRRA09] D. Wiese, G. Rabinovitch, M. Reichert, S. Arenswald. ATE: Workload-oriented DB2 Tuning in Action. In Tagungsband zur 13. GI-Fachtagung Datenbanksysteme für Business, Technologie und Web (BTW) 2009. Demo. Münster, Deutschland, 2.-6. März 2009.

[WsM09] IBM Coorperation: WebSphere sMash 1.1.1 Information Center. Letzte Aktualisierung: 26 Juni 2009. http://publib.boulder.ibm.com/infocenter-/wsmashin/v1r1m1/index.jsp

[Wor04] D. Worden: Understand autonomic maturity levels. Stand: 15.01.08. www.ibm.com/developerworks/library/ac-mature.html, 2004.

[ZHA99] M. J. Zaki, C.-T. Ho, R. Agrawal: Parallel Classification for Data Mining on Shared-Memory Multiprocessors. Seite 198, Washington, DC, USA, 1999. IEEE Computer Society.

Hinweis: Eigene Veröffentlichungen bzw. Arbeiten, die direkt mit der vorliegenden Dissertation in Zusammenhang stehen (u.a. Studien- und Diplomarbeiten), wurden gesondert hervorgehoben.

Ehrenwörtliche Erklärung
zur Eröffnung des Promotionsverfahrens

Hiermit erkläre ich,

- dass mir die Promotionsordnung der Fakultät für Mathematik und Informatik der Friedrich-Schiller-Universität Jena bekannt ist,
- dass ich die Dissertation selbst angefertigt und alle von mir benutzten Hilfsmittel, persönlichen Mitteilungen und Quellen in meiner Arbeit angegeben habe,
- dass mich bei der Auswahl und Auswertung des Materials sowie bei der Herstellung des Manuskripts Herr Prof. Dr. Klaus Küspert (und sonst niemand) unterstützt hat,
- dass ich die Hilfe eines Promotionsberaters nicht in Anspruch genommen habe und dass Dritte weder unmittelbar noch mittelbar geldwerte Leistungen von mir für Arbeiten erhalten haben, die im Zusammenhang mit dem Inhalt der vorgelegten Dissertation stehen,
- dass ich die Dissertation noch nicht als Prüfungsarbeit für eine staatliche oder andere wissenschaftliche Prüfung eingereicht habe und
- dass ich die gleiche, eine in wesentlichen Teilen ähnliche oder eine andere Abhandlung nicht bei einer anderen Hochschule als Dissertation eingereicht habe.

Norderstedt, 30. Juni 2011

Selbstständigkeitserklärung

Ich erkläre, dass ich die vorliegende Arbeit selbstständig und nur unter Verwendung der angegebenen Hilfsmittel und Literatur angefertigt habe.

Norderstedt, 30. Juni 2011

Lebenslauf

Persönliche Informationen

Name	David Wiese
Geburtsdatum	13.September 1981
Geburtsort	Gera, Deutschland
Familienstand	Ledig

Schulische und universitäre Ausbildung

Schule	***Albert-Schweitzer-Gymnasium Gera***
1993 - 2000	Spezialisierung in Mathematik, Informatik und Englisch. Abitur am 30.06.2000.
Studium	***Friedrich-Schiller-Universität Jena***
Oktober 2000 – März 2005	Studium der Informatik mit Vertiefung in „Datenbanken und Informationssysteme“ (Lehrstuhl Prof. Dr. K. Küspert) und dem Nebenfach Wirtschaftswissenschaften.
Oktober 2004 – März 2005	Externe, vertrauliche Diplomarbeit („Framework for Data Mart Design and Implementation in DB2 Performance Expert“) bei der **IBM Deutschland Research & Development GmbH** in Böblingen, Baden-Württemberg.
Promotion	***Friedrich-Schiller-Universität Jena***
Seit 2008	Anfertigung der Promotion zum Thema „Gewinnung, Verwaltung und Anwendung von Performance-Daten zur Unterstützung des autonomen Datenbank-Tuning“ am Lehrstuhl für Datenbanken und Informationssysteme.

Berufliche Tätigkeiten

November 2002 – Februar 2003	Studentische Hilfskraft am **Institut für Stochastik**.
Januar 2006 – Oktober 2009	Hospitantenvertrag mit der **IBM Deutschland Research & Development GmbH**.
April 2005 - März 2010	Wissenschaftlicher Mitarbeiter am Lehrstuhl **für Datenbanken und Informationssysteme**, Institut für Informatik, Fakultät für Mathematik und Informatik, Friedrich-Schiller-Universität Jena.
Seit Oktober 2010	Software-Entwickler und System-Analytiker bei der **Lufthansa Revenue Services GmbH.**

Juni 2011