Beschreibung, Verwaltung und Ausführung von Arbeitsabläufen im 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. Gennadi Rabinovitch
geboren am 12. Juli 1978 in Gomel

Gutachter:

1. Prof. Dr. Klaus Küspert, Friedrich-Schiller-Universität Jena
2. Prof. Dr. Lutz Wegner, Universität Kassel
3. Dr. Andreas Wickenhäuser, IBM Deutschland Research & Development GmbH

Tag der öffentlichen Verteidigung: 20. April 2012

Die vorliegende Dissertation entstand auf Basis eines Kooperationsprojekts (*IBM Center for Advanced Studies, CAS*) zwischen der *IBM Deutschland Research & Development GmbH* und dem *Lehrstuhl für Datenbanken und Informationssysteme* der *Friedrich-Schiller-Universität Jena*. Die im Kontext dieses Kooperationsprojekts und im Rahmen der Dissertation durchgeführten Forschungsarbeiten wurden weiterhin durch die Vergabe eines *IBM Shared University Research Grant „Autonomic DB2 Performance Management“* (November 2007) sowie eines *IBM Faculty Award „Community-driven Management and Evolution of Database Tuning Strategies“* (November 2008) unterstützt.

Meiner Familie

Danksagung

Auf dem langen Weg von der Idee bis zur Fertigstellung der Dissertationsschrift wurde ich von einigen Menschen ganz wesentlich unterstützt. An dieser Stelle möchte ich ihnen herzlich danken.

Meinem Doktorvater, Herrn Prof. Dr. Klaus Küspert von der Friedrich-Schiller-Universität Jena, möchte ich dafür herzlich danken, dass er mich während des Studiums für die Datenbanken als Vertiefungsfach begeisterte und mir nach Abschluss des Studiums die Möglichkeit eröffnete, als Mitarbeiter an seinem Lehrstuhl zu lehren, zu forschen und zu promovieren. Im Besonderen gilt ihm mein Dank für die kontinuierliche fachliche und ausserfachliche Unterstützung während meiner Zeit an der FSU Jena und darüberhinaus.

Herrn Prof. Dr. Lutz Wegner von der Universität Kassel und Herrn Dr. Andreas Wickenhäuser von der IBM Deutschland Research & Development GmbH danke ich für die Übernahme der beiden weiteren Dissertationsgutachten und ihre wertvollen Kommentare.

Mein großer Dank gilt weiterhin den am Kooperationsprojekt beteiligten Mitarbeitern des IBM-Labors in Böblingen. Insbesondere seien hier Michael Reichert und Rüdiger Stumm genannt. Ihre Bemühungen und ihre fachliche sowie administrative Unterstützung waren von großer Wichtigkeit für den Erfolg des gemeinsamen Forschungsprojekts.

Mein Dank gebührt auch den zahlreichen Studenten, die durch ihre Studien- und Diplomarbeiten sowie durch ihre Programmiertätigkeiten wesentlich zum Gelingen der Arbeit beigetragen haben.

Carola Eichner sowie meinen (ehemaligen) Kollegen Dr. Klaus Friedel, Andreas Göbel, Matthias Liebisch, Dr. Thomas Müller, Dr. Knut Stolze und Dr. David Wiese danke ich für das angenehme Arbeitsumfeld am Lehrstuhl.

Schließlich möchte ich mich ganz herzlich bei meiner Familie für die motivierende Unterstützung in den vergangenen Jahren bedanken. Ihnen sei diese Arbeit vom ganzen Herzen gewidmet.

Kurzfassung

Durch die in den letzten Jahrzehnten gestiegene Komplexität und Heterogenität von IT-Systemen wurde auch ihre Administration deutlich aufwendiger und kostspieliger. Zur Gewährleistung einer hohen Verfügbarkeit und Performance dieser Systeme reichen eine kontinuierliche manuelle Administration und Optimierung der Systeme im laufenden Betrieb kaum noch aus. Der Trend der heutigen IT geht daher zum einen in Richtung der Entwicklung „intelligenter“ Werkzeuge zur Unterstützung der Administratoren. Zum anderen wird versucht, die Administrationskomplexität neuer Systeme zu reduzieren, indem Mechanismen, die einzelne Teilaspekte der Administration autonom umsetzen, entwickelt und in diese Systeme integriert werden.

Diesem Ziel haben sich Initiativen wie das *Autonomic Computing* verschrieben. Sie ermöglichen eine Automatisierung der komplexen Systemverwaltungs- und -konfigurationsaufgaben und ihre anschließende Übertragung an die Systeme selbst. Die vorliegende Arbeit entstand im Rahmen eines Kooperationsprojekts zwischen der *IBM Deutschland Research & Development GmbH* und dem *Lehrstuhl für Datenbanken und Informationssysteme* der *Friedrich-Schiller-Universität Jena*, welches das Ziel verfolgte, die Übertragbarkeit der Konzepte des *Autonomic Computing* auf das Datenbank-Tuning zu untersuchen und eine Infrastruktur zur Automatisierung typischer Datenbank-Tuning-Aufgaben unter Reduzierung menschlicher Interaktion zu konzipieren.

Als eine der Grundvoraussetzungen für die Automatisierung der Datenbank-Tuning-Aufgaben wurden die Beschreibung und Modellierung des Tuning-Wissens identifiziert. Die im Rahmen der vorliegenden Arbeit vorgestellten Konzepte ermöglichen es den Administratoren daher, sowohl die Problemsituationen als auch die entsprechenden bewährten Tuning-Abläufe zu erfassen und im System zu hinterlegen.

Mit Hilfe einer aufbauend auf den Konzepten zur Formalisierung des Tuning-Wissens konzipierten Architektur lassen sich IT-Systeme kontinuierlich überwachen und beim Feststellen eines problematischen Verhaltens entsprechende, vorab im System hinterlegte Tuning-Abläufe einleiten. Zur Unterstützung und Steuerung des Überwachungs- sowie Tuning-Prozesses werden in dieser Arbeit Konzepte zur automatischen Erkennung der am überwachten System aktuell anliegenden Arbeitslast und zur Einbeziehung von Metadaten über das überwachte System bzw. über die in der Datenbasis vorliegenden Tuning-Praktiken sowie die Historie ihrer Ausführungen vorgestellt. Mittels spezieller Tuning-Richtlinien können weiterhin Tuning-Ziele bzw. andere Nebenbedingungen spezifiziert werden, die zur Laufzeit berücksichtigt werden und somit den Tuning-Prozess indirekt beeinflussen. Des Weiteren wird hier ein Ansatz vorgestellt, der den Administratoren neben der Modellierung von ihren bewährten Tuning-Praktiken auch die Modellierung der Planungslogik des Tuning ermöglicht.

Zur Unterstützung einer kollaborativen Entwicklung und eines Austauschs von Tuning-Praktiken wird in dieser Arbeit eine Community-Plattform konzipiert. Dabei wird auf eine effiziente, feingranulare, semantikreiche Speicherung und Versionierung von Tuning-Praktiken sowie Konzepte zu ihrer Evolution im Mehrbenutzerbetrieb Wert gelegt. Weiterhin spielen bei den Betrachtungen die Auffindbarkeit von Tuning-Praktiken und Konzepte zur Motivation der Nutzer, sich am Austauschprozess zu beteiligen, eine wichtige Rolle.

Inhaltsverzeichnis

Abbildungsverzeichnis

Tabellenverzeichnis

Verzeichnis der Listings

Kapitel 1

Einleitung

Durch eine in den letzten Jahrzehnten stattgefundene rapide Weiterentwicklung der Hard- und Software und somit eine steigende Leistungsfähigkeit der IT-Systeme erkannten die IT-Anwender das Potenzial, immer mehr Aufgaben an die IT-Systeme übertragen zu können. Dies führte zu einer kontinuierlich zunehmenden Komplexität dieser Systeme. Diese Komplexität ist ein schwerwiegendes Problem der heutigen Informationstechnologie [IBM01]. Nicht nur die Bedienung heutiger IT-Systeme, sondern insbesondere auch ihre Administration sind zunehmend schwieriger geworden. Für die meisten Unternehmen hat die IT jedoch mittlerweile eine strategische Rolle eingenommen und ist unverzichtbar. Kleinste Ausfälle oder Performance-Probleme wirken sich unmittelbar auf die Wertschöpfungskette des Unternehmens aus und müssen daher nach Möglichkeit vermieden oder zumindest schnellstmöglich erkannt und behoben werden.

Eine hohe Verfügbarkeit, Zuverlässigkeit und Performance der Systeme sowie die Einhaltung von vorgegebenen Richtlinien setzen unter anderem eine vernünftige Ressourcenplanung, Wartung, Administration und Optimierung der IT-Systeme voraus, was den Einsatz von gut ausgebildetem Administrationspersonal unabdingbar macht. Aufgrund der hohen Komplexität und Heterogenität eingesetzter IT-Systeme reicht es für große Unternehmen längst nicht mehr aus, einen oder wenige Administratoren zu beschäftigen. So sind heutzutage Administrationstätigkeiten auf verschiedenen Ebenen der eingesetzten IT notwendig: von System- über Netzwerk- bis hin zu Datenbankadministratoren (DBA) bzw. Anwendungsbetreuern. Zu ihren Hauptaufgaben gehört neben der Installation und initialen Konfiguration der Systeme eine kontinuierliche Überwachung des Systemverhaltens und eine zeitnahe Behebung von erkannten Problemen.

Die für den Unternehmenserfolg notwendigen, gut ausgebildeten und teuren Administratoren müssen jedoch nicht nur ständig geschult werden, um immer komplexer werdende Systemkomponenten zu administrieren, auch ihre Verfügbarkeit ist aufgrund von Urlaubszeiten, Weiterbildungszeiten, krankheitsbedingten Ausfällen bzw. Pausen nur schwer rund um die Uhr zu gewährleisten. Aber nicht nur die unter Umständen eingeschränkte Verfügbarkeit, sondern auch zum Teil fehlende Erfahrung und Kompetenz wirken sich oft hinderlich auf eine zeitnahe Erkennung und Auflösung von aufgetretenen Problemen aus.

Manuelle Administration ist somit in den meisten Fällen nur noch durch erfahrene und gut ausgebildete Spezialisten durchführbar. Diese sind jedoch rar und tragen wesentlich zu Betriebskosten (*Total Cost of Ownership, TCO*) der IT-Systeme bei. Der Trend der heutigen IT geht daher zum einen in Richtung der Entwicklung intelligenter Werkzeuge

zur Unterstützung der Administratoren. Das wirkt zwar der Komplexität der Administration entgegen, reduziert jedoch nur bedingt die typischen Probleme der manuellen Administration wie die Fehleranfälligkeit, fehlende Kompetenz der Administratoren, zeitliche Verzögerungen nach Problemauftreten bis zur Lösung etc. Zum anderen wird versucht, die Administrationskomplexität der neuen Systeme zu reduzieren, indem intelligente Mechanismen, die Teilaspekte der Administration autonom umsetzen, entwickelt und in die Systeme integriert werden.

Diesem Ziel hat sich unter anderem die im Oktober 2001 ins Leben gerufene Initiative der *IBM* namens *Autonomic Computing* [IBM01, IBM06a] verschrieben. In Anlehnung an das vegetative Nervensystem des Menschen sollen hier die wichtigen, den Gesundheitszustand des Systems beschreibenden Indikatoren überwacht werden. Sobald ihre Werte den „gesunden“ Wertebereich verlassen, soll autonom mit entsprechenden Aktionen reagiert werden. Das Ziel der Initiative ist die Entwicklung von IT-Systemen, die sich flexibel auf sich ständig veränderbare Arbeitslast sowie andere wechselnde Umgebungsbedingungen einstellen können und in der Lage sind, eventuelle Probleme zu erkennen und autonom zu beheben, ohne Eingriffe der Administratoren zu erfordern. Da die Administratoren dadurch von ihren Routineaufgaben entlastet werden, verschiebt sich ihr Tätigkeitsschwerpunkt auf die langfristige, strategische Planung und Steuerung der autonom agierenden Systeme durch Definition von im Rahmen der autonomen Administration zu befolgenden Richtlinien.

Die Umsetzung vollständig autonom agierender, selbstverwaltender Systeme ist ein sehr langwieriger Prozess. Besonders fortgeschritten auf diesem Gebiet sind die aktuellen Versionen der Datenbank-Management-Systeme (DBMS) der großen DBMS-Hersteller wie *IBM*, *Oracle* bzw. *Microsoft*. Sie haben bereits erste große Schritte zur Selbstverwaltung erfolgreich gemacht, ihre autonomen Funktionalitäten beschränken sich aktuell jedoch auf einzelne Teilaspekte der Administration und reichen zu einer System-weiten bzw. gar System-übergreifenden Administration sowie Leistungssteigerung ohne Intervention der Datenbankadministratoren (DBA) noch nicht aus.

Vor diesem Hintergrund wurde 2005 ein Kooperationsprojekt zwischen der *IBM Deutschland Research & Development GmbH* und dem *Lehrstuhl für Datenbanken und Informationssysteme* der *Friedrich-Schiller-Universität Jena* ins Leben gerufen. Ziel dieses Projekts ist die Konzeption einer Infrastruktur zur Automatisierung typischer Datenbank-Tuning-Aufgaben unter Reduzierung menschlicher Interaktion [KWR05, KRW+07].

Als Grundvoraussetzungen für die Automatisierung der Datenbank-Tuning-Aufgaben wurden die Beschreibung und Modellierung des Tuning-Wissens identifiziert. Die im Rahmen der vorliegenden Arbeit vorgestellten Konzepte ermöglichen es den Administratoren daher, sowohl die Problemsituationen als auch die entsprechenden bewährten Tuning-Vorgehensweisen zu erfassen und im System zu hinterlegen.

Mit Hilfe des im Rahmen des Projekts entwickelten *Autonomic Tuning Expert (ATE)* lassen sich nun Systeme kontinuierlich überwachen und beim Feststellen eines problematischen Verhaltens entsprechende, im System hinterlegte Reaktionsabläufe einleiten. Bei der Entscheidung darüber, welche Tuning-Abläufe (Tuning-Pläne) beim Auftreten eines bestimmten Problems initiiert werden, lassen sich sowohl die Information über die anliegende Workload als auch diverse Metadaten über das überwachte System bzw. über die in der Datenbasis vorliegenden Tuning-Praktiken sowie die Historie ihrer Ausführungen und damaliger Effektivität miteinbeziehen. Mittels spezieller Tuning-Richtlinien können

Tuning-Ziele bzw. andere Nebenbedingungen spezifiziert werden, die zur Laufzeit vom ATE berücksichtigt werden und somit den Tuning-Prozess indirekt beeinflussen.

Der Ansatz zeichnet sich dabei durch seine Universalität aus, da der Zugriff auf die zu tunenden Komponenten ausschließlich über die nach außen sichtbaren Schnittstellen erfolgt und kein Einblick in die Systeminterna benötigt wird. Somit ist er nicht auf das Tuning eines einzelnen Systems beschränkt, sondern kann zum System-übergreifenden Tuning eingesetzt werden, indem beispielsweise einzelne Systeme wie das Betriebssystem, das DBMS, der Anwendungs- sowie der Webserver gleichzeitig optimiert und aufeinander abgestimmt werden.

Die Überführung unstrukturierter, bei den Administratoren selbst liegender bzw. in Produkthandbüchern, Foren, Newsgroups oder IT-Blogs zur Verfügung stehender Tuning-Informationen in formale, maschinell abarbeitbare Tuning-Abläufe ist ein äußerst zeitraubender und mühsamer Prozess. Zur Unterstützung dieses Prozesses wurde eine webbasierte Community-Plattform konzipiert, die den Austausch von Datenbank-Tuning-Praktiken unter den Community-Mitgliedern ermöglicht. Dabei gilt es nicht nur, die Erfassung und Weitergabe eines häufig unvollständig oder unter Umständen gar nicht dokumentierten Tuning-Wissens einzelner Experten umzusetzen, sondern auch eine zentralisierte, effiziente Speicherung dieses Wissens zu gewährleisten und seine Evolution im Mehrbenutzerbetrieb sowie seine Wiederauffindbarkeit zu unterstützen. Bei der Umsetzung der Community-Plattform spielen insbesondere auch Community-Funktionen eine wichtige Rolle. Dabei sollen die Community-Mitglieder bei der Informationsbereitstellung bzw. -benutzung unterstützt und zu einer Beteiligung am Wissensaustausch motiviert werden.

Zu den Schwerpunkten dieser Arbeit zählen die Beschreibung von Konzepten zur Modellierung von Problemsituationen und der damit verknüpften Datenbank-Tuning-Praktiken sowie die Vorstellung eines Architekturvorschlags zur kontinuierlichen Überwachung des Systems und einer Einleitung problemkorrigierender Abläufe beim Auftreten von Problemen. Auch Konzepte, die es ermöglichen, die Qualität des Tuning zu steigern bzw. durch die Angabe von Tuning-Richtlinien zu beeinflussen, spielen in dieser Arbeit eine entscheidende Rolle. Schließlich werden hier ausgewählte Konzepte zur zentralisierten Verwaltung und Austausch von Tuning-Plänen mittels einer Community-Plattform vorgestellt und ein Vorschlag für eine entsprechende Systemarchitektur unterbreitet. Dabei grenzt diese Arbeit an die 2011 fertiggestellte Dissertationsschrift von D. Wiese [Wie11] an, die sich hauptsächlich mit der Gewinnung, Verwaltung und Anwendung von Performance-Daten befasste und somit eine Grundlage für das Autonome Datenbank-Tuning bildet.

Abbildung 1.1 skizziert grob das Zusammenspiel der wichtigsten in dieser Arbeit behandelten Themen und verweist dabei auf die entsprechenden Kapitel, die diese Aspekte beschreiben. Für eine detaillierte Beschreibung der Inhalte einzelner Kapitel sei auf *Abschnitt 1.2* verwiesen.

1.1 Ziele der Arbeit

Dieser Abschnitt fasst die wichtigsten Ziele sowie die wesentlichen Beiträge der vorliegenden Arbeit zusammen. Zur Wahrung einer besseren Übersichtlichkeit wurden die Ziele dabei nach Themenbereichen zusammengefasst.

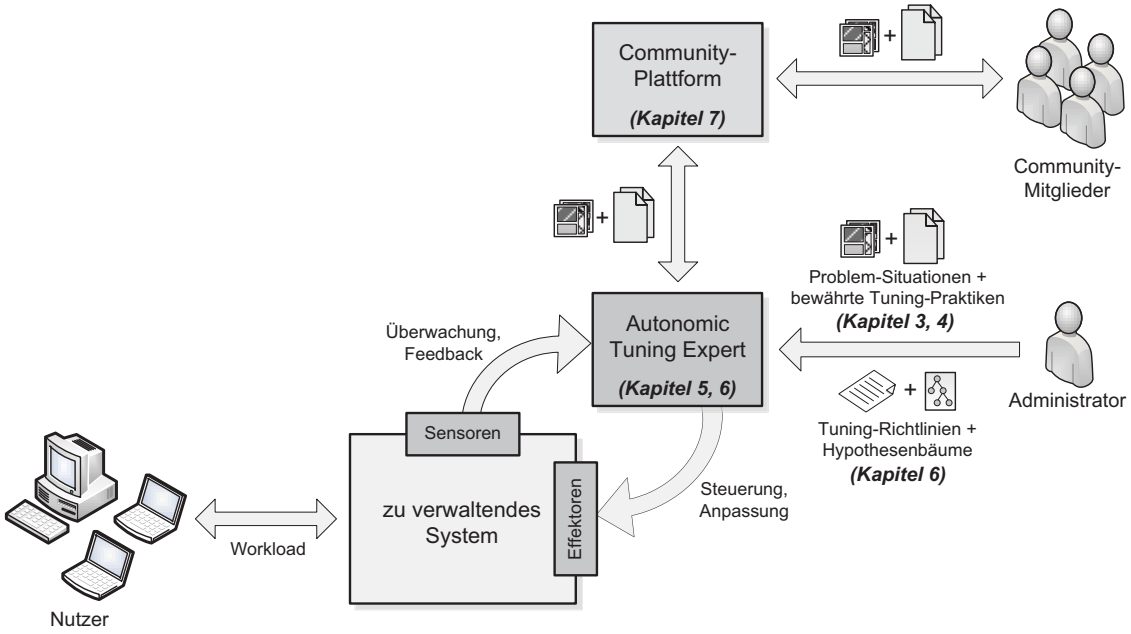


Abbildung 1.1: Kontext und Aufbau der Arbeit

Konzepte des *Autonomic Computing* und des autonomen Datenbank-Tuning

Es wird ein kurzer Überblick über die Thematik des *Autonomic Computing* gegeben. Basierend darauf werden anschließend die Konzepte zur Umsetzung autonomer Eigenschaften in IT-Systemen hinsichtlich ihrer Übertragbarkeit auf autonomes Datenbank-Tuning untersucht.

Formalisierung und Modellierung bewährter Tuning-Vorgehensweisen der DBA

Es wird aufgezeigt, wie sich die Problemdiagnose sowie die typischerweise darauf folgenden bewährten Tuning-Vorgehensweisen der Datenbankadministratoren formalisieren und modellieren lassen und welchen Nutzen eine solche Formalisierung bzw. Modellierung bei der Umsetzung des autonomen Tuning datenbankbasierter Software-Systeme und somit bei der Entlastung der Administratoren bei ihren alltäglichen Routinetätigkeiten mit sich bringt.

Zur Ermöglichung einer Problemerkennung bzw. -diagnose ist eine formalisierte **Beschreibung von Problemsituationen** unabdingbar, weshalb sie einen Schwerpunkt der vorliegenden Arbeit bildet. Zur **Formalisierung eigentlicher, nach der Problemerkennung initiierter Tuning-Abläufe** wird eine erweiterbare Sprache entworfen, die einen Satz vordefinierter Tuning-spezifischer Basisaktivitäten bereitstellt und mittels Kontrollstrukturen ihre Verknüpfung zu speziellen Tuning-Workflows (*Tuning-Plänen*) ermöglicht. Anschließend wird untersucht, welche Werkzeuge sich zur **Modellierung von Tuning-Plänen** eignen und wie sie eingesetzt werden können.

Eine Infrastruktur zur probleminitiierten Ausführung von Tuning-Plänen

Ein weiteres wichtiges Ziel dieser Arbeit bildet die Vorstellung einer Architektur zum best-practices-orientierten Datenbank-Tuning, die auf den Konzepten zur Formalisierung des Tuning-Wissens aufbaut und eine kontinuierliche **Überwachung ei-**

nes zu optimierenden Systems, die **Erkennung von aufgetretenen Problemen** und schließlich die **Einleitung von Tuning-Plänen**, die zur Behebung dieser Probleme konzipiert wurden, ermöglicht. Zu einem wichtigen Bestandteil dieser Architektur gehört weiterhin eine Workload-Erkennungskomponente, die basierend auf einer Auswahl Workload-relevanter Systemmetriken eine **System- und Tuning-unabhängige Erkennung des Workload-Typs** (beispielsweise OLTP bzw. OLAP) umsetzt. Die Kenntnis des Workload-Typs kann dabei unter anderem bei der Auswahl oder Ausführung von Tuning-Plänen berücksichtigt werden.

Konzepte zur Steuerung des autonomen Datenbank-Tuning

Nicht nur die Kenntnis des aktuellen Workload-Typs hat einen entscheidenden Einfluss auf das Datenbank-Tuning. Insbesondere können dabei weitere Informationen, wie die Abhängigkeiten unter den einzelnen Sensoren und Effektoren, die mit der Änderung der Effektoren verbundenen Kosten bzw. die Historie der Tuning-Plan-Ausführungen sowie ihrer damaliger Effektivität Anwendung finden.

Es sind daher **Metadaten** zu identifizieren, die zur Unterstützung der Tuning-Steuerung eingesetzt werden können und beispielsweise die Auswahl von zur Auflösung eines erkannten Problems unter gegebenen Nebenbedingungen geeigneten Tuning-Plänen unterstützen oder Einfluss auf die Ausführung dieser Tuning-Pläne haben.

Diese Metadaten sollen als Grundlage für die Konstruktion eines **Wissensmodells für das autonome Datenbank-Tuning** verwendet werden. Dieses Wissensmodell soll nicht nur das für die einzelnen Phasen des Tuning-Prozesses notwendige Wissen bereitstellen, sondern auch die Weitergabe des in diesen Phasen gesammelten bzw. generierten Wissens an entsprechende Komponenten des Tuning-Systems ermöglichen.

Autonom agierende Systeme stellen typischerweise eine Schnittstelle bereit, die es den Nutzern ermöglicht, Zielvorgaben zu definieren und dadurch das Verhalten des Systems indirekt zu beeinflussen. Zur Steuerung des hier vorgestellten autonomen Datenbank-Tuning werden in der vorliegenden Arbeit eine eigens dafür konzipierte **Sprache zur Definition von Tuning-Richtlinien** sowie das zugrunde liegende Modell vorgestellt.

Während sowohl die Metadaten als auch die Tuning-Richtlinien unter anderem dazu verwendet werden, die Planungsphase des Tuning-Prozesses indirekt zu beeinflussen, werden in dieser Arbeit auch Konzepte erarbeitet, die eine unmittelbare **Einflussnahme auf die Planung durch die Modellierung des gesamten Planungsablaufs** ermöglichen. Mit Hilfe von sogenannten Hypothesenbäumen soll den Administratoren die Möglichkeit geboten werden, das Tuning ganzer Problembereiche selbstständig zu planen und zu modellieren.

Konzepte zur zentralisierten Verwaltung und Austausch von Tuning-Plänen

Zur Ermöglichung des Austauschs von Tuning-Plänen in einer Community ist eine Community-Plattform zu konzipieren. Nach einem Einstieg in die Thematik ist dazu unter anderem zu untersuchen, welche Aspekte der Versionierung im Kontext des best-practices-orientierten autonomen Datenbank-Tuning Anwendung finden bzw. wie eine **zentralisierte Verwaltung von Tuning-Plänen** umgesetzt werden kann.

Aber auch Aspekte wie die **Evolution von Tuning-Plänen im Mehrbenutzerbetrieb** bzw. die **Auffindbarkeit von Tuning-Plänen** und nicht zuletzt die **Community-Funktionen zur Motivation der Nutzer** spielen in dieser Arbeit eine wichtige Rolle.

1.2 Aufbau der Arbeit

Zu einem besseren Verständnis der nachfolgenden Beschreibung der Inhalte einzelner Kapitel sei auf *Abbildung 1.1* verwiesen. Die Abbildung veranschaulicht den Kontext und skizziert grob das Zusammenspiel der wichtigsten in dieser Arbeit behandelten Themen sowie ihre Zuordnung zu den entsprechenden Kapiteln.

Im *Kapitel 2* werden zunächst grundlegende Begriffe, Konzepte und ausgewählte Grundlagen vorgestellt, die zum Verständnis dieser Arbeit beitragen sollen. Die Ausführungen konzentrieren sich dabei auf die Themenbereiche wie Performance-Tuning, *Autonomic Computing* sowie autonomes Datenbank-Tuning und bieten einen Einblick in die Begriffswelt des im Verlauf der vorliegenden Arbeit für zahlreiche Beispiele gewählten Datenbank-Management-Systems *IBM DB2*.

Um typische Datenbank-Tuning-Aufgaben zu automatisieren, ist eine Beschreibung und Modellierung des Tuning-Wissens unabdingbar. Im *Kapitel 3* werden daher Konzepte zur Beschreibung von Performance-Problemen untersucht, die ihre Modellierung durch die Administratoren und ihre automatische Erkennung zur Laufzeit durch entsprechende Software-Komponenten ermöglichen.

Das darauffolgende *Kapitel 4* befasst sich mit der Beschreibung von bewährten Tuning-Vorgehensweisen der Administratoren. Basierend auf einer Untersuchung von typischen Aufgaben und Diagnose- bzw. Tuning-Vorgehensweisen der DBA wird eine Sprache konzipiert, die einen Satz vordefinierter, auf das Tuning datenbankbasierter IT-Systeme zugeschnittener Basisaktivitäten bereitstellt und mittels Kontrollstrukturen ihre Verknüpfung zu speziellen Tuning-Workflows (*Tuning-Plänen*) ermöglicht. Anschließend wird untersucht, welche Werkzeuge sich zur Modellierung von Tuning-Plänen eignen und wie sie eingesetzt werden können.

Da neben dem Wissen über die typischen Tuning-Vorgehensweisen auch das Wissen über das zu tunende System bzw. seine Umgebung einen zentralen Baustein des autonomen Datenbank-Tuning bildet, wird im *Kapitel 5* ein Wissensmodell vorgestellt, welches diese Informationen umfasst. Dieses Wissensmodell bildet dabei die Grundlage für die im Rahmen des Kooperationsprojekts zwischen *IBM* und dem Lehrstuhl entstandene und ebenfalls in diesem Kapitel beschriebene Infrastruktur zur Automatisierung typischer Datenbank-Tuning-Aufgaben unter Reduzierung menschlicher Interaktion. Der vorgestellten Infrastruktur liegt die aus der Kontrolltheorie bzw. aus dem *Autonomic Computing* bekannte rückgekoppelte Kontrollschleife zugrunde. Aufbauend auf den Konzepten zur Formalisierung des Tuning-Wissens wird hier eine kontinuierliche Überwachung des zu optimierenden Systems, die Erkennung von aufgetretenen Problemen und die Einleitung von zur Auflösung dieser Probleme konzipierten Tuning-Plänen unter Berücksichtigung der aktuell anliegenden Workload ermöglicht.

Ausgewählte Ansätze zur Steigerung der Tuning-Qualität und Steuerung des Tuning finden sich im *Kapitel 6*. Zum einen werden hier Metadaten identifiziert und kategorisiert, die

sich zur Unterstützung der Tuning-Steuerung einsetzen lassen. Zum anderen stellt dieses Kapitel die im Rahmen unserer Forschungsarbeiten konzipierte Sprache zur Definition von Tuning-Richtlinien vor, welche es den Administratoren ermöglicht, Zielvorgaben an das zu tunende System bzw. weitere beim Tuning zu berücksichtigenden Nebenbedingungen zu definieren. Einen weiteren wichtigen Bestandteil dieses Kapitels bildet die Vorstellung des Hypothesenbaum-Ansatzes, welcher eine nutzerdefinierte Planung unterstützt, indem er den Administratoren eine Möglichkeit zur Beschreibung und Modellierung der Planungslogik bereitstellt.

Des Weiteren beschäftigt sich diese Arbeit mit der Erarbeitung von Konzepten zur Ermöglichung eines Austauschs des erfassten Tuning-Wissens in einer Community. *Kapitel 7* stellt daher zunächst vor, welche Konzepte bei der Umsetzung eines Repository zur Verwaltung von Tuning-Plänen Anwendung finden. Dabei spielen Konzepte wie Versionsverwaltung, Evolution von Tuning-Plänen im Mehrbenutzerbetrieb bzw. Auffindbarkeit von Tuning-Plänen eine wichtige Rolle. Anschließend wird untersucht, welche Community-Funktionen zur Motivation von Nutzern, sich am Wissensaustausch mittels der Community-Plattform zu beteiligen, eingesetzt werden können und wie die Benutzer dadurch bei der Informationsbereitstellung bzw. -benutzung unterstützt werden. Schließlich stellt das Kapitel einen Architekturvorschlag zur Umsetzung der Community-Plattform vor.

Anschließend fasst das *Kapitel 8* die Ergebnisse der vorliegenden Arbeit kompakt zusammen und gibt einen Ausblick auf mögliche weiterführende Arbeiten.

1.3 Eigene themenrelevante Veröffentlichungen und Vorträge

Nachfolgend findet sich eine Auswahl von Veröffentlichungen, die im Kontext der vorliegenden Arbeit entstanden sind und einige der in dieser Arbeit vorgestellten Konzepte beinhalten (alphabetisch geordnet):

- Küspert, Klaus ; Rabinovitch, Gennadi ; Wiese, David ; Stumm, Rüdiger ; Reichert, Michael: *Autonomic Database Performance Tuning.* Vortrag anlässlich der Verleihung des IBM Shared University Research Grant 2007 an den Lehrstuhl für Datenbanken und Informationssysteme, Friedrich-Schiller-Universität Jena. November 2007

- Küspert, Klaus ; Wiese, David ; Rabinovitch, Gennadi: *Autonome Datenbank-Administration.* Poster zum Tag der Forschung der Friedrich-Schiller-Universität Jena, Oktober 2005

- Rabinovitch, Gennadi: Technologien und Konzepte zur autonomen Verwaltung von IT-Systemen. In: *Tagungsband zum 18. Workshop über Grundlagen von Datenbanken, Institut für Informatik, Martin-Luther-Universität Halle-Wittenberg.* Wittenberg, Juni 2006, S. 120–124

- Rabinovitch, Gennadi ; Wiese, David: Non-linear Optimization of Performance Functions for Autonomic Database Performance Tuning. In: *Proceedings of the Third International Conference on Autonomic and Autonomous Systems.* Washington, DC, USA : IEEE Computer Society, 2007.

- Rabinovitch, Gennadi ; Wiese, David ; Reichert, Michael ; Arenswald, Stephan: Datenbank-Tuning mit dem Autonomic Tuning Expert. In: *Datenbank-Spektrum* Heft 27 (2008), Dezember, S. 18–26
- Rabinovitch, Gennadi: Policy-Based Coordination of Best-Practice Oriented Autonomic Database Tuning. In: *Proceedings of the 2009 Computation World: Future Computing, Service Computation, Cognitive, Adaptive, Content, Patterns*. Washington, DC, USA : IEEE Computer Society, 2009, S. 55–60
- Rabinovitch, Gennadi ; Wiese, David ; Reichert, Michael ; Arenswald, Stephan: *Workflow- und Workload-orientiertes Datenbank-Tuning*. Forschungsbericht. Lehrstuhl für Datenbanken und Informationssysteme. Friedrich-Schiller-Universität Jena, November 2010
- Reichert, Michael ; Arenswald, Stephan ; Wiese, David ; Rabinovitch, Gennadi: *Workload Aware Exception Detection*. Patent Proposal, September 2008
- Wiese, David ; Rabinovitch, Gennadi: A Generic Knowledge Model for Autonomic Database Performance Tuning. Poster. In: *4. Konferenz zum Professionellen Wissensmanagement: Erfahrungen und Visionen*. Potsdam, März 2007
- Wiese, David ; Rabinovitch, Gennadi: *DB2 Autonomic Performance Management*. IBM CAS Technical Report. IBM Confidential, Juni 2007
- Wiese, David ; Rabinovitch, Gennadi: *Autonomie ohne Aufpreis – Selbstverwaltende Datenbankmanagementsysteme*. Fachgruppentreffen der GI-Fachgruppe Datenbanksysteme: Selbstoptimierende und autonome Datenbanksysteme. Vortrag. Darmstadt, November 2007
- Wiese, David ; Rabinovitch, Gennadi ; Reichert, Michael ; Arenswald, Stephan: Autonomic Tuning Expert: A Framework for Best-practice oriented Autonomic Database Tuning. In: *Proceedings of the 2008 Conference of the Center for Advanced Studies on Collaborative Research: Meeting of Minds (CASCON 2008)*. New York, NY, USA : ACM, 2008, 3:27–3:41
- Wiese, David ; Rabinovitch, Gennadi ; Reichert, Michael ; Arenswald, Stephan: ATE: Workload-oriented DB2 Tuning in Action. Demo/Poster. In: *Tagungsband zur 13. GI-Fachtagung Datenbanksysteme für Business, Technologie und Web (BTW) 2009*. Münster, März 2009
- Wiese, David ; Rabinovitch, Gennadi: Knowledge Management in Autonomic Database Performance Tuning. In: *Proceedings of the Fifth International Conference on Autonomic and Autonomous Systems*. Washington, DC, USA : IEEE Computer Society, April 2009. S. 129–134

Kapitel 2

Ausgewählte Grundlagen

Dieses Kapitel fasst die für weitere Betrachtungen im Rahmen dieser Arbeit notwendigen und hilfreichen Grundlagen zusammen. Die Auswahl der hier vorgestellten Begriffe und Konzepte beschränkt sich dabei auf Themenbereiche wie das Performance-Tuning, das für zahlreiche Beispiele im Verlauf der vorliegenden Arbeit gewählte Datenbank-Management-System *IBM DB2*, das *Autonomic Computing* und das, als ein zentrales Ziel dieser Arbeit anzusehende, autonome Datenbank-Tuning. Dabei werden sowohl etablierte Vorgehensweisen zur Umsetzung autonomer Funktionalitäten vorgestellt und voneinander abgegrenzt, als auch unser best-practices-orientierter Ansatz beschrieben, der die Basis für die nachfolgenden Kapitel bildet.

2.1 Terminologie

2.1.1 Performance-Tuning

Ressourcen. Typischerweise erstrecken sich datenbankbasierte IT-Systeme über mehrere Schichten im sogenannten Hard- und Software-Stack. Bei diesen Schichten handelt es sich hauptsächlich um die Hardware, das darauf laufende Betriebssystem, ein darauf operierendes Datenbank-Management-System und schließlich einen Anwendungsserver sowie gegebenenfalls einen Webserver, die den Nutzern den Zugriff auf die Anwendungen bereitstellen. Dabei stellt jede Schicht ihre *Ressourcen* der darüber liegenden Schicht mittels einer Schnittstelle zur Verfügung. Der Begriff Ressource ist hier weit gefasst. So fallen darunter sowohl physische Hardware-Ressourcen, wie CPU, Hauptspeicher, Festplatte und Netzwerk, als auch die darauf aufbauenden logischen Ressourcen, wie Dateien, Puffer, Prozesse, Services bzw. Datenbankobjekte wie beispielsweise Tabellen oder Indizes sowie andere abstrakte Objekte, die auf der jeweiligen Schicht zur Verfügung stehen.

Ressourcen-Schnittstellen: Sensoren, Effektoren. Jeder Ressourcen-Typ verfügt dabei typischerweise über zwei Arten von Schnittstellen: Die *Effektoren-Schnittstelle* ist für einen schreibenden Zugriff auf Ressourcen-Informationen verantwortlich. Typischerweise können damit neue Instanzen des Ressourcen-Typs erzeugt, entfernt oder konfiguriert werden (z. B. Erstellen einer Tabelle, Entfernen eines Index bzw. Veränderung der Größe eines Bufferpools). Die *Sensoren-Schnittstelle* ist hingegen für einen lesenden Zugriff auf Ressourcen-spezifische Beschreibungsinformationen (Metadaten) verantwortlich. Im Allgemeinen besitzt jeder Ressourcen-Typ Attribute, die über Sensoren ausgelesen werden können. Diese spiegeln dabei entweder den

aktuellen Zustand oder vergangene Zustände im Zeitverlauf einer oder mehrerer Ressourcen-Instanzen wider. Durch Sensoren können Informationen auf verschiedenen Ebenen geliefert werden. So kann beispielsweise der Sensor „Bufferpool Hitratio“ diese entweder für einen spezifischen Bufferpool oder aber auch aggregiert für eine bestimmte Menge an Bufferpools angeben.

Performance. Die *Performance* bezeichnet die Art und Weise, wie ein System unter einer bestimmten Auslastung und gegebenen Anforderungen arbeitet. Sie wird anhand von Performance-Maßzahlen (*Performance-Metriken*) gemessen. Die Performance eines DBMS kann beispielsweise durch Metriken wie Antwortzeit, Durchsatz oder Verfügbarkeit gemessen werden und wird von den im DBMS vorhandenen Ressourcen, deren Auslastung und Ausmaß der gemeinsamen Nutzung beeinflusst.

Performance-Problem. Ein *Performance-Problem* ist gekennzeichnet durch die Abweichung des durch eine einzelne Performance-Metrik oder Kombination mehrerer Performance-Metriken erfassten Ist-Zustands von einem geforderten Soll-Zustand.

Richtlinien. Die Definition von Soll-Zuständen erfolgt typischerweise durch die Angabe von speziellen *Richtlinien (Service Level Agreements, SLA)*. Dabei handelt es sich um Performance-Ziele bzw. Anforderungen an das System und seine Performance-Metriken.

Performance-Tuning. Als *Performance-Tuning (Tuning)* wird das Vorgehen bezeichnet, welches die Erreichung der angestrebten System-Performance ermöglicht. Dabei kann versucht werden, das System in einem vorab definierten Soll-Zustand zu halten oder neue Zielvorgaben durch entsprechende Maßnahmen zu erreichen. Die Tuning-Vorgehensweise ist in der Regel ein iterativer Prozess, der auf dem Erfahrungsschatz des jeweiligen Administrators basiert. Zwar hat jeder Administrator seine eigene Herangehensweise, jedoch haben diese Herangehensweisen gewisse Ähnlichkeiten. So wird typischerweise regelmäßig geprüft, ob die System-Performance (noch) den vorgegebenen Richtlinien entspricht. Ist dies nicht der Fall, so wird versucht, die Ursachen für diese Abweichung zu diagnostizieren. Wurden einmal die Problemursachen identifiziert, so können als nächstes Maßnahmen geplant und eingeleitet werden, die die Problemursachen und somit auch das Problem beheben. Dieser Ablauf entspricht im Wesentlichen einer rückgekoppelten Kontrollschleife (*Abschnitt 2.2*), die als Grundlage für autonomes Tuning von IT-Systemen verwendet werden kann und in unserem Ansatz (*Abschnitt 2.3.2*) Anwendung findet.

2.1.2 Das Datenbank-Management-System *IBM DB2*

Die im Rahmen dieser Arbeit erarbeiteten Konzepte und Architekturen sind weitgehend universell und Produkt-unabhängig, dennoch musste die Entscheidung für ein bestimmtes DBMS getroffen werden, um die vorgestellten Vorgehensweisen anhand von konkreten Beispielen zu veranschaulichen. Da auch der in einem Kooperationsprojekt mit IBM entwickelte und in dieser Arbeit vorgestellte Prototyp auf *IBM DB2* aufbaut, wurde DB2 als das Datenbank-Management-System für die meisten DBMS-bezogenen Beispiele gewählt. Nachfolgend erfolgt eine kurze Vorstellung ausgewählter Aspekte von DB2, die beim Verständnis der in den nachfolgenden Kapiteln enthaltenen DBMS-nahen Ausführungen behilflich sein werden. Die Ausführungen basieren dabei primär auf [IBM06d, IBM06g,

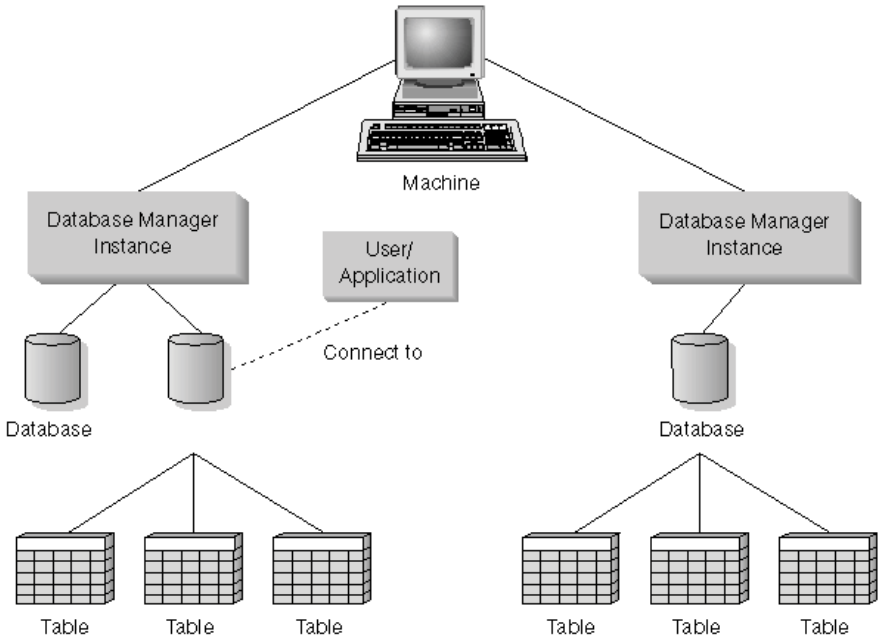


Abbildung 2.1: Hierarchie ausgewählter DB2-Datenbankobjekte [IBM04b]

IBM06f]. An Stellen, wo ein tieferes Verständnis ausgewählter DB2-Konzepte vorausgesetzt wird, werden vorab die notwendigen Informationen zusammenfassend dargestellt.

IBM DB2 for Linux, Unix and Windows (DB2 LUW) ist ein komplexes und weit verbreitetes relationales Datenbank-System der Firma *IBM*, das den aktuellen ANSI-SQL-Standard weitgehend abdeckt und in vielen großen geschäftskritischen Anwendungen (unter anderem auch als eine Basis der SAP-Anwendungswelt) eingesetzt wird.

System, Instanz. Die Architektur eines DB2-LUW-Datenbank-Systems erlaubt den Betrieb und die Verwaltung von mehreren DB2-Datenbanken. Diese ordnen sich in die Hierarchie der DB2-Datenbankobjekte ein [IBM06d]. Auf der obersten Ebene dieser Hierarchie findet sich das DB2-System, welches einer DB2-Installation entspricht. Einem DB2-System können mehrere *DB2-Instanzen* zugeordnet werden. Diese sind jeweils für die Verwaltung mehrerer ihnen zugeordneter Datenbanken zuständig und sind daher auch als *Datenbankmanager (DBM)* bekannt (***Abbildung 2.1***). Sie stellen logische Datenbankserverumgebungen dar.

Die Einrichtung mehrerer Instanzen bietet einerseits Vorteile, wie beispielsweise Trennung der Entwicklungsumgebung von der Geschäftsumgebung, Begrenzen des Zugriffs auf sensible Daten, Optimieren der Datenbankmanager-Konfiguration für jede Instanz oder Begrenzen der durch den Ausfall einer Instanz entstehenden Auswirkungen durch Verteilung der Daten auf mehrere Instanzen. Andererseits sind die für jede Instanz zusätzlich benötigten Systemressourcen sowie der Verwaltungsaufwand nicht zu vernachlässigen.

Datenbank, Tabelle, Tablespace. Einer Datenbankinstanz können wiederum mehrere *Datenbanken (DB)* zugeordnet werden, mit denen sich Nutzer oder Anwendungsprogramme direkt verbinden und ihre Anfragen absetzen können (*Abbildung 2.1*).

Eine Datenbank selbst besteht aus einer Menge von Datenbank-Objekten, wie beispielsweise *Tabellen* und *Sichten*, die sich logisch in *Schemas* gruppieren lassen. Die Informationen über die Datenbank selbst (Metadaten) sind dabei im *Datenbank-Katalog* abgelegt.

Jede Tabelle wird einem *Tabellenbereich* (*Tablespace*) zugeordnet, welcher für die physische Speicherung der Daten zuständig ist, und wiederum aus mehreren *Containern* bestehen kann. Ein Container kann dabei eine Datei, ein Verzeichnis oder eine ganze Festplatte sein.

Bufferpool, Bufferpool Hitratio. Die Anzahl der Zugriffe auf die (im Vergleich zum Arbeitsspeicher langsame) Festplatte wird durch die Einführung von *Bufferpools* reduziert. Diese werden den Tablespaces zugeordnet und dienen als Puffer für Eingabe-/Ausgabe-Operationen.

In welchem Maße ein Bufferpool genutzt wird (welchen quantitativen Nutzen er also bringt), lässt sich durch die *Bufferpool Hitratio* ausdrücken. Sie beschreibt das Verhältnis von der Anzahl logischer (über Cache) und physischer (über Sekundärspeicher) Datenbankzugriffe und ist damit ein verlässliches Maß für die Ausführungs-Performance von I/O-dominierten Anfragen.

Package, Package Cache, Package Cache Hitratio. Als *Paket* (*Package*) wird ein Objekt bezeichnet, das in einer Datenbank gespeichert ist und Informationen, wie beispielsweise Zugriffspläne, enthält, die zur Verarbeitung der einer bestimmten Quelldatei eines Anwendungsprogramms zugeordneten SQL-Anweisungen benötigt werden.

Durch das Caching von Paketen wird es dem Datenbankmanager ermöglicht, den internen Systemaufwand zu verringern, da beim erneuten Laden eines Pakets kein Zugriff auf die Systemkataloge bzw. bei dynamischen SQL- oder XQuery-Anweisungen keine Kompilierung und Optimierung mehr erforderlich ist. Dadurch kann insbesondere die wiederholte Ausführung vieler gleicher bzw. ähnlicher Anweisungen (z. B. mehrere Insert-Anweisungen, die sich nur durch die einzufügenden Werte unterscheiden) beschleunigt werden.

Die Performance eines *Paket-Cache* kann mit Hilfe der *Package Cache Hitratio* gemessen werden. Diese gibt an, wie oft beim Neuladen von Paketen der Zugriff auf Systemkatalogtabellen für statische SQL-Anweisungen bzw. das Neukompilieren und -optimieren von dynamischen SQL-Anweisungen vermieden werden konnten, weil die Daten sich noch im Paket-Cache befanden.

Datenbankagent. *Datenbankagenten* sind spezielle Prozesse, die die Kommunikation zwischen Datenbankmanager und Clients verwalten und insbesondere den größten Teil der SQL-Verarbeitung im Auftrag von Anwendungen ausführen. Die Anfragen werden dabei bei Bedarf von koordinierenden Agenten in mehrere Teile aufgeteilt und an Arbeitsagenten zur weiteren Verarbeitung übergeben.

Für weiterführende Informationen zu DB2-Konzepten sei an dieser Stelle auf [IBM06g, IBM06d] verwiesen.

2.2 Autonomic Computing

Durch die in den letzten Jahrzehnten stattgefundene Weiterentwicklung der Hardware und somit eine steigende Leistungsfähigkeit der IT-Systeme erkannte die Industrie das Potenzial, immer mehr Aufgaben an die IT-Systeme übertragen zu können, was jedoch zu einer zunehmenden Komplexität der Software führte. Diese Komplexität ist ein schwerwiegendes Problem der heutigen Informationstechnologie [IBM01]. Nicht nur die Bedienung heutiger IT-Systeme, sondern insbesondere auch ihre Administration sind zunehmend schwieriger geworden. Während das Problem der komplexen Bedienung durch Verbesserung der Benutzerschnittstelle sowie Schulung des Personals adressiert werden kann, erfordert die Lösung des Administrationsproblems neuartige Ansätze: Zum einen lassen sich Werkzeuge zur Unterstützung der Administratoren entwickeln. Das wirkt zwar der Komplexität der Administration entgegen, reduziert jedoch nur bedingt die typischen Probleme der manuellen Administration, wie die Fehleranfälligkeit, fehlende Kompetenz der Administratoren, zeitliche Verzögerungen nach Problemauftreten bis zur Lösung etc. Zum anderen kann das Problem auch durch tiefgreifende Systemveränderungen adressiert werden. So können IT-Systeme angepasst bzw. um Komponenten erweitert werden, die eine automatisierte Systemüberwachung und -verwaltung ermöglichen.

Diesem Ziel haben sich Ansätze wie das *Autonomic Computing* – eine im Oktober 2001 von *IBM* ins Leben gerufene Initiative – verschrieben. Sie sollen helfen, das Problem der steigenden Komplexität zu adressieren, indem sie Technik nutzen, um Technik zu verwalten [IBM06a]. Neben IBM sind mittlerweile auch andere große Softwarehersteller aktiv im Bereich des *Autonomic Computing*. So ist beispielsweise die *Dynamic Systems Initiative (DSI)* von *Microsoft* und seinen Partnern zu nennen, die das Ziel verfolgt, IT-Teams bei der Entwicklung leichter zu verwaltender Systeme und der Automatisierung sich wiederholender Vorgänge zu unterstützen. Es ist ein Baustein des Versprechens von Microsoft, sich selbst verwaltende dynamische Systeme zu entwickeln [Mic11c]. Auch *Hewlett-Packard* bietet mit seiner *Adaptive Infrastructure* einen Ansatz zur Entwicklung von autonomen Systemen [Hew07]. *Intel* ist seinerseits aktiv bei der Entwicklung von Standards für das *Autonomic Computing* [TM06].

Die Idee von autonom agierenden Systemen ist jedoch nicht neu. In der Regelungs- und Automatisierungstechnik sind bereits länger Systeme bekannt, die durch die Umsetzung eines sogenannten Regelkreises den Zustand des überwachten Systems (Ist-Zustand) kontrollieren und bei starken Abweichungen vom Soll-Zustand entsprechende Funktionen einleiten, um das vorgegebene Ziel unter Berücksichtigung der Randbedingungen zu erfüllen [DHP+05]. Bei der Übertragung dieses Ansatzes auf IT-Systeme kann somit von den Forschungsergebnissen und Erfahrungen auf dem Gebiet der Automatisierungstechnik profitiert werden.

Nachfolgend sollen sowohl die Ziele des *Autonomic Computing* als auch die Anforderungen, die von IT-Systemen auf dem Weg zur Selbstverwaltung zu erfüllen sind, vorgestellt werden. Anschließend wird ein kurzer Einblick in die von IBM konzipierte Referenzarchitektur des *Autonomic Computing* gegeben.

2.2.1 Die Vision des *Autonomic Computing*

In Anlehnung an das vegetative Nervensystem des Menschen sollen IT-Systeme in die Lage versetzt werden, sich ohne Eingriffe der Administratoren an ihre Umgebung anzupassen. Durch das vegetative Nervensystem werden menschliche Körperfunktionen gesteuert und an die Veränderungen der Umweltbedingungen angepasst. Eine autonome Regelung lebensnotwendiger Funktionen wird dabei durch eine kontinuierliche Überwachung der Indikatoren wie Atemfrequenz, Herzschlag, Blutdruck bzw. Blutzuckerspiegel und eine Reaktion auf ihre Veränderungen ermöglicht.

Nach [IBM01] sollen auch IT-Systeme Routineaufgaben selbst übernehmen, die Funktionalität ihrer Komponenten überwachen und bei erkannten Problemen Gegenmaßnahmen einleiten können. Dieses Verhalten soll dabei weitgehend ohne Unterstützung der Administratoren ermöglicht werden. Durch ihre Entlastung von repetitiven Routineaufgaben sollen sich die Administratoren auf die langfristige, strategische Planung und Koordination von IT-Systemen konzentrieren können.

Im Gegensatz zum vegetativen Nervensystem, welches die Verantwortung für alle Lebensfunktionen übernimmt, können die Administratoren jedoch nicht nur entscheiden, welche Funktionen an die IT-Systeme delegiert werden, sondern auch Richtlinien angeben, nach welchen die IT-Systeme agieren sollen [IBM06a]. Hierauf wird in *Kapitel 6* näher eingegangen.

2.2.2 Anforderungen an selbstverwaltende Systeme

Der Schlüssel zu einem autonomen System ist die Selbstverwaltung. Ein selbstverwaltendes System soll nach [IBM01] den folgenden acht Anforderungen genügen:

Systemidentität. Ein autonomes System kann sich über mehrere Schichten seiner Umgebung erstrecken und aus heterogenen Komponenten zusammengesetzt sein. Zur Ermöglichung von autonomen Fähigkeiten benötigt das System daher ein detailliertes Wissen über die einzelnen Komponenten des Systems, ihren Status, Kapazitäten sowie Schnittstellen zu umgebenden Systemen. Die Herausforderung, diese Anforderung zu erfüllen, steigt mit zunehmender Komplexität und Heterogenität von IT-Systemen.

Selbst(re)konfiguration. Ein autonom agierendes System muss in der Lage sein, sich automatisch an die sich ändernden und unvorhersehbaren Umweltbedingungen wie beispielsweise Lastprofile anzupassen. Die Konfiguration von Systemen ergibt sich durch die Konfigurationen ihrer einzelnen Komponenten. Viele Komponenten haben hunderte von Konfigurationsalternativen, was in einer hohen Anzahl von möglichen Konfigurationen des Zielsystems resultiert. Ein menschlicher Administrator ist somit nicht mehr in der Lage, das System permanent zu überwachen und die optimalen Konfigurationseinstellungen zur Laufzeit zu bestimmen. Ein autonomes System soll also durch kontinuierliches Überwachen seiner Komponenten die optimale Konfiguration selbst bestimmen und gewährleisten können.

Kontinuierliche Optimierung. Ein autonomes System soll ständig nach Optimierungsmöglichkeiten suchen, um die von außen vorgegebenen (und unter Umständen einander widersprechenden) Ziele zu erreichen. Aufgrund einer Dynamik in der Systemumgebung und der möglichen Veränderungen der Ziele ist der Optimierungsprozess potentiell unendlich. Zur Umsetzung der kontinuierlichen (Selbst-)Optimierung kann die aus der Regelungstechnik bekannte rückgekoppelte Kontrollschleife eingesetzt werden. Die Übertragung des Verfahrens auf die Welt der Computersysteme ist jedoch nicht trivial und wirft viele Fragen auf. So muss beispielsweise geklärt werden, wie oft die Ausführung der Problem-korrigierenden Aktionen stattfinden soll, wie lang die Verzögerung zwischen diesen Aktionen und der Sichtbarkeit ihrer Wirkung sein darf und welchen Einfluss die ausgeführten Aktionen auf die Stabilität des Systems haben.

Selbstheilung. Das System muss auf Fehler geeignet reagieren können. Diese müssen zum einen zeitnah erkannt werden, zum anderen muss das System eine alternative Konfiguration bzw. Ressourcenbelegung bestimmen können, um den Fehler zu beheben bzw. nach Außen hin zu maskieren. Zur Beseitigung eines aufgetretenen Problems muss zuerst jedoch die Problemursache lokalisiert werden. Dies ist bei den heutigen komplexen Systemlandschaften nur schwer manuell möglich.

Selbstschutz. Um Systemsicherheit und -integrität bieten zu können, muss das System in der Lage sein, Angriffe bzw. Bedrohungen zu erkennen und sich davor zu schützen. Zwar sind heutige Mechanismen erfolgreich beispielsweise in der Erkennung von verdächtigen Code-Fragmenten, jedoch müssen diese anschließend von Experten manuell auf ihre Gefährlichkeit überprüft werden. Der Bedarf nach weiterer Automatisierung dieses Vorgehens steigt mit zunehmender Vernetzung und Komplexität von IT-Systemen.

Kenntnis der Umgebung und des Kontextes. Ein autonomes System benötigt detaillierte Informationen über seine Umgebung sowie den Kontext aller seiner Aktivitäten, um dies zur Laufzeit entsprechend zu berücksichtigen. Zur Erfüllung dieser Anforderung werden Verfahren benötigt, die es einerseits ermöglichen, Systeme sowie ihre Ressourcen zu beschreiben und diese Informationen den umgebenden Systemen zur Verfügung zu stellen. Andererseits müssen Systeme in der Lage sein, die von anderen Systemen bereitgestellten Informationen aufzunehmen und zu verarbeiten. Dabei kann unter anderem von den bereits existierenden Ansätzen im Bereich des *Grid Computing* profitiert werden.

Existenz in heterogenen Systemen. Wie bereits aus der Beschreibung der vorherigen Anforderungen sichtbar geworden ist, kann ein autonomes System per Definition nicht in einer abgeschlossenen Umgebung existieren. Es agiert notwendigerweise in einer heterogenen Systemlandschaft, deren Komplexität mit Entwicklung neuer Technologien weiter steigt.

Verbergen der Systemkomplexität nach Außen. Die zur Erfüllung der Anforderungen nötige Komplexität des Systems soll den Endnutzern verborgen bleiben.

Die acht zuvor genannten Anforderungen beschreiben zugleich die Eigenschaften eines autonomen Systems. Nach [KC03] lassen sie sich auf folgende vier Schlüsseleigenschaften reduzieren:

Selbstkonfiguration (*Self-Configuration*). Das System reagiert selbstständig auf Änderungen der IT-Umgebung und passt sich entsprechend an.

Selbstheilung (*Self-Healing*). Das System erkennt automatisch fehlerhafte oder problematische Abläufe und reagiert auf diese, ohne dabei den normalen Betrieb zu unterbrechen.

Selbstoptimierung (*Self-Optimization*). Das System ist in der Lage, seine Ressourcen weitestgehend selbst zu verwalten und so die Performance auf einem möglichst hohen Level zu halten.

Selbstschutz (*Self-Protection*). Das System steuert selbstständig die Datensicherheit, das heißt, dass nicht autorisierte Zugriffe oder sogar Angriffe erkannt und entsprechend behandelt werden.

Diese Merkmale sind auch als *CHOP*-Eigenschaften bzw. *Self**-Eigenschaften bekannt. Im Kontext der autonomen Datenbank-Management-Systeme lassen sich die vier zuvor genannten Merkmale um zwei weitere, und zwar die Selbstorganisation (*self-organizing*) und Selbstkontrolle (*self-inspecting*), erweitern [EPBM03].

2.2.3 Der Weg zur Autonomie

Die Umsetzung eines komplett selbstverwaltenden Systems ist ein langwieriger Prozess. Dabei kann grundsätzlich zwischen zwei Herangehensweisen unterschieden werden: Zum einen lässt sich ein solches System neu entwickeln, was jedoch typischerweise nicht nur hohe Kosten bei der Entwicklung, sondern auch bei der Einführung des neuen Systems verursacht. Alternativ lässt sich ein bestehendes System schrittweise um entsprechende Komponenten erweitern, die die Umsetzung einzelner autonomer Mechanismen übernehmen. Diesen Ansatz verfolgt auch IBM bei der Umsetzung von autonomen Systemen und spricht in dem Zusammenhang von der Evolution von Computersystemen [GC03]. Der Weg von einem manuell administrierten zu einem vollständig autonom agierenden System ist dabei lang und lässt sich nach [IBM03a] in einzelne Evolutionsstufen unterteilen (***Abbildung 2.2***). Je nach erreichtem Grad ihrer Autonomie-Eigenschaften können IT-Systeme diesen Stufen zugeordnet werden. Diese fünf Stufen auf dem Weg zu vollständig autonom agierenden Systemen werden im Folgenden kurz skizziert:

Basic. Systeme, die dieser Stufe zugeordnet sind, verfügen über keinerlei autonome Mechanismen. Ihre Administration wird ausschließlich manuell durch entsprechend qualifiziertes Personal durchgeführt.

Managed. Das Management der Systeme, die der Stufe „*Managed*“ zugeordnet sind, wird bereits durch einfache, selbstständig agierende Werkzeuge zur Datensammlung unterstützt. Auch einfache Administrationsaufgaben werden bei solchen Systemen durch entsprechende Automatisierungswerkzeuge ermöglicht. Die Problemanalyse sowie die Entscheidung, welche Aktionen durchzuführen sind, sind dennoch vom Fachpersonal durchzuführen.

Predictive. Systeme dieser Stufe verfügen über einzelne Komponenten, die sich selbst überwachen, Änderungen analysieren und Verbesserungsvorschläge unterbreiten. Die

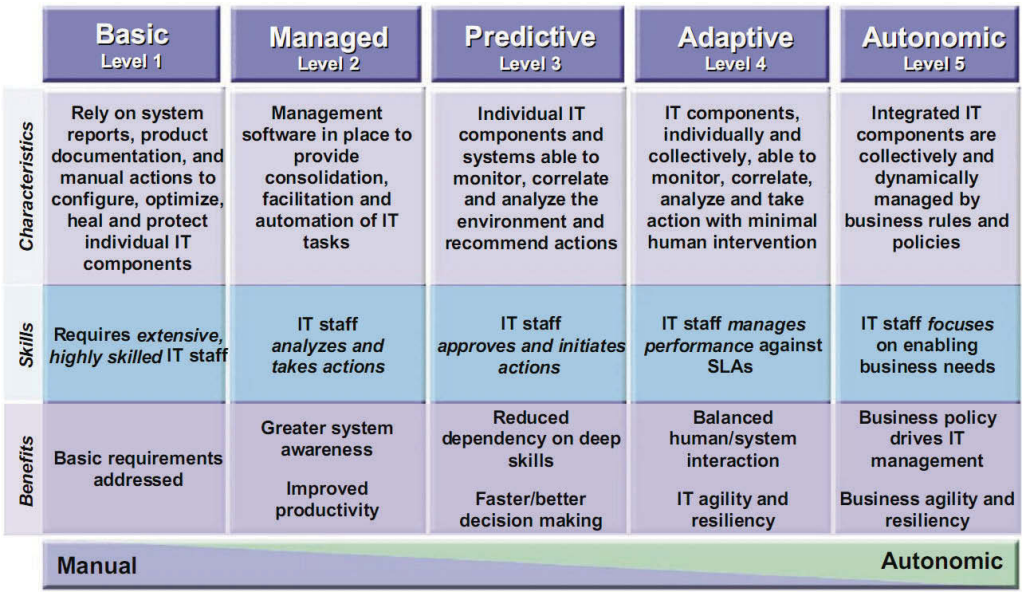


Abbildung 2.2: Der Weg zur Autonomie – *IBM Maturity Model* [IBM03a]

Entscheidung, ob die Verbesserungsvorschläge angenommen werden oder nicht, wird jedoch auch hier von Menschen getroffen.

Adaptive. Auf dieser Stufe wird nun sowohl die Überwachung als auch die Analyse nicht mehr pro Komponente, sondern komponentenübergreifend durchgeführt. Auch die vom System unterbreiteten Verbesserungsvorschläge beziehen sich nicht mehr isoliert auf einzelne Komponenten, sondern auf das gesamte System. Der Grad der notwendigen Interaktion mit Administratoren wird weiter reduziert.

Autonomic. Auf der letzten Stufe auf dem Weg zu vollständig autonomen Systemen erfolgt die Steuerung der durch das System durchgeführten Überwachungs-, Analyse- sowie Planungsaktivitäten im Gegensatz zur „*Adaptive*“-Stufe durch Richtlinien, die von Administratoren bereitgestellt werden.

Bei der Betrachtung der soeben erklärten Stufen nimmt also mit zunehmender Stufe der Automatisierungsgrad zu und der Bedarf an manueller Intervention ab. Aus heutiger Sicht liegen zwar komplett autonom agierende IT-Systeme in ferner Zukunft, durch eine schrittweise Integration autonomer Funktionalitäten in Subkomponenten befindet sich jedoch eine Vielzahl von Softwareprodukten bereits auf dem Weg dahin. Grundsätzlich lassen sich die meisten heutigen Systeme zwischen den Stufen „*Managed*“ und „*Predictive*“ einordnen. Einzelne Prototypen aus der Forschung sind jedoch bereits viel weiter und reichen je nach ihren Anwendungsgebieten bis hin zur Stufe „*Autonomic*“.

2.2.4 Eine Referenzarchitektur für das *Autonomic Computing*

Das Ziel von autonomen Systemen ist die Reduktion der Betriebskosten (*Total Cost of Ownership, TCO*) von komplexen IT-Systemen durch die Verschiebung des Verwaltungsaufwands an die Systeme selbst. Ein vielversprechender Ansatz bei der Konzeption und

Umsetzung von solchen autonomen Systemen ist die Verwendung einer *rückgekoppelten Kontrollschleife* (*Feedback Control Loop*).

Eine rückgekoppelte Kontrollschleife durchläuft kontinuierlich vier aufeinander folgende Phasen: *Collect*, *Analyze*, *Decide* und *Act* [DDF+06]. In der Datensammlungsphase (*Collect*) erfolgt die Sammlung der den Zustand des überwachten Systems repräsentierenden Daten. Die gesammelten Informationen wie beispielsweise Messergebnisse werden anschließend bereinigt, gefiltert und abgelegt für weitere Auswertungen. In der anschließenden Analyse-Phase (*Analyze*) werden die gesammelten Daten ausgewertet und Trends bzw. auf Probleme hindeutende Symptome identifiziert. Die darauf aufbauende Entscheidungsphase (*Decide*) ist für die Generierung von Strategien zur Lösung von bestehenden bzw. Vermeidung von künftigen Problemen verantwortlich. Diese Strategien werden anschließend in der Ausführungsphase (*Act*) ausgeführt und ihre Wirkung in der nachfolgenden *Collect*-Phase überprüft. Die gesammelten Informationen werden wiederum ausgewertet und als Grundlage für neue Strategien verwendet.

Auch der in [IBM06a] vorgestellte Architekturentwurf für autonome Computersysteme basiert auf einer rückgekoppelten Kontrollschleife. In diesem Ansatz bilden *Autonomic Manager* den zentralen Baustein autonomer Systeme. *Autonomic Manager* werden hier jeder einzelnen Ressource zugeordnet und sind für ihre Optimierung unter Berücksichtigung vorgegebener Nebenbedingungen verantwortlich. Bei den mittels *Autonomic Manager* verwalteten Ressourcen kann es sich einerseits um Hard- bzw. Software-Ressourcen unterschiedlicher Granularität (z. B. Netzwerkkarten, Datenbanken, Datenbanktabellen bzw. Datenbankseiten) handeln. Andererseits können *Autonomic Manager* für die Koordination der Funktionsweise weiterer *Autonomic Manager* verantwortlich sein. Um die Interaktion eines *Autonomic Manager* mit der zugehörigen Ressource zu ermöglichen, verfügt jede Ressource über zwei Arten von Schnittstellen: eine *Sensor*- sowie eine *Effektorschnittstelle*. Sensoren liefern Informationen über den aktuellen Zustand der überwachten Ressource und Effektoren bieten die Möglichkeit, die Konfiguration und somit den Zustand dieser Ressource zu ändern.

Jeder *Autonomic Manager* implementiert seinerseits eine rückgekoppelte Kontrollschleife, die kontinuierlich folgende Phasen durchläuft: Überwachung der Ressource, Analyse der gesammelten Daten, Planung von Optimierungsaktionen und Ausführung der geplanten Aktionen. Nach [IBM06a] werden diese Phasen als *Monitor*, *Analyze*, *Plan* und *Execute* (*MAPE*) bezeichnet. Daher spricht man auch von der *MAPE-Schleife*. Die einzelnen MAPE-Phasen entsprechen dabei den oben erwähnten *Collect*-, *Analyze*-, *Decide*- und *Act*-Phasen.

Die im Folgenden dargestellten vier Phasen der MAPE-Schleife bilden einen (potentiell endlosen) iterativen Prozess (***Abbildung 2.3***). Die Phasen können dabei durch einzelne Komponenten realisiert werden, die aufeinander aufbauen und untereinander Informationen austauschen (vgl. [Rab06]).

Monitor. Die *Monitor*-Komponente überwacht die Ressourcen über die Sensoren und sammelt Daten, die je nach Anwendung detaillierte Informationen über die Ressourcenkonfiguration, -status, -kapazität und -durchsatz sowie Topologieinformationen, Metriken und andere enthalten können. Diese Daten werden vom *Monitor* gespeichert, aggregiert, korreliert und gefiltert, bis die die Gesundheit des Systems gefähr-

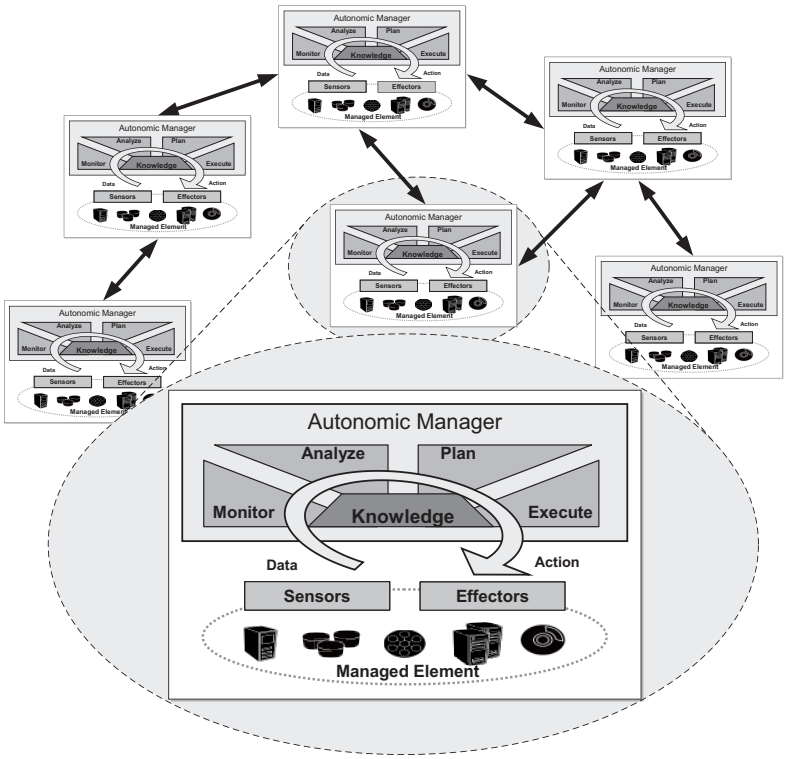


Abbildung 2.3: Miteinander interagierende *Autonomic Manager* (in Anlehnung an [KC03])

denden Symptome identifiziert werden können, die anschließend von der *Analyze*-Komponente ausgewertet werden.

Analyze. Der *Analyzer* bietet Mechanismen, um Symptome zu korrelieren und die über die möglichen Probleme getroffenen Annahmen zu überprüfen. Mit Hilfe dieser Komponente kann das System seine Umgebung erfassen, um Vorhersagen über das künftige Systemverhalten zu treffen.

Plan. Die Planungsfunktionalität ermittelt die Aktionen, die zum Erreichen der vorgegebenen Zielrichtlinien ausgeführt werden müssen. Bei der Planung werden die Systemkomponenten mit allen Abhängigkeiten in Betrachtungen einbezogen, um das System stabil zu halten und Oszillationen[1] zu vermeiden.

Execute. Die in der Plannungsphase ermittelten Aktionsabfolgen werden anschließend durch die *Execute*-Komponente mit Hilfe von Effektoren der Ressourcen ausgeführt. Die somit verursachten Systemänderungen werden wiederum durch den *Monitor* überwacht, der Kreis schließt sich damit.

Jeder *Autonomic Manager* muss weiterhin über das Wissen (*Knowledge*) über die Anforderungen an das System, über seine Infrastruktur sowie über die Abhängigkeiten unter den einzelnen Komponenten verfügen. Erst durch dieses Wissen wird eine selbstständige

[1] Insbesondere soll eine Oszillation zwischen zwei suboptimalen Zuständen vermieden werden.

Verwaltung von Ressourcen durch die *Autonomic Manager* ermöglicht. Das Wissen kann nach [Mil05] in drei Bereiche eingeteilt werden[2]:

- das Wissen über die Topologien,
- das Wissen über die Richtlinien und
- das Wissen, welches zur Lösung von identifizierten Problemen benötigt wird.

Diese Informationen können entweder durch die Systemkomponenten gesammelt, angereichert bzw. generiert oder aber auch von außen, beispielsweise durch Administratoren, vorgegeben werden. Dabei können zwar einzelne Informationen lediglich für einen einzelnen *Autonomic Manager* interessant sein, in der Regel handelt es sich bei diesen Informationen jedoch um ein für alle *Autonomic Manager* wichtiges Wissen, welches diesen entsprechend zur Verfügung gestellt werden muss.

Grundsätzlich ist jeder *Autonomic Manager* ausschließlich für das Tuning einer ihm zugeordneten Ressource zuständig. Die Betrachtung der Probleme einzelner Ressourcen erfolgt daher durch *Autonomic Manager* lokal. Zur Ermöglichung einer ressourcenübergreifenden Problemerkennung und -auflösung muss eine Kommunikation unter den *Autonomic Managern* gewährleistet werden. Dies kann beispielsweise durch eine hierarchische Anordnung von *Autonomic Managern* und eine Verlagerung der ressourcenübergreifenden Planung nach oben in der Hierarchie bzw. durch die Zuordnung mehrerer *Autonomic Manager* zu einer *Peer Group* [BBC+03], die eine Kommunikation unter den dieser *Peer Group* zugeordneten, gleichberechtigten *Autonomic Managern* ermöglicht, umgesetzt werden (vgl. *Abbildung 2.3*).

2.3 Autonomes Datenbank-Tuning

Im vorangegangenen *Abschnitt 2.2* wurden die Ziele des *Autonomic Computing* sowie die Konzepte, um diese Ziele zu erreichen, vorgestellt. Speziell im Bereich der autonomen Verwaltung von Datenbank-Management-Systemen finden viele Konzepte des Forschungsgebiets des *Autonomic Computing* Anwendung. Das Ziel ist dabei die Bereitstellung von Mechanismen und Werkzeugen, die die Administratoren im laufenden Betrieb entlasten und die fehleranfälligen Routine-Aufgaben übernehmen. Der Tätigkeitsschwerpunkt der Administratoren soll sich dabei idealerweise auf die Definition von Tuning-Zielen, die durch die autonomen Mechanismen zu erreichen sind sowie auf strategische, System-übergreifende Planung verlagern.

Nachfolgend wird in *Abschnitt 2.3.1* ein kurzer Überblick über die Ansätze gegeben, die entweder für die Umsetzung von autonomen Funktionalitäten im Kontext des Datenbank-Tuning konzipiert wurden oder sich auf das Datenbank-Tuning übertragen lassen. Anschließend wird in *Abschnitt 2.3.2* der von uns gewählte Ansatz des best-practices-orientierten Datenbank-Tuning vorgestellt und von anderen etablierten Ansätzen abgegrenzt.

[2] In *Abschnitt 5.1* wird ein Wissensmodell vorgestellt, welches neben diesen Wissensbereichen noch viele weitere abdeckt und dadurch das autonome Tuning von Datenbank-Systemen unterstützt.

2.3.1 Vorgehensweisen zur Umsetzung autonomer Funktionalitäten

Es existieren verschiedene Ansätze zur Umsetzung autonomer Systeme. Nach [Sul03] lassen sich die Ansätze in drei Kategorien einteilen:

- Techniken, die auf empirischen Vergleichen beruhen,
- Techniken, die auf analytischen Modellen beruhen und
- Feedback-Schleifen-basierte Techniken.

Nachfolgend werden sowohl die drei Kategorien kurz beschrieben als auch einige Forschungsinitiativen, die diesen Kategorien zugeordnet sind, exemplarisch vorgestellt. Die Ausführungen basieren dabei größtenteils auf [Sul03].

2.3.1.1 Auf empirischen Vergleichen beruhende Ansätze

Bei auf empirischen Vergleichen basierenden Techniken geht es darum, die optimale Konfiguration der einzelnen Tuning-Stellschrauben für eine vorgegebene Workload durch das Vergleichen der System-Performance-Zustände unter verschiedenen Konfigurationen zu ermitteln. Diesem Verfahren liegt die Annahme zugrunde, dass in endlicher Zeit alle möglichen Konfigurationen eines Systems „durchprobiert“ und die System-Performance unter diesen Konfigurationen gemessen werden können. Anschließend lässt sich für eine vorgegebene Workload die optimale Konfiguration ermitteln. Es fällt jedoch auf, dass dieses Verfahren bei komplexen Systemen mit einer großen Anzahl an möglichen Stellschrauben nicht praktikabel ist und mit sehr hohen Messzeiten verbunden ist.

Unterliegt das zu tunende System nur einer geringen Anzahl an Workload-Arten, die unter Umständen sogar im Voraus bekannt sind, so können die empirischen Vergleiche einmalig im Voraus durchgeführt werden, um so für jede unterscheidbare Workload-Art vorab die optimale Konfiguration zu ermitteln. Da die Workload in der Praxis jedoch in vielen Fällen permanenten Änderungen unterliegt, lässt sich eine aufwendige Neubestimmung der optimalen Konfiguration bei jeder substantiellen Änderung der anliegenden Workload nicht vermeiden. Es kann ferner sogar notwendig sein, neue Vergleiche für bereits untersuchte Workload-Arten durchzuführen, um die Konfiguration an die sich oft ändernde Systemumgebung anzupassen.

Es existiert eine Reihe an Forschungsansätzen, die dieser Kategorie zuzuordnen sind. So wird beispielsweise in [RP81] ein Ansatz zur dynamischen Adaption von Stellschrauben eines Betriebssystems vorgestellt. Die Vergleiche werden initial während eines Testlaufs durchgeführt. Bei signifikanten Änderungen des Systemzustands wird zufällig eine Stellschrauben-Konfiguration ausgewählt und die resultierende System-Performance gemessen. Liegt der Systemzustand über einen längeren Zeitraum vor, so können in der Zeit mehrere Stellschrauben-Konfigurationen durchprobiert und die jeweiligen Performance-Auswirkungen erfasst werden. Mit der Zeit liegen ausreichend viele Daten vor, um die optimale Konfiguration für jeden Systemzustand zu bestimmen. Testläufe dieses Verfahrens haben geringe Performance-Verbesserungen gezeigt. Die durch die Testphase verursache Last auf dem System wurde in [RP81] jedoch nicht berücksichtigt.

In [SS97] wird ein Ansatz zur Erstellung eines Betriebssystemkernels vorgestellt. Das Betriebssystem überwacht seine Performance und passt sich kontinuierlich der aktuell

anliegenden Workload an. Der Ansatz konzentriert sich dabei auf die Betrachtung von Workload-abhängigen Konfigurations-Strategien wie beispielsweise der Cache-Seiten-Verdrängungsstrategie. Der Ansatz ist jedoch auf ein spezielles, erweiterbares Betriebssystem zugeschnitten, welches den Austausch bestimmter Module im Kernel zur Laufzeit ermöglicht. So kann das Betriebssystem mit den Standard-Strategien weiter operieren, während spezielle Simulationsmodule mögliche alternative Strategien für die anliegende Workload simulieren und ihre Auswirkungen auf die Performance erfassen. Der Ansatz ist aufgrund seiner Abhängigkeit von einem solchen modular aufgebauten, erweiterbaren Betriebssystem nicht verallgemeinerbar und zudem mit hohem Entwicklungsaufwand verbunden.

Selbst-optimierende Betriebssysteme sind der Schwerpunkt von Arbeiten, die in [FN99] beschrieben werden. Hier wird auf ein erschöpfendes Durchprobieren aller möglichen Konfigurationen verzichtet. Stattdessen finden generische Algorithmen für die Suche nach optimalen Konfigurationen Einsatz. In jeder Vergleichsrunde werden mehrere Konfigurationen anhand von Simulationen miteinander verglichen und die weniger effizienten aussortiert. Die verbleibenden, effizienteren Konfigurationen werden leicht verändert und miteinander kombiniert. Es wird dabei erwartet, dass das Verfahren mit einer Auswahl von effizienten Konfigurationen, die die System-Performance verbessern, konvergiert. Das Verfahren wird anhand des Tuning eines parametrisierbaren Scheduling-Algorithmus getestet. Auf Simulationen anderer Aspekte des Betriebssystems wird dabei verzichtet.

[VDB01] beschäftigt sich mit Methoden zur Bestimmung der besten Implementierung einer Unterprogrammbibliothek für eine bestimmte Plattform und eine vorliegende Menge von weiteren Nebenbedingungen. Auch dieses Verfahren basiert auf empirischen Vergleichen zur Identifikation der optimalen Implementierung. Zur Reduktion des Suchraums werden hier jedoch anwendungsspezifische Heuristiken hinzugezogen. Dazu schlagen die Autoren vor, Performance-Statistiken auszuwerten, die bei der Suche nach optimalen Implementierungen gesammelt wurden und anhand davon zu entscheiden, ob die Suche fortgesetzt oder abgebrochen werden soll. Mit statistischen Verfahren wird die Wahrscheinlichkeit dafür bestimmt, dass die beste der bereits untersuchten Implementierungen deutlich schlechter als die (unbekannte) optimale Implementierung ist. Ist die Wahrscheinlichkeit dafür hoch, so wird die Suche fortgesetzt, anderenfalls angehalten und die beste identifizierte Implementierung als die „nahezu optimale“ Implementierung angesehen.

Es ist naheliegend, dass die Ansätze, die auf empirischen Vergleichen basieren, nicht verallgemeinerbar sind. Für jede neue Workload-Art sind entsprechende Anpassungen bzw. ein zusätzliches Training erforderlich. Diese Kategorie von Ansätzen ist daher gut für fixe Workload-Arten geeignet. Dynamische Workloads hingegen resultieren typischerweise in Performance-Degradationen und verursachen große Verzögerungen zwischen der Erkennung von Problemen und den notwendigen Tuning-Maßnahmen.

2.3.1.2 Auf analytischen Modellen beruhende Ansätze

Eine Vorhersage der System-Performance bei einer bestimmten Konfiguration und unter einer bestimmten Workload ist mit Hilfe analytischer Modelle möglich. Auch die Identifikation einer optimalen Konfiguration, die unter einer bestimmten Workload die System-Performance maximiert, ist mit Hilfe solcher Modelle umsetzbar. Ähnlich wie bei den Ansätzen, die auf empirischen Vergleichen basieren (*Abschnitt 2.3.1.1*), erfolgen auch

hier Vergleiche der Performance-Kennzahlen bei bestimmten Konfigurationen für eine bestimmte Workload. Die bei den Vergleichen verwendeten Daten sind jedoch nicht das Ergebnis von Simulationen mit anschließenden Messungen, sondern von Vorhersagen mit Hilfe mathematischer Modelle.

Es existieren mehrere Typen von Modellen, die im Kontext eines autonomen Systems Anwendung finden können. Dazu zählen unter anderem statistische, auf Regression basierende Modelle bzw. probabilistische graphische Modelle. Jeder Modelltyp hat eine Menge assoziierter Parameter, die typischerweise aus den Trainingsdaten erlernt werden, die wiederum Statistiken über die Workload-Arten, einzelne Konfigurationen sowie die System-Performance aus einem bestimmten Zeitintervall beinhalten. Zur Laufzeit können diese Parameter an die Systemumgebungsbedingungen angepasst werden und ermöglichen somit eine genauere Vorhersage.

Wird die aktuelle Systemumgebung durch ein analytisches Modell erfasst, so kann mit diesem Modell für die vorliegende Workload die optimale Konfiguration in nur einem, vergleichsweise kurzen, Berechnungslauf ermittelt werden und bedarf im Gegensatz zum Feedback-orientierten Ansatz (*Abschnitt 2.3.1.3*) keiner iterativen Neuberechnungen und Anpassungen, bis die optimale Konfiguration bestimmt ist. Weiterhin lassen sich die in den analytischen Modellen erfassten Daten verallgemeinern und so beispielsweise optimale Konfigurationen für bisher unbekannte Workloads ermitteln, ohne dabei zusätzliche Konfigurationen durchzutesten bzw. zu simulieren.

Zwar haben die auf analytischen Modellen basierenden Verfahren deutlich kürzere Laufzeiten bei der Ermittlung von optimalen Konfigurationen, der Aufwand bei der Modellerstellung ist jedoch sehr hoch, da in der Regel eine initial ausreichende Menge an Trainingsdaten gesammelt und ausgewertet werden muss. Weiterhin ist die Erstellung solcher Modelle durch die hohe Komplexität heutiger IT-Systeme stark beeinträchtigt. Nicht nur eine hohe Anzahl von Konfigurationsparametern und eine Vielzahl von möglichen Werten pro Parameter, sondern auch die unter Umständen versteckten Abhängigkeiten unter den einzelnen Systemkomponenten machen die Erstellung analytischer Modelle nur schwer möglich, weshalb solche Modelle in der Regel lediglich für einzelne Konfigurationsparameter bzw. stark vereinfachte Szenarien erstellt werden können.

In [Bre94, Bre95] werden Regressionsmodelle zur Optimierung von Unterprogrammbibliotheken verwendet. Dabei findet die lineare Regression Anwendung, die unabhängigen Variablen können jedoch nicht-linear sein. Zwar funktioniert dieser Ansatz bei der Optimierung von Unterprogrammbibliotheken, jedoch ist es unklar, ob auch für komplexe Software-Systeme entsprechende Performance-Modelle erstellt werden können. Problematisch könnten dabei insbesondere die möglichen Abhängigkeiten zwischen den zahlreichen Variablen werden.

Auch die *Odyssey-Plattform* für *Remote-Computing* [NSN+97] verwendet analytische Modelle. Anwendungen werden hier an Änderungen in der Ressourcenverfügbarkeit bzw. in den Nutzerzielen durch eine Veränderung ihres Ausführungsmodus (beispielsweise die Bildwechselfrequenz einer Video-Streaming-Anwendung) angepasst. Dabei wird ein regressionsbasiertes, analytisches Modell bei der Vorhersage des Ressourcenverbrauchs einzelner Anwendungen in Abhängigkeit von relevanten Eingabeparametern und Ausführungsmodi eingesetzt, so dass für jede Operation ein geeigneter Ausführungsmodus empfohlen wird

[NFS00]. Die Sammlung von Trainingsdaten erfolgt hier im Offline-Modus, um den laufenden Betrieb des Systems nicht zu beeinträchtigen.

[Sul03] entwickelt ein Modell zum autonomen Tuning von Anwendungen anhand eines speziellen Abhängigkeitsdiagramms, welches die Grundlage für spezielle Algorithmen bietet. Der Ansatz wurde mit ausgewählten Konfigurationsparametern der *Berkeley DB* getestet. In [Sul03] finden sich jedoch zahlreiche Empfehlungen bezüglich der Übertragung des Ansatzes auf das Tuning beliebiger Software-Systeme.

Im *AutoAdmin*-Projekt von *Microsoft* wurden modellbasierte Techniken zur Automatisierung der Auswahl von Indizes und materialisierten Sichten entwickelt [CN97, ACN00]. Bei der Erstellung des Modells fanden insbesondere Kostenabschätzungen des Datenbankoptimierers Anwendung. Die Ergebnisse dieser Forschungsinitiative wurden teilweise in verschiedene Komponenten des *Microsoft SQL Servers* übernommen. So basieren beispielsweise der *Index Tuning Wizard* (*Microsoft SQL Server 7.0 / Microsoft SQL Server 2000*) sowie der *Database Engine Tuning Advisor* (*Microsoft SQL Server 2005*) auf den Ergebnissen [Mic11a].

Insbesondere im Bereich der autonomen Speicherverwaltung greifen die DBMS-Hersteller auf mathematische Modelle zurück: Eine effiziente Speicherallokation benötigt tiefgreifendes Wissen über die Abhängigkeiten unter den einzelnen DBMS-internen Komponenten. Dieses Wissen lässt sich wiederum in entsprechenden Modellen abbilden.

So überwacht der *Self-tuning Memory Manager (STMM)* von *IBM DB2* die zur Verfügung stehenden Hauptspeicherressourcen und nimmt mit Hilfe eines zugrunde liegenden Modells eine automatische, Workload-abhängige Anpassung verschiedener Konfigurationsparameter vor, die vom Administrator auf den Wert „`Automatic`“ gesetzt wurden[3]. Die betrachteten Konfigurationsparameter beschränken sich dabei auf die Aufteilung des verfügbaren Hauptspeichers auf die einzelnen Bereiche (*Buffer Pools*, *Package Cache*, *Sort Heap*, Sperrliste etc.) [SGAL+06].

Auch die autonomen Funktionalitäten des *Microsoft SQL Servers* wurden in den letzten Jahren kontinuierlich weiterentwickelt. So sind in den aktuellen Versionen des *Microsoft SQL Servers* die autonomen Mechanismen in Punkto Speichermanagement besonders ausgereift [Mic11b]. Der Arbeitsspeicher kann hier dynamisch verschiedenen Konsumenten zur Verfügung gestellt werden. Dabei wird versucht, so viel Speicher zu allokieren, wie nötig ist, ohne dabei Speicherknappheit auf dem System zu verursachen. Durch eine tiefe Integration des Ansatzes in die hauseigene (*Microsoft*-)Betriebssystemebene kann hier insbesondere auf Speicherprobleme reagiert werden, die nicht nur durch DBMS-interne, sondern auch andere Prozesse verursacht wurden.

Des Weiteren unterstützen die aktuellen Versionen der *Oracle Database* eine automatische Verteilung des verfügbaren Hauptspeichers auf die Konsumenten. Dafür ist das *Automatic Shared Memory Management (ASMM)* verantwortlich [CLP05]. Nach Vorgabe einer Obergrenze für den verfügbaren Hauptspeicher erfolgt hier eine Workload-abhängige Aufteilung des Speichers auf die Bereiche wie *Buffer Cache*[4], *Shared Pool*, *Large Pool* etc. Im Gegensatz zu *IBM DB2* und *Microsoft SQL Server* ist hier eine Speicherneuverteilung

[3] Der *Self-tuning Memory Manager* findet in *IBM DB2* seit der Version 9.1 Anwendung.
[4] Der Oracle-Begriff *Buffer Cache* entspricht dem Begriff *Buffer Pool* aus der DB2-Welt.

ausschließlich im Rahmen des zum Start fest allokierten Speichers möglich. Weiterhin ist hier aktuell eine Neuanpassung der Größen der Sortierbereiche nicht möglich.

Es handelt sich bei den soeben vorgestellten Ansätzen von *IBM*, *Microsoft* und *Oracle* zur autonomen Speicherverwaltung nicht um Verfahren, die *ausschließlich* auf mathematischen Modellen basieren. Typischerweise profitieren diese Ansätze von den Vorteilen mehrerer Vorgehensweisen und greifen beispielsweise auf Kombinationen aus Simulationsmodellen, Kosten-Nutzen-Analyse, Betriebssystem-Ressourcen-Analyse sowie Feedback-Schleifen-basierten Ansätzen zurück [SGAL+06].

Auch wir unternahmen in [Gan06, Exn07, RW07] einen Versuch der Modellierung eines DBMS-Ausschnitts am Beispiel von *IBM DB2*. Dabei wurde versucht, die Zusammenhänge zwischen Effektoren und Sensoren des Systems zu erfassen und in einem mathematischen Modell abzubilden, um damit eine optimale Systemkonfiguration entsprechend den vorgegebenen Zielsensorwerten identifizieren zu können. Bei der Modellerstellung wurden sogenannte Performance-Funktionen mit Hilfe der nicht-linearen Regression approximiert. Wie grundsätzlich bei der Erstellung von analytischen Modellen üblich, war der Aufwand bei der Erstellung des Modells bereits bei nur wenigen betrachteten Effektoren und Sensoren sehr hoch und nahm beim Einbeziehen weiterer Effektoren und Sensoren drastisch zu. Auch die systeminternen Mechanismen (wie asynchrone Prozesse bzw. Automatismen) beeinträchtigten die Erstellung des Modells, so dass schließlich auf die Weiterverfolgung dieses Ansatzes im Rahmen der vorliegenden Arbeit verzichtet wurde.

2.3.1.3 Auf Feedback-Schleifen beruhende Ansätze

Wie bereits in *Abschnitt 2.2.4* erwähnt, ist die Verwendung einer rückgekoppelten Kontrollschleife ein vielversprechender Ansatz bei der Konzeption und Umsetzung von autonomen Systemen. Dabei erfolgt eine kontinuierliche Überwachung des Systems und in bestimmten Fällen wird eine iterative Anpassung von Tuning-Stellschrauben eingeleitet. Der durch die Konfigurationsveränderung erreichte, neue Systemzustand wird analysiert und bei Bedarf eine neue Iteration mit einer erneuten Anpassung der Tuning-Stellschrauben eingeleitet.

Während Ansätze, die auf empirischen Vergleichen basieren, die Performance verschiedener Konfigurationen, die unter einer bestimmten Workload erreicht wurde, miteinander vergleichen und daraus eine optimale Konfigurationsparameterbelegung unter dieser Workload ableiten, wird beim Feedback-Schleifen-orientierten Ansatz ermittelt, welche Konfigurationsparameter einen Einfluss auf die untersuchte Performance-Metrik ausüben. Diese Konfigurationsparameter werden anschließend iterativ verändert, bis der Zielzustand des Systems erreicht wurde und keine weiteren Rekonfigurationen notwendig sind. In der Regel erfolgt bei solchen Ansätzen insbesondere kein Vergleich von verschiedenen Konfigurationsszenarien. Auch die anliegende Workload wird bei diesen Ansätzen typischerweise nicht berücksichtigt. Unser in *Abschnitt 2.3.2* vorgestellter Ansatz zum Feedback-Schleifen-basierten Tuning von Datenbank-Management-Systemen ermöglicht im Gegensatz dazu nicht nur die Berücksichtigung aktueller Workload, sondern kann durch die Auswertung von entsprechenden Metadaten optimale Tuning-Strategien ermitteln und somit die Qualität des Tuning beispielsweise durch ein schnelleres Erreichen des vorgegebenen Zielzustands steigern.

Feedback-Schleifen-basierte Ansätze bieten eine Reihe von Vorteilen. So ist insbesondere eine schnelle Anpassbarkeit an die sich ändernde Systemumgebung hervorzuheben. Des Weiteren verzichten diese Ansätze auf eine initiale Training-Phase. Nicht zuletzt ist diese Vorgehensweise Workload-unabhängig und kann daher auch bei unbekannten Workloads effektiv eingesetzt werden. Durch eine Integration von mathematischen Modellen bzw. empirischen Vergleichen in die Planungsphase eines Feedback-Schleifen-basierten Ansatzes können weiterhin die Systemzielzustände schneller erreicht werden.

Es sind aber auch die Nachteile solcher Ansätze zu nennen. Zum einen ist es nur schwer möglich, mehrere Konfigurationsparameter simultan zu optimieren. Während bei der Betrachtung eines isolierten Konfigurationsparameters lediglich die Entscheidung getroffen werden muss, ob dieser erhöht oder verringert werden muss, um sich dem Zielzustand anzunähern, muss bei der Betrachtung mehrerer Konfigurationsparameter zunächst untersucht werden, inwiefern diese von einander abhängen und welchen gemeinsamen Einfluss auf den Zielzustand sie ausüben. Durch eine vorwiegend iterative Vorgehensweise kann es weiterhin unter Umständen vorkommen, dass eine hohe Anzahl an Durchläufen der Feedback-Schleife gebraucht wird, bis der Zielzustand des Systems erreicht werden kann. Diesem Problem kann jedoch beispielsweise durch die Verwendung von analytischen Modellen und somit durch eine bessere Schätzung des neuen Parameterwerts (also des Delta zum aktuellen Parameterwert) entgegengewirkt werden. Es ist auch zu berücksichtigen, dass oft der Zielzustand des Systems nicht exakt spezifizierbar ist bzw. eine optimale Konfiguration entweder erst nach unangemessen vielen Schleifendurchläufen erreicht werden kann oder aber auch gar nicht zu erreichen ist. Mit entsprechenden Mechanismen muss in diesen Fällen dafür Sorge getragen werden, dass die Schleifenausführung rechtzeitig beendet wird.

Auch auf diesem Gebiet existieren zahlreiche Ansätze sowohl in der Forschung als auch in der Industrie. Die nachfolgende Auswahl basiert größtenteils auf [Sul03, RWRA10, RWRA08, WRRA08].

In Betriebssystemen finden Feedback-Mechanismen bereits seit vielen Jahren Anwendung. So existieren beispielsweise zahlreiche Ansätze auf dem Gebiet des CPU-Scheduling [CMDD62, MP90, SGG+99] bzw. des Kontrolflusses in Netzwerken [Jac95, Kes95]. Um den Einsatz von Feedback-Schleifen-basierten Mechanismen bei der Ressourcenverwaltung zu erleichtern, entwickelte [GSPW99] einen Toolkit von einfachen, modularen Feedback-Komponenten, die sich miteinander kombinieren und wiederverwenden lassen.

In [WHMZ94] wird im Rahmen des *COMFORT*-Projekts das Ziel verfolgt, Tuning-Prozesse zu automatisieren oder gar vollständig in die Verantwortung des Systems zu geben. Die Betrachtungen konzentrieren sich dabei auf den *Grad des Multiprogramming* (*Multiprogramming Level, MPL*), wodurch größtenteils der Durchsatz des Systems beeinflusst wird. Übersteigt die Anzahl der Sperrkonflikte eine kritische Schranke, so wird der MPL-Wert gesenkt. Fällt die Anzahl der Sperrkonflikte hingegen unter einen kritischen Wert, so kann der MPL-Wert wieder angehoben werden. In [WMHZ02] werden unter anderem Erkenntnisse und Nutzen des COMFORT-Projektes im Hinblick auf heutige DBMS und deren Anforderungen diskutiert und eine Neuorientierung für die Entwicklung neuer Datenbankprodukte angeraten.

In [BMCL94, Bro95, BCL96] werden Feedback-Schleifen-basierte Mechanismen vorgestellt, die sich auf die Einstellung von speicherrelevanten Konfigurationsparametern sowie auf die

Laststeuerung konzentrieren. Dabei wird versucht, in Abhängigkeit von der aktuell anliegenden Workload bestimmte Antwortzeiten des Systems zu gewährleisten. Der Konfigurationsprozess läuft bis entweder das vorgegebene Tuning-Ziel erreicht ist oder festgestellt wird, dass es nicht erreicht werden kann. Im Gegensatz zu den meisten anderen Ansätzen, die sich jeweils lediglich auf das Tuning eines einzelnen Parameters beschränken, werden in diesem Ansatz simultan zwei Stellschrauben verändert. Dennoch erwähnen die Autoren, dass es äußerst schwierig ist, eine Feedback-Schleifen-basierte Lösung zu entwickeln, die mehr als einen Parameter pro Schleifendurchlauf anpasst.

Zwar finden bei den aktuellen Speicherverwaltungsmechanismen von *Microsoft SQL Server* hauptsächlich mathematische Modelle Anwendung (vgl. *Abschnitt 2.3.1.2*), dennoch basierten die ersten Arbeiten auf diesem Gebiet auf einer rückgekoppelten Kontrollschleife. Wie in [CGNZ99] beschrieben, wurde mit ihrer Hilfe die automatische Anpassung der Cache-Größe umgesetzt. Die automatisch ermittelte Cache-Größe hängt in diesem Ansatz von verfügbarem freien Speicher im System ab. Fällt die Anzahl der freien Speicherseiten unter eine bestimmte Schranke, so wird die Cache-Größe reduziert, damit wieder mehr Speicherplatz zur Verfügung steht. Übersteigt die Anzahl von freien Seiten eine andere Schranke, so wird die Cache-Größe entsprechend angehoben. Die Information darüber, wie die beiden Schranken gewählt werden bzw. ob eine Berücksichtigung von weiteren Speicherkonsumenten erfolgt und wie diese umgesetzt wird, bleibt dem Leser jedoch vorenthalten.

In den neueren Versionen von Oracle Database findet eine Feedback-Schleife unter dem Namen *Self-Managing Loop* [DD06] Anwendung. Diese besteht aus drei Phasen: der Observierung, der Diagnose und schließlich der Problemauflösung. Verschiedene Advisor-Komponenten setzen auf der Diagnose-Phase auf. In Abhängigkeit von den Ergebnissen der Observierung sowie Diagnose werden dem Administrator Vorschläge zur Handlung unterbreitet. Der Administrator kann anschließend die Ausführung ausgewählter Vorschläge anstoßen, wodurch die Problemauflösungsphase der *Self-Managing Loop* eingeleitet wird.

Das von IBM veröffentlichte *„Architectural Blueprint for Autonomic Computing“* beschreibt eine Referenzarchitektur zur Umsetzung von autonomen Systemen [IBM06a]. Die Architektur basiert dabei auf dem MAPE-Schleifen-Paradima (vgl. *Abschnitt 2.2.4*). Die vorgeschlagene Architektur ist in den letzten Jahren zur Grundlage zahlreicher Forschungsansätze geworden.

In [PM06] wird beispielsweise ein generisches Framework eingeführt, welches auf dem MAPE-Paradigma aufbaut und dabei ausschließlich von Konzepten relationaler DBMS wie *Triggers* bzw. *Stored Procedures* Gebrauch macht.

Die Forschungsgruppe um Hellerstein stellt eine Verbindung zwischen dem *Autonomic Computing* und der Kontrolltheorie her [DHP+05, BHJS00]. Basierend auf der Ähnlichkeit beider 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 zugrunde liegenden Ressourcenmodells unter Berücksichtigung von Sensorverzögerungen und Ausführungszeiten.

Benoit beschreibt ein Verfahren zur automatischen Problemdiagnose von Performance-Engpässen in *IBM DB2* unter OLTP-Workload [Ben05]. Ausgehend von einem Ressourcen-

Modell wird ein systemspezifischer, auf fest verdrahteten Heuristiken beruhender Diagnosebaum entwickelt. Dieser Entscheidungsbaum wird zur Laufzeit traversiert und genutzt, um durch schrittweise Anpassung problembehafteter Ressourcen die Gesamt-Performance des Systems zu verbessern.

An dieser Stelle ist zu betonen, dass obwohl wir hier in Anlehnung an [Sul03] eine Aufteilung der Ansätze zur Umsetzung autonomer Mechanismen in drei Kategorien (empirische, analytische bzw. auf Feedback-Schleifen basierende Ansätze) vornehmen, die Grenzen zwischen den Kategorien jedoch verschwimmen. Viele Ansätze erstrecken sich über mehrere Kategorien und vereinen somit die Vorteile dieser Kategorien. Sehr oft finden sich insbesondere Ansätze, denen eine rückgekoppelte Kontrollschleife zugrunde liegt, die jedoch auch von mathematischen bzw. Simulationsmodellen profitieren, um insbesondere die Planungsphase der Kontrollschleife zu verbessern.

2.3.2 Best-practices-orientiertes Datenbank-Tuning

Die Wahl des Ansatzes hängt in der Regel von dem zu optimierenden System, seiner Umgebung und dem Umfang der gewünschten autonomen Funktionalitäten ab. Ist das System einer geringen Anzahl von einander unterscheidbarer, regelmäßig wiederkehrender Workload-Arten ausgesetzt, so ist die Wahl des auf empirischen Vergleichen beruhenden Ansatzes besonders ratsam. Unterliegt dagegen die Workload ständigen Schwankungen und sind die Tuning-Ziele durch entsprechende Richtlinien vorgegeben, so sind die analytischen Modelle bzw. Feedback-Schleifen-basierte Ansätze besonders effektiv. Während ein Feedback-Schleifen-basierter Ansatz in der Regel mehrere Schleifendurchläufe benötigt, um sich dem Zielzustand des Systems anzunähern, haben die auf analytischen Modellen basierenden Techniken den Vorteil, nach einer kurzen Berechnungszeit die „optimale“ Systemkonfiguration ermitteln zu können. Der Aufwand bei der Modellerstellung ist jedoch, wie bereits in *Abschnitt 2.3.1.2* erwähnt, sehr hoch und insbesondere bei hoch komplexen Systemen nicht zu vernachlässigen. Der Ansatz führt also also bei komplexen Systemen in der Regel nicht zum Erfolg. Außerdem verursacht er oft eine zusätzliche Systemauslastung, die im laufenden Betrieb das zu verwaltende System stark beeinträchtigen kann.

Neben einer Unterscheidung zwischen den drei oben genannten Ansätzen kann zwischen zwei Herangehensweisen bezüglich der Platzierung der neuen autonomen Systemkomponenten unterschieden werden: dem *Integrationsansatz* sowie dem *Auslagerungsansatz*.

Der *Integrationsansatz* verfolgt das Ziel, die autonomen Mechanismen in das bestehende System zu integrieren. Bedingt durch eine tiefe Systemintegration und den dadurch ermöglichten Zugriff auf die Systeminterna bieten sich hier insbesondere mathematische bzw. auf Simulationen basierende Modelle an.

Wie dem *Abschnitt 2.3.1.2* bzw. [WR07b, WR10] zu entnehmen ist, verfolgen die DBMS-Hersteller bei der Umsetzung autonomer Funktionalitäten den Integrationsansatz. Die aktuell verfügbaren autonomen Funktionalitäten der weit verbreitetsten DBMS wie *IBM DB2*, *Oracle Database* bzw. *Microsoft SQL Server* lassen sich weiterhin wie folgt kategorisieren [WR10]:

- Mechanismen zur *statischen* Optimierung des physischen Datenbank-Designs und der Datenbank-Konfiguration sowie

- Mechanismen zur Umsetzung *dynamischer* DBMS-interner Systemverwaltungsfunktionalitäten.

Während die Mechanismen der ersten Kategorie für die Umsetzung einer initialen Systemkonfiguration bzw. des physischen Datenbankentwurfs verantwortlich sind, umfasst die zweite Kategorie Bereiche wie die Hauptspeicher- sowie Plattenspeicherverwaltung, Workload-Verwaltung, Wartung sowie automatische Verwaltung von Statistiken. Für eine vergleichende Gegenüberstellung diesen Kategorien zugeordneter autonomer Mechanismen der drei „großen“ Datenbank-Management-Systeme *IBM DB2*, *Microsoft SQL Server* sowie *Oracle Database* sei an dieser Stelle auf [WR10] bzw. [Wie11] verwiesen.

Bei dieser Evaluierung autonomer Mechanismen konnte festgestellt werden, dass die DBMS-Hersteller bereits erste große Schritte zur Selbstverwaltung erfolgreich gemacht haben. Die meisten verfügbaren Werkzeuge bzw. Mechanismen sind aktuell jedoch bezüglich des Reifemodells des *Autonomic Computing* (*Abschnitt 2.2.3*) zwischen den Stufen „*Managed*“ und „*Predictive*“ einzuordnen, da sie noch nicht vollständig autonom arbeiten und ein gewisses Maß an Vor- bzw. Nachbereitung der Daten bzw. eine Interaktion mit dem DBA benötigen [RMHA09]. Zwar decken die autonomen Mechanismen und Werkzeuge einzelne Bereiche der DBMS-Verwaltung ab, jedoch fehlt zumeist noch eine durchgehende Integration dieser Mechanismen, die eine problembereichsübergreifende Selbstverwaltung des DBMS entsprechend den Nutzervorgaben ermöglichen würde.

Alternativ zum Integrationsansatz kann der *Auslagerungsansatz* verfolgt werden. Hier werden die autonomen Mechanismen in Systemkomponenten gekapselt, die auf dem zu verwaltenden System aufbauen. Zwar ist dieser Ansatz in der Regel als weniger effizient einzustufen, jedoch verfügt er über entscheidende Vorteile. Dadurch, dass die autonomen Mechanismen außerhalb des zu verwaltenden Systems angesiedelt sind und eine Interaktion mit diesem System ausschließlich über die entsprechenden Schnittstellen erfolgt, wird hier eine *System-übergreifende* Administration ermöglicht. So lassen sich unter Berücksichtigung der Nutzervorgaben ganze IT-Systeme optimieren, indem beispielsweise die einzelnen Subsysteme wie das Betriebssystem, das DBMS, der Anwendungs- sowie der Webserver aufeinander abgestimmt werden. Des Weiteren ermöglicht dieser Ansatz eine einfache, schrittweise Erweiterung eines bestehenden Systems um autonome Funktionalitäten. Bei der Umsetzung von ausgelagerten autonomen Systemkomponenten bieten sich insbesondere Feedback-Schleifen-basierte Mechanismen an.

Aufgrund der hohen Flexibilität und der Leichtgewichtigkeit des Feedback-Schleifen-basierten Ansatzes und insbesondere aufgrund einer dadurch ermöglichten System-übergreifenden Administration erscheint dieser Ansatz im Rahmen unserer Forschungsarbeiten sehr vielversprechend. Wie bereits in *Abschnitt 2.3.1.3* erwähnt, kann dabei die Planungsphase dieses Ansatzes durch die Integration mathematischer Modelle verbessert und somit der durch Richtlinien vorgegebene Zielzustand schneller erreicht werden. Im Gegensatz zur in *Abschnitt 2.2.4* vorgestellten Vorgehensweise mit mehreren hierarchisch angeordneten bzw. gleichberechtigten *Autonomic Managern* haben wir uns jedoch für die Verwendung eines einzigen *Autonomic Managers* entschieden, der für die Überwachung und Administration des gesamten zu verwaltenden Systems verantwortlich ist. Dadurch lassen sich sowohl der Koordinationsaufwand als auch die Wahrscheinlichkeit des Auftretens von Seiteneffekten und Oszillationen reduzieren.

Bei der Umsetzung einer rückgekoppelten Kontrollschleife stellt sich die Frage, in welchen Fällen die Anpassung von Tuning-Stellschrauben eingeleitet werden soll und wie die neuen Werte für diese Stellschrauben zu bestimmen sind. Die Problemsituationen lassen sich beispielsweise durch die Definition von kritischen Schwellenwerten für einzelne Systemmetriken umsetzen. Sowohl für die Wahl von geeigneten Schwellenwerten als auch von effektiven Vorgehensweisen zur Problemlösung lassen sich bewährte Datenbank-Tuning-Praktiken hinzuziehen, die in elektronischer Form beispielsweise in News-Groups, IT-Blogs, Online-Magazinen, „IBM Redbooks"-Veröffentlichungen oder Produkthandbüchern zur Verfügung stehen. All diese Informationen sind jedoch zum größten Teil unstrukturiert, das heißt nicht formalisiert, so dass ihre Verwertung mit einem hohen Aufwand für die Administratoren verbunden ist. Zu einer effizienten Integration von bewährten Datenbank-Tuning-Praktiken in eine *Autonomic-Computing*-Architektur ist eine Formalisierung dieser Informationen jedoch unabdingbar [RWRA10].

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[5]. Die Medizinische Informatik entwickelte in diesem Zusammenhang formale Repräsentationen für SOPs sowie Workflow-Systeme für deren semi-automatische Abarbeitung [PTB+03, SRRM06]. Der Fokus der Medizinischen Informatik war hierbei jedoch nicht die vollständige Automatisierung der Behandlung, sondern die Qualitätsunterstützung bzw. -sicherung.

Darüber hinaus sind die angesprochenen Verfahren nur begrenzt erweiterbar und das automatische Tuning ist lediglich auf spezifische Bereiche sowie eine Auswahl an Konfigurationsparametern zugeschnitten. Im Gegensatz dazu zeichnet sich der in diesem Beitrag vorgeschlagene Ansatz durch seine weitergehende Universalität aus. Die Überwachung und Administration des zu tunenden (Datenbank-Management-) Systems erfolgen dabei ausschließlich über definierte Schnittstellen. Wissen über System-Interna wird grundsätzlich nicht benötigt, kann aber im Tuning-Prozess verwertet werden. Dadurch ist der Ansatz nicht ausschließlich auf das Tuning von Datenbank-Systemen spezialisiert und kann zur Administration beliebiger IT-Systeme verwendet werden.

Die Fähigkeit zur adaptiven Reaktion auf aufgetretene Problemsituationen bzw. einer Vorhersage und proaktiven Vermeidung dieser erfordert zudem Wissen über zu kontrollierende Ressourcen und deren dynamische Umgebung. Gerade weil universelle DBMS wie *IBM DB2* zunehmend unterschiedlichen, schwankenden Workloads ausgesetzt werden, ist das Wissen über die Art der Workload für das autonome Tuning entscheidend. Die Arbeit von Elnaffar beschreibt eine System-unabhängige Workload-Erkennung für *DB2* mit Hilfe von Data-Mining-Techniken [Eln04]. Jedoch wird dabei der Einfluss des (autonomen) Tuning auf die Workload-Erkennung nicht berücksichtigt. In *Abschnitt 5.2.3* wird dieser Ansatz aufgegriffen und weitgehend erweitert, um eine System- und Tuning-unabhängige Workload-Erkennung zu ermöglichen.

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

Der im Rahmen vorliegender Dissertation vorgestellte Ansatz zum best-practices-orientierten Datenbank-Tuning mittels *Autonomic Tuning Expert (ATE)* kombiniert zentrale Konzepte des *Autonomic Computing* mit dem an SOPs angelehnten Konzept der Formalisierung des Tuning-Wissens und den Mechanismen zur Workload-Erkennung. Dies ermöglicht eine nutzerfreundliche Definition von maschinenlesbaren Tuning-Abläufen (*Tuning-Plänen*) durch die Administratoren, eine semantische Auswertung dieser zur Laufzeit, ihr Austausch in einer Community und automatische Workload-abhängige Anwendung dieser bewährten Datenbank-Tuning-Praktiken zur Laufzeit. Die einzige Voraussetzung für unseren Ansatz ist die Bereitstellung einer Sensor- sowie einer Effektor-Schnittstelle durch das zu verwaltende System. Diese Schnittstellen bieten die Anknüpfungspunkte für den auf dem Konzept einer MAPE-Schleife aufbauenden und die Rolle eines *Autonomic Manager* übernehmenden *Autonomic Tuning Expert* und ermöglichen eine kontinuierliche Überwachung und bei Bedarf eine Rekonfiguration des Systems im Sinne einer mittels Tuning-Richtlinien vorgegebenen optimalen Konfiguration bzw. anderer Tuning-Ziele.

In den nachfolgenden Kapiteln befassen wir uns zunächst mit der Beschreibung von Performance-Problemen (*Kapitel 3*) sowie von Tuning-Abläufen zur Behebung dieser Probleme (*Kapitel 4*). Anschließend wird in *Kapitel 5* die Architektur des *Autonomic Tuning Expert* vorgestellt, die für eine Erkennung von Performance-Problemen und autonome Einleitung von entsprechenden, vorab im System hinterlegten, Tuning-Abläufen zuständig ist. Ausgewählte Aspekte zur Steuerung des Tuning-Prozesses werden in *Kapitel 6* vorgestellt. Und schließlich stellt *Kapitel 7* Konzepte zum Austausch von durch die Administratoren definierten Tuning-Abläufen in einer Community vor.

Kapitel 3

Klassifikation und Beschreibung von Performance-Problemen

Aufgrund der zunehmenden Komplexität moderner IT-Anwendungen steigen auch die Anforderungen an die IT-Systeme und ihre Performance. Zur Erfassung der System-Performance kann auf die Verwendung von Performance-Metriken zurückgegriffen werden. Diese bauen auf den Sensoren einzelner Systemressourcen auf (vgl. *Abschnitt 2.1.1*) und ermöglichen die Beschreibung des IT-System-Zustands in Form einer Auswahl von Performance-Metriken.

Jede inakzeptable Abweichung des Ist-Systemzustands von dem normalen bzw. dem durch die Anforderungen vorgegebenen Soll-Zustand wird als problematisch angesehen und sollte daher im Rahmen eines autonomen Performance-Tuning automatisch erkannt werden. In diesem Kapitel werden daher Konzepte vorgestellt, die eine Erkennung bzw. Analyse solcher inakzeptablen Abweichungen ermöglichen und somit die Grundlage zur Problembehebung (*Kapitel 4*) bilden.

Da in dieser Arbeit unter anderem das Ziel verfolgt wurde, die durch den Problemerkennungsprozess verursachte Belastung des überwachten Systems zu reduzieren, wurde versucht auf die Einbindung neuer Systemüberwachungskomponenten zu verzichten. Aus diesem Grund wurde hier die Integration von typischerweise in komplexen IT-Infrastrukturen bereits vorhandenen, sich über mehrere Schichten der IT-Infrastruktur erstreckenden Überwachungswerkzeugen in den Problemerkennungsprozess angestrebt.

Unter Berücksichtigung dieses Ziels lässt sich die Beschreibung von Problemsituationen basierend auf einem zweistufigen Ansatz umsetzen. Für die einzelnen integrierten Überwachungswerkzeuge erfolgt dabei im ersten Schritt die Betrachtung der den Systemzustand repräsentierenden Metriken, auf denen „ungesunde“ Werte bzw. Wertebereiche definiert werden können, die auf ein Problem hindeuten und daher im Rahmen eines weiteren Problemdiagnose- bzw. Tuning-Vorgehens adressiert werden sollten. Zum einen können jedoch solche Werte bzw. Wertebereichsüberschreitungen in einer komplexen Systemumgebung häufig auftreten. Zum anderen beziehen sich solche atomaren Ereignisse typischerweise auf einzelne Systembereiche (wie Netzwerk, *Application Server* bzw. Datenbank-Management-System), in denen sie durch entsprechende Überwachungswerkzeuge isoliert voneinander erkannt werden.

Im zweiten Schritt erfolgt daher eine Konsolidierung dieser Ereignisse aus sämtlichen aktiven Überwachungswerkzeugen und eine mittels Eventkorrelationsmechanismen darauf durchgeführte Erkennung von vorab definierten Ereignismustern. Dadurch kann nicht nur die Eventflut reduziert werden. Insbesondere wird durch dieses Vorgehen ermöglicht, die

einzelnen Problemsituationen miteinander in Beziehung zu bringen und die Ursachen der aufgetretenen Probleme durch die Berücksichtigung des entsprechenden Problemkontexts (waren noch andere Schwellenwerte überschritten, während die interessierende Metrik den „gesunden“ Wertebereich verlassen hat, oder ähnliches) genau einzugrenzen.

Bevor wir uns mit einer formalisierten Beschreibung von Performance-Problemen befassen, untersuchen und klassifizieren wir in *Abschnitt 3.1* die Performance-Probleme im Datenbankumfeld. Anschließend werden in *Abschnitt 3.2* Konzepte zur Verarbeitung komplexer Ereignisse vorgestellt, da diese in unserem Ansatz eine wichtige Rolle spielen. Der darauf folgende *Abschnitt 3.3* beschäftigt sich mit der Beschreibung von primitiven Ereignissen und stellt exemplarisch einige Werkzeuge zur Erkennung und Analyse von Datenbankproblemen vor. In *Abschnitt 3.4* wird anschließend erläutert, mit Hilfe welcher Operatoren primitive Ereignisse korreliert werden können und welche weiteren Aspekte bei der Ereignisekorrelation eine wichtige Rolle spielen. Dabei findet auch eine kurze Vorstellung der *Tivoli Active Correlation Technology* statt, die im Rahmen unseres Kooperationsprojekts Anwendung fand. Eine Zusammenfassung der Ergebnisse dieses Kapitels findet sich schließlich in *Abschnitt 3.5*.

3.1 Performance-Probleme im Datenbankumfeld

Grundsätzlich können Performance-Probleme auf jeder der Systemschichten (Hardware-, Betriebssystem-, DBMS- bzw. Anwendungsschicht) entstehen (vgl. *Abschnitt 2.1.1*). Je nach Ursprung des Problems unterscheiden wir daher zwischen Anwendungs- (Datenbankanwendungsschicht), Datenbank- (DBMS-Schicht) und Umgebungsproblemen (Hardware- bzw. Betriebssystemschicht). Während die Problemursache in einer bestimmten Schicht zu finden ist, erstrecken sich die Auswirkungen eines Problems üblicherweise über mehrere höher liegende Schichten. So kann beispielsweise ein Hardwareproblem zu Feststellung von Symptomen (wie hohe Zugriffszeiten) in der DBMS-Schicht führen.

Da der Schwerpunkt der vorliegenden Arbeit auf dem Tuning von Datenbank-Systemen liegt, konzentrieren wir uns in den nachfolgenden Betrachtungen primär auf die Datenbankprobleme und stellen zunächst einige Ansätze zu ihrer Klassifikation vor.

3.1.1 Klassifikation nach Problembereichen

Performance-Probleme können nach Problembereichen eingeteilt werden. *Abbildung 3.1* veranschaulicht eine solche Klassifikation. Zunächst kann zwischen Datenbank- und Systemumgebungsproblemen unterschieden werden. Die Systemumgebungsprobleme umfassen die Problembereiche Betriebssystem, Hardware, Netzwerk und Anwendungsprogramme. Die Datenbank-Probleme lassen sich weiter in drei Bereiche einteilen: Probleme mit der Datenbankkonfiguration, Probleme mit dem Datenbankentwurf und Probleme mit den Datenbankanwendungen. Nachfolgend findet sich eine detaillierte Beschreibung dieser Problembereiche. Dabei wird zwar oft auf DB2-spezifische Beispiele zurückgegriffen, jedoch ist der hier vorgestellte Klassifikationsansatz universell und Produkt-unabhängig.

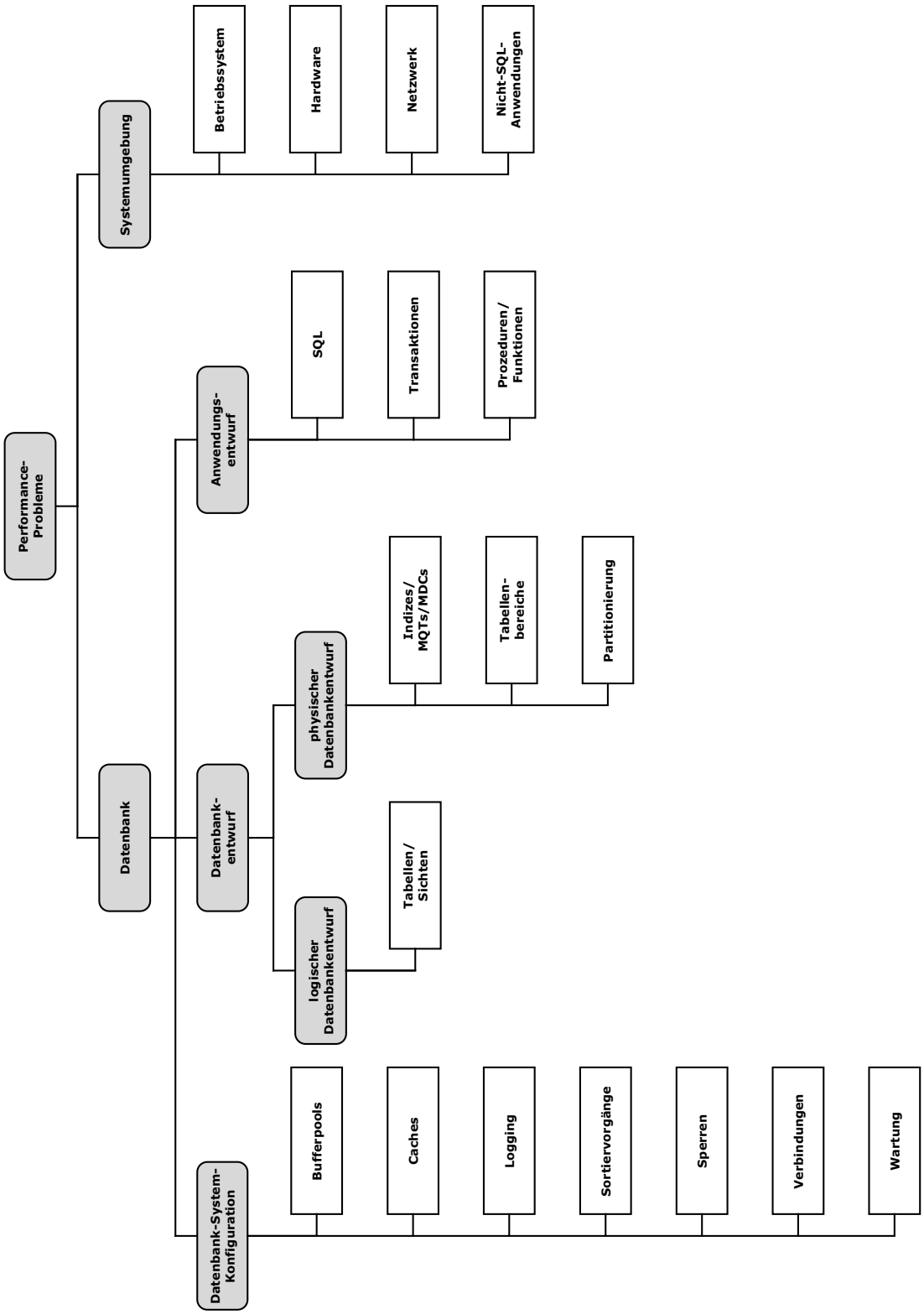


Abbildung 3.1: Kategorisierung von Datenbankproblemen nach Problembereichen

Datenbank-System-Konfiguration

Die Hauptursache für Performance-Probleme dieser Kategorie liegt in einer falschen Konfiguration entsprechender Datenbank-System-Ressourcen. Das Konfigurieren dieser Ressourcen ist mit Hilfe von entsprechenden Konfigurationsparametern möglich. Nähere Erklärungen zu den DB2-spezifischen Konfigurationsparametern finden sich in [IBM06g, IBM06d, IBM06f].

Bufferpools. Aufgrund falscher Entscheidungen bezüglich der Größe bzw. Anzahl der Bufferpools kann es zu Performance-Problemen kommen. Hat beispielsweise ein Bufferpool eine falsche Größe, so kann es dazu führen, dass aufgrund einer Verdrängung der Daten aus dem Bufferpool die Notwendigkeit der Plattenzugriffe steigt. Oft lässt es sich erreichen, dass bestimmte, Performance-kritische Daten länger im Bufferpool vorgehalten werden, indem dem entsprechenden Tabellenbereich (*Tablespace*) ein eigener Bufferpool zugeordnet wird.

Caches. Zu den Cache-Problemen zählt beispielsweise die falsche Konfiguration des Parameters für die Größe des *Package Cache* (`pckcachesz`). Mit Hilfe des Paket-Cachings wird eine Reduktion des internen Systemaufwands ermöglicht, da bei erneuter Benutzung eines Pakets kein erneuter Zugriff auf den Systemkatalog (statische Anweisungen) bzw. keine erneute Kompilierung (dynamische Anweisungen) stattfinden muss. Besonders leistungssteigernd wirkt sich der *Package Cache* bei einer wiederholten Ausführung gleicher bzw. ähnlicher Anweisungen aus.

Auch eine falsch eingestellte Katalogcachegröße (`catalogcache_sz`) fällt in diesen Problembereich. Durch das Caching von Kataloginformationen lässt sich der interne Systemaufwand verringern, da der Externspeicherzugriff auf den Systemkatalog zum Abruf bereits zuvor abgerufener Informationen nicht mehr erforderlich ist. Von einem richtig konfigurierten Katalogcache können beispielsweise nicht nur das Binden von Paketen und Kompilieren von SQL-Anweisungen, sondern auch sämtliche Operationen, die eine Überprüfung von Zugriffsrechten umfassen, profitieren.

Logging. Ein Beispiel für ein Logging-Problem ist der falsch eingestellte Konfigurationsparameter für die Protokollpuffergröße (`logbufsz`). Der Protokollpuffer wird für die Zwischenspeicherung der Protokollsätze verwendet, bevor sie auf die Festplatte geschrieben werden, was zu einer Senkung der Anzahl von I/O-Operationen führt.

Die Größe der Protokolldatei (`logfilsiz`) spielt ebenfalls eine wichtige Rolle. Dadurch wird die Größe jeder primären und sekundären Protokolldatei definiert und damit die Anzahl der Protokollsätze, die in die Dateien geschrieben werden können, beschränkt. Eine zu kleine Protokolldatei kann sich negativ auf die Systemleistung auswirken, da das Archivieren alter Protokolldateien, das Zuordnen neuer Protokolldateien sowie das Warten auf verwendbare Protokolldateien einen erhöhten Systemaufwand mit sich bringen.

Sortiervorgänge. DB2 führt Sortieroperationen durch, sobald in dem abgesetzten SQL-Statement ein explizites `order by`, `group by`, `union` oder `distinct` vorkommt. Auch bei OLAP- bzw. Aggregationsfunktionen sowie bei der Reorganisation bzw. der Index-Erzeugung 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 Partitions-internen 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 *gemeinsamen Sortiervorgängen* (*Shared Sorts*). Im Gegensatz dazu handelt es sich um *Private Sortiervorgänge* (*Private Sorts*), wenn dieser Parallelismus inaktiv ist und jeder DB2-Agent auf seinen privaten Speicherbereich (*Agent Private Memory*) zum Sortieren zugreift.

Die Steuerung der Sortiervörgänge erfolgt unter anderem mittels der Konfigurationsparameter `sortheap` für die Zwischenspeichergröße der Sortierlisten sowie `sheapthres` für den Schwellenwert für Sortierspeicher.

Mit dem Parameter `sortheap` wird die maximale Anzahl von Seiten des privaten Speichers definiert, die für private Sortiervorgänge verwendet werden sollen bzw. die maximale Anzahl von Seiten des gemeinsamen Speichers, die für gemeinsame Sortiervorgänge verwendet werden sollen. Reicht der durch `sortheap` definierte Speicher für die Sortierung nicht aus, kommt es zu einem kostspieligen Überlauf (*Sort Overflow*).

Private und gemeinsame Sortiervorgänge verwenden Speicher aus verschiedenen Speicherquellen. Die Größe des gemeinsamen Sortierspeicherbereichs wird auf der Grundlage des Werts von `sheapthres` statisch vorherbestimmt. Durch die explizite Angabe dieses Schwellenwerts wird verhindert, dass der Datenbankmanager „auf Verdacht“ übermäßig große Speichermengen für sehr viele Sortiervorgänge verwendet. Zu berücksichtigen ist, dass beide Konfigurationsparameter in einem unmittelbaren Zusammenhang miteinander stehen und entsprechend zusammen eingestellt werden sollten (der Wert von `sheapthres` sollte auf ein Vielfaches des Parameters `sortheap` mit dem größten Wert gesetzt werden) [IBM06h].

Sperren. Performance-Probleme, die auf das Sperrverhalten von Anwendungen zurückzuführen sind, fallen in diesen Bereich. Falsche Parameterwerte für den maximalen Speicher für Sperrliste (`locklist`) bzw. für die maximale Anzahl Sperren pro Anwendung (`maxlocks`) führen unter Umständen zu einer hohen Anzahl an Sperreskalationen. Bei den *Sperreskalationen* handelt es sich um das Ersetzen mehrerer feingranularer Sperren (beispielsweise Zeilensperren) durch eine gröbere Sperre (beispielsweise eine Tabellensperre), um die Anzahl der Sperren in der Liste zu verringern [IBM06f]. Der Parameter `maxlocks` legt fest, wie viel Prozent der Sperrliste eine Anwendung belegen muss, damit eine Eskalation durchgeführt wird. Ist in der Sperrliste aufgrund von Sperren mehrerer Anwendungen kein weiterer Platz mehr verfügbar, so wird eine Sperreskalation ausgeführt, auch wenn keine der Anwendungen den durch `maxlocks` festgelegten Prozentsatz an Sperrlistenbelegung erreicht hat.

Der Konfigurationsparameter `locktimeout` gibt die Dauer an, die eine Anwendung auf eine Sperre wartet, bis die Wartesituation mittels eines *Locktimeouts* aufgelöst wird. Dies trägt zur Vermeidung globaler gegenseitiger Sperren von Anwendungen (*Deadlocks*) bei. Ein falsch gesetzter Wert für diesen Konfiguationsparameter resultiert unter Umständen in einer hohen Anzahl an (spät erkannten) Deadlocks.

Für Vorgehensweisen zur Auflösung von Sperrproblemen sei hier auf *Abschnitt 4.2* sowie [Roe08] verwiesen.

Verbindungen. Zu dieser Problemkategorie zählen sämtliche Probleme des Verbindungsmanagements, wie beispielsweise ein falsch konfiguriertes *Connection Pooling* – eine Technologie, die es ermöglicht, eine aufgebaute Verbindungsinfrastruktur für nachfolgende Verbindungen erneut zu verwenden [IBM06b]. In diesem Zusammenhang sind richtige Einstellungen unter anderem für die Konfigurationsparameter für die maximale Anzahl von Client-Verbindungen (`max_connections`) sowie für die maximale Anzahl von Agenten (`maxagents`) von Bedeutung. Auch der Konfigurationsparameter `maxappls`, der die maximale Anzahl von Anwendungen angibt, die gleichzeitig auf eine Datenbank zugreifen können, spielt im Verbindungsmanagement eine wichtige Rolle.

Wartung. Zu den Wartungsproblemen zählen Performance-Probleme, die auf unzureichende Wartung des laufenden Systems zurückzuführen sind. In DB2 sind *Reorg* und *Runstats* die gängigen Werkzeuge für die Wartung des Systems.

Nach zahlreichen Änderungen an Tabellen- bzw. Indexdaten (Einfügungen, Änderungen, Löschungen) können sich logisch sequentielle Daten auf physisch nicht sequentiellen Datenseiten befinden (verloren gegangene Cluster-Eigenschaft), so dass beim Zugriff auf die Daten zusätzliche Leseoperationen nötig sind und die System-Performance sinkt. Mit Hilfe des Werkzeugs *Reorg* kann eine Reorganisation der Tabelle durchgeführt werden, im Rahmen welcher die Daten defragmentiert werden und der unnötig belegte Speicherplatz freigegeben wird. Dabei lassen sich die Daten anhand eines bestimmten Index neu anordnen, wodurch der Zugriff über diesen Index effizienter wird.

Mit Hilfe des Befehls *Runstats* lassen sich die Statistiken aktualisieren. Insbesondere für Tabellen bzw. Indizes, deren Daten häufigen Einfügungen, Änderungen und Löschungen unterlagen, ist eine Aktualisierung von Statistiken sinnvoll. Des Weiteren sollten die Statistiken nach einem vorgenommenen *Reorg* aktualisiert werden. Verzichtet man auf eine zeitnahe Statistikaktualisierung, so kann sich das dramatisch auf die Performance des Systems auswirken, da die vom Optimizer aufgrund veralteter Statistiken bestimmten Zugriffspfade sich als nicht vorteilhaft erweisen können.

Nach Durchführung einer Reorganisation der Daten sowie einer Aktualisierung von Statistiken sollten die Datenbankpakete mittels des Befehls *Rebind* neu gebunden werden, damit auch das statische SQL von den aktuellen Statistiken profitieren kann.

Datenbankentwurf

In diese Kategorie fallen Probleme, die durch „schlechten“ Datenbankentwurf hervorgerufen werden. Dabei lässt sich weiterhin zwischen den Problemen des physischen bzw. logischen Datenbankentwurfs unterscheiden.

Während beim *logischen* Datenbankentwurf die Domänen der zu verwaltenden Datenelemente unter Berücksichtigung ihrer Beziehungen untereinander in Relationen aufgeteilt werden, wird beim *physischen* Datenbankentwurf festgelegt, wie die im Rahmen des logischen Datenbankentwurfs modellierten Sachverhalte physisch gespeichert werden. Dabei spielen folgende Aspekte eine Rolle:

- Verteilung der Daten auf mehrere Systeme (verteilte Datenbanken),
- Aufteilung der Daten auf mehrere physische Speicherorte (Partitionierung der Datenbank),
- Festlegung von Tabellenbereichen (der Speicherstrukturen zur physischen Speicherung von Daten),
- Definition von Hilfsobjekten wie Indizes, materialisierte Sichten (in DB2 unter dem Namen *Materialized Query Tables, MQTs* bekannt) bzw. Mehrdimensionale Clustering-Tabellen (*Multidimensional Clustering Tables, MDCs*) zum effizienten Datenzugriff.

Tabellen/Sichten. Das primäre Problem des logischen Datenbankentwurfs liegt in einer ungünstigen Datenmodellierung, was unter Umständen zu einer sehr schlechten System-Performance führen kann. Aus Performance-Gründen ist es oft ratsam, auf eine zu starke Normalisierung von Daten zu verzichten, um etwa (zu) viele kleine Tabellen sowie Joins zu vermeiden.

Indizes/MQTs/MDCs. Sämtliche Probleme, die auf eine inadäquate Verwendung von Mechanismen wie Indizes, materialisierte Sichten bzw. mehrdimensionale Clustering-Tabellen zurückzuführen sind, fallen in diese Kategorie.

Mit Hilfe von Indizes kann für eine Performance-Steigerung beim lesenden Datenzugriff gesorgt werden. Insbesondere bei häufigen lesenden Zugriffen auf bestimmte Tabellenspalten ist das Anlegen eines entsprechenden Index auf diese Tabellenspalte ratsam. Bei Datenänderungen müssen die Indizes jedoch aktualisiert werden, so dass bei einer hohen Anzahl an Änderungsoperationen der Aufwand für die Indexaktualisierung den Indexnutzen beim lesenden Zugriff übersteigen kann.

Materialisierte Sichten ermöglichen das Vorhalten von Anfrageergebnissen in speziellen Tabellen. Erkennt der Optimizer, dass der Zugriff auf eine solche materialisierte Sicht effizienter ist als der Zugriff auf die entsprechenden Basistabellen, so wird die Anfrage umgeschrieben. Die Speicher- und Verwaltungskosten von materialisierten Sichten sind jedoch nicht zu vernachlässigen. Es ist daher wichtig, den Nutzen und die Kosten dieses Hilfsmittels in einzelnen Fällen abzuwägen.

Das mehrdimensionale Clustering ermöglicht eine flexible, kontinuierliche und automatische Anordnung von Daten in Clustern in Bezug auf mehrere Dimensionen. Dadurch kann die Anfrageleistung gesteigert und der Aufwand von Datenpflegeoperationen (Reorganisations- und Indexpflegeoperationen bei Einfügungen, Änderungen und Löschungen) verringert werden. Dieses Verfahren findet primär in Data-Warehouse-Anwendungen Einsatz. Auch die mehrdimensionalen Clustering-Tabellen haben nicht zu vernachlässigende Verwaltungskosten, die es zu berücksichtigen gilt. Für weitere Informationen zu Indizes, MQTs sowie MDCs sei hier auf [SSH10, IBM06f] verwiesen.

Tabellenbereiche. Durch Aufteilung der Daten auf einzelne Tabellenbereiche (*Tablespaces*) kann die Verwaltung großer Datenmengen vereinfacht werden. Tabellenbereiche können über mehrere physische Speichereinheiten (*Container*) verteilt werden, um die Leistung zu verbessern. Weiterhin kann beispielsweise pro Tabellenbereich

festgelegt werden, ob die Indexdaten bzw. die *Large-Object-Daten* (*LOB-Daten*) separat von den restlichen Tabellendaten gespeichert werden sollen.

Ist die Datenverteilung über Tabellenbereiche und ihre Container nicht optimal an die Systemanforderungen und die Gegebenheiten der zugrunde liegenden Hardware angepasst, so kann dies in Performance-Problemen resultieren, die dann dieser Kategorie zuzuordnen sind.

Partitionierung. Die Partitionierung erleichtert die Administration großer Datenmengen und steigert die Zugriffsperformance durch die Ermöglichung einer Parallelverarbeitung oder auch Partitionsauschluss beim Zugriff.

Bei der Tabellenpartitionierung wird beim Erstellen einer Tabelle festgelegt, auf wie viele Partitionen und nach welchem Verteilungskriterium die Daten dieser Tabelle aufgeteilt werden sollen [SSH10]. Dabei lassen sich die Tabellenpartitionen über mehrere Tabellenbereiche verteilen. Wurden diese Tabellenbereiche auf unterschiedlichen Speicherplatten angelegt, so kann dadurch eine Parallelisierung der Plattenzugriffe erreicht werden.

Mit Hilfe der Datenbankpartitionierung lassen sich die Daten auf mehrere Rechnerknoten bzw. mehrere Festplatten verteilen. Auch damit kann eine gleichmäßige Verteilung der Daten erreicht werden, was eine Parallelisierung der Plattenzugriffe und somit eine Performance-Steigerung ermöglicht.

Datenbankanwendungsentwurf

In diese Kategorie fallen nun die Probleme, deren Ursache in einem nicht optimalen Datenbankanwendungsentwurf liegt. Die typischen Vertreter dieser Problemklasse sind die Probleme mit den SQL-Statements selbst, Probleme in der Anwendungslogik, die im problematischen Transaktionsverhalten resultieren bzw. Probleme, die durch eine fehlerhafte Verwendung von Prozeduren oder Funktionen verursacht wurden. Die Performance-Probleme dieser Kategorie lassen sich genauso wie die Probleme beim logischen Datenbankentwurf nicht automatisch beheben, da sie einen Eingriff in die Anwendungssemantik erfordern.

SQL. Zu den klassischen SQL-Problemen gehört fehlender Gebrauch von vorbereiteten Anweisungen (*Prepared Statements*). Eine Anweisung wird vorbereitet, indem sie syntaktisch analysiert („geparst“) wird und ein optimaler Zugriffsplan dafür bestimmt wird. In Fällen, wo eine Anweisung mehrfach ausgeführt werden muss, ist es ratsam, auf das Konzept der vorbereiteten Anweisungen zurückzugreifen, da dadurch die Kosten für Parsen und Optimieren der SQL-Anweisung nur einmal bei der Vorbereitung und nicht bei jeder Ausführung anfallen, wie es sonst der Fall wäre. Insbesondere in OLTP-Umgebungen führt ein Verzicht auf vorbereitete Anweisungen zu starken Performance-Problemen, da dort typischerweise die Anfragen weniger Anfragetypen dominieren und sich nur anhand der Werte voneinander unterscheiden (z. B. Eingang einer neuen Bestellung eines bestimmten Artikels mit der Artikelnummer x).

Transaktionen. Ein weit verbreitetes Problem ist die unzureichende Verwendung von Commit-Statements in der Anwendungslogik, was zu langlaufenden Transaktionen führt. Dieser Effekt wird oft dadurch verstärkt, dass innerhalb der Transaktionen auf Nutzereingaben gewartet wird. Lange Transaktionen erhöhen durch lange Ressourcenbindung die Wahrscheinlichkeit für Deadlocks bzw. hohe Wartezeiten auf Freigabe von Sperren und sollten daher nach Möglichkeit vermieden werden.

Prozeduren, Funktionen. Dieser Kategorie sind Probleme, die durch eine falsche Verwendung von bzw. Fehler in Prozeduren (*Stored Procedures*) oder Funktionen (*User-definied Functions*) verursacht wurden, zuzuordnen.

Mit Hilfe von Prozeduren bzw. Funktionen lassen sich bestimmte Funktionalitäten einer Anwendung in das DBMS verlagern. Dadurch kann einerseits die Performance (durch die Ausführung des Codes auf einem leistungsstarken Datenbankserver) gesteigert und andererseits, falls der Großteil der für die Berechnung benötigten Daten in der Datenbank gespeichert ist, die Kommunikation zwischen der Anwendung und dem Datenbankserver eingespart werden. Das Ausführen von Anwendungslogik auf dem Datenbankserver kann jedoch auch Probleme mit sich bringen. Wird eine Prozedur nicht abgeschirmt (Deklaration als *not-fenced*), so hat diese einen direkten Zugriff auf sämtliche Systemressourcen. Ein fehlerhafter bzw. bösartiger Code kann dann beispielsweise die Leistung des Datenbankservers beeinträchtigen oder vorgegebene Sicherheitsrichtlinien verletzen.

Systemumgebung

In diese Problemkategorie fallen sämtliche Nicht-Datenbankprobleme, wie Probleme mit der Hardware, dem Betriebssystem oder den (Nicht-SQL-)Anwendungen.

Betriebssystem. Zu Betriebssystemproblemen zählen beispielsweise die falsche Betriebssystem-Konfiguration oder Treiberprobleme.

Hardware. Bei dieser Problemkategorie handelt es sich um Hardware-Probleme wie defekte oder zu wenige Festplatten, ungünstige Datenverteilung auf die Festplatten etc.

Netzwerk. In diese Kategorie fallen sämtliche Netzwerkprobleme, die sowohl durch fehlerhafte Netzwerkhardware als auch durch falsche Netzwerkkonfiguration verursacht wurden.

Nicht-SQL-Anwendungen. Dazu zählen Anwendungsprobleme wie beispielsweise fehlerhafte oder unoptimierte Algorithmen, die nicht auf Datenbankprobleme zurückzuführen sind.

3.1.2 Klassifikation nach Problemursachen bzw. -auswirkungen

Neben der bereits erwähnten Möglichkeit der Klassifikation der Performance-Probleme nach den Problembereichen kommt auch die Klassifikation von Problemen nach ihren Auswirkungen in Betracht. Eine exemplarische Veranschaulichung dieses Klassifikationsansatzes findet sich in ***Abbildung 3.2***. Im Beispiel ist eine Auswahl der Problembereiche

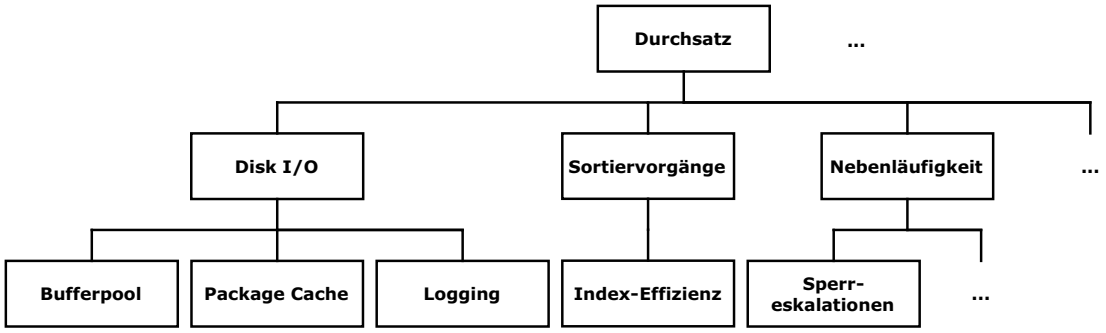


Abbildung 3.2: Beispiel einer Kategorisierung von Datenbankproblemen nach ihren Auswirkungen

enthalten, die einen Einfluss auf den Systemdurchsatz[6] haben. Diese sind dabei hierarchisch angeordnet, so dass die möglichen Ursachen eines Performance-Problems als seine Kinder in der Problemhierarchie dargestellt werden.

Der Systemdurchsatz kann also (unter anderem) durch folgende Probleme negativ beeinflusst werden:

- Probleme mit Lese- bzw. Schreiboperationen von/auf Festplatte (*Disk I/O*)
- Sortierprobleme
- Nebenläufigkeitsprobleme

Disk I/O-Probleme werden dabei durch falsch konfigurierte Bufferpools bzw. *Package Cache* sowie Logging-Probleme verursacht. Die Sortierprobleme können beispielsweise durch fehlende Indizes verursacht werden. Die Nebenläufigkeit wird durch eine hohe Anzahl von Sperreskalationen beeinträchtigt.

Eine solche Problemklassifikation unterstützt insbesondere die Problemdiagnose sowie -auflösung. Werden beispielsweise bestimmte Symptome (z. B. schlechter Durchsatz) beobachtet, so lassen sich entsprechende Problembereiche bestimmen, die diese Symptome verursacht haben könnten (z. B. *Disk I/O*, Sortierprobleme bzw. Nebenläufigkeitsprobleme). Nach einer Analyse und weiteren Eingrenzung der identifizierten Problembereiche kann die Suche nach Ursachen der verbleibenden zu untersuchenden Probleme (z. B. *Disk I/O*) fortgesetzt werden. Durch einen Abstieg in der Hierarchie bei der näheren Problemanalyse und -eingrenzung werden Probleme identifiziert, deren direkte oder indirekte Auswirkungen die System-Performance (z. B. Systemdurchsatz, hohe Antwortzeiten von bestimmten Anwendungen etc.) beeinträchtigen.

Obwohl in *Abbildung 3.2* eine Hierarchie von Problembereichen angegeben ist, ist eine rein hierarchische Darstellung unter Umständen nicht ausreichend. So können hier Probleme mit den Sortiervorgängen beispielsweise einen Einfluss auf die *Disk I/O*-Performance haben. In *Abschnitt 6.1.2* werden daher die Problemabhängigkeiten mit Hilfe von gerichteten Graphen modelliert.

[6] Systemdurchsatz wurde hier lediglich als ein Beispiel gewählt. Genauso wichtig beim Tuning sind weitere Aspekte wie beispielsweise Antwortzeiten.

Eine Klassifikation der Probleme nach ihren Auswirkungen kann sich unter anderem auf die Problemdiagnose unterstützend auswirken. Werden auf einer höheren Systemschicht bestimmte Symptome beobachtet, so lassen sich entsprechende Problem-Klassen bestimmen, deren Auswirkungen sich in den beobachteten Symptomen auf der aktuellen Schicht bemerkbar machen. Somit kann der Suchraum bei der Problemdiagnose eingeschränkt werden und es kann ermittelt werden, wo genau die Problemsuche fortzusetzen ist.

3.1.3 Klassifikation nach Problemausmaß

Die Klassifikation der Performance-Probleme nach dem Ausmaß des Problems kann beispielsweise in den Stufen „kritisch“, „mittelkritisch“ und „unkritisch“ erfolgen. Die Entscheidung, ob ein Problem als „kritisch“, „mittelkritisch“ oder „unkritisch“ eingestuft wird, kann dabei unter Berücksichtigung des Kontexts dieses Problems erfolgen. Ein Performance-Problem führt zu einer Verletzung von einer oder mehreren Anforderungen an das DBMS. In Abhängigkeit von der Gewichtung dieser Anforderungen lässt sich die Problemeinstufung vornehmen. Eine solche Klassifizierung ist beispielsweise beim gleichzeitigen Auftreten mehrerer Performance-Probleme hilfreich. Im Rahmen des Performance-Tuning kann dann automatisch entschieden werden, welche Performance-Probleme zuerst angegangen werden müssen.

3.2 Eventverarbeitungsmechanismen

Verarbeitung komplexer Ereignisse (engl.: *Complex Event Processing*, *CEP*) ist ein von David Luckham geprägter Begriff, der sämtliche Techniken der Eventverarbeitung primär aus den Bereichen der Eventsimulation, der Netzwerktechnik, der Aktiven Datenbanken und der Service-Orientierten Architektur vereinigt. Obwohl diese Wurzeln von CEP bis in die 1950-er Jahre zurückreichen [Luc07], entwickelte und etablierte sich CEP erst in den letzten Jahren zu einem eigenständigen Forschungsgebiet. Die Gründung der *Event Processing Technical Society (EPTS)* [EPT10] Anfang 2008 zeigt die Wichtigkeit der Verarbeitung komplexer Ereignisse für die Industrie.

Die Einsatzmöglichkeiten von Eventverarbeitungsmechanismen sind mittlerweile sehr breit. So lassen sich mit ihrer Hilfe unter anderem Anwendungen mit folgenden Funktionalitäten umsetzen [EN10]:

Monitoring. Hier erfolgt mit Hilfe der Eventverarbeitung eine Überwachung eines Systems bzw. eines Prozesses, indem das Verhalten dieses Systems bzw. Prozesses beobachtet wird und bei Abweichungen entsprechende Warnungen (*Alerts*) ausgelöst und zuständige Personen benachrichtigt werden.

Aktive Diagnose. Bei der aktiven Diagnose geht es darum, ein Problem ausgehend von beobachteten Symptomen zu diagnostizieren und wenn möglich zu beheben. Dabei können in Abhängigkeiten von festgestellten Symptomen weitere Systemuntersuchungen initiiert werden.

Dynamisches Verhalten. Weiterhin lässt sich durch die Eventverarbeitung ein dynamisches Systemverhalten umsetzen, indem auf bestimmte Events mit entsprechenden Aktionen reagiert wird. Ein Monitoring-Dienst erkennt beispielsweise „problematische“ Zustände und generiert Events, die anschließend durch entsprechende Mechanismen ausgewertet werden. Als Reaktion auf (unter Umständen aggregierte) erkannte Events können entsprechende Aktionen ausgeführt werden, die das Verhalten des Systems beeinflussen. Das Verhalten des Systems hängt also unmittelbar von den ausgewerteten Events ab.

Prädiktive Verarbeitung. Bei der prädiktiven Eventverarbeitung soll versucht werden, die Events vor ihrem Eintreten vorherzusagen und durch entsprechende präventive Maßnahmen vorzubeugen. Für die Eventvorhersage können beispielsweise Trends von Metrikverläufen ausgewertet oder es kann auf bekannte Werteverläufe aus der Vergangenheit zurückgegriffen werden.

Informationsverteilung. Mit Hilfe der Eventverarbeitungsmechanismen kann auch die Informationsverteilung umgesetzt werden. Dabei geht es darum, die richtige Information im richtigen Umfang zur richtigen Zeit an die richtige Stelle zu transportieren.

Zur Ermöglichung einer Überwachung und Auswertung von Ereignissen sind entsprechende Mechanismen zur Ereignisbeschreibung unabdingbar. [Eck08] bezeichnet diese als Ereignisanfragesprachen und unterscheidet zwischen Kompositionssprachen, Datenstromabfragesprachen und Produktionsregeln. In der vorliegenden Arbeit konzentrieren wir uns primär auf die Betrachtung von Kompositionssprachen, die ursprünglich den aktiven Datenbank-Systemen [Pat99] entstammen und eine Zusammensetzung einzelner, primitiver Ereignisse ermöglichen. Bei der Zusammensetzung von komplexen Ereignissen kommen Operatoren zur Event-Filterung, -Transformation bzw. Mustererkennung zum Einsatz (vgl. *Abschnitt 3.4.1*).

Im nachfolgenden *Abschnitt 3.3* werden zunächst die primitiven Events vorgestellt. *Abschnitt 3.4* fasst anschließend einige Konzepte zur Bildung von komplexen Events zusammen. Eine kurze Einführung in die Funktionsweise einer typischen Korrelationsengine am Beispiel von *Tivoli Active Correlation Technology*, die auch in unserem Prototyp (vgl. *Abschnitt 5.2*) eingesetzt wird, findet sich in *Abschnitt 3.4.3*.

3.3 Primitive Events

Bei den primitiven Events[7] handelt es sich um die Events, die direkt von einem sogenannten *Event-Erzeuger* [EN10] generiert wurden. Dabei kann zwischen drei Arten von Event-Erzeugern unterschieden werden: Die Events können einerseits von Hardware- bzw. Software-Komponenten generiert werden. Andererseits können Nutzer gewisse Systemzustände als problemhaft identifizieren und für die Generierung eines entsprechenden Events sorgen.

[7] In der vorliegenden Arbeit werden die Begriffe „primitive Events“ und „atomare Events“ synonym verwendet.

Hardware-Event-Erzeuger. Bei dieser Art von Event-Generierungsmechanismen handelt es sich um Hardware-Komponenten, die mit Sensoren ausgestattet sind und dadurch Systemveränderungen erkennen können. Solche physischen Sensoren können beispielsweise auf Temperatur, Feuchtigkeit, Druck, Licht, Bewegung, Strom oder ähnliches reagieren und finden in verschiedenen IT-Bereichen wie Sicherheitssysteme, medizinische Geräte, Logistiksysteme oder Wettervorhersage Anwendung.

Software-Event-Erzeuger. Events finden oft in der Anwendungsentwicklung Einsatz. Dabei sorgt eine Software-Komponente für die Erzeugung von bestimmten Events, die an andere Software-Komponenten zwecks Auswertung gereicht werden.

Software-generierte Events lassen sich auch im Rahmen von Überwachungsfunktionalitäten verwenden. Dabei werden die Events nicht durch die Anwendung selbst, sondern durch die Software, die diese Anwendung überwacht, generiert. Ereignisse, die aus einem bestimmten Grund für das Überwachungssystem von Bedeutung sind, werden erkannt und an die zuständigen Eventauswertungskomponenten weitergereicht. Diese Art der Überwachung ist auch als Monitoring bekannt.

Bezieht eine Software-Komponente Informationen aus verschiedenen Quellen, um basierend darauf Events zu generieren, spricht man von einem *Adapter*. Mit Hilfe von solchen Adaptern lassen sich mehrere Anwendungen miteinander verknüpfen und so anwendungsübergreifende Events erkennen. Die Adapter können dabei oft von anderen Anwendungen generierte Events verarbeiten und miteinander in Beziehung setzen. In diesem Zusammenhang spricht man von komplexen Events (*Abschnitt 3.4*).

Nutzer als Event-Erzeuger. Im Gegensatz zu den beiden oben genannten Kategorien können Events auch durch Nutzerinteraktion mit einer Software generiert werden. Benötigt die Anwendungslogik einer Software beispielsweise, dass der Nutzer eine bestimmte Aktion ausführt (z. B. Entgegennehmen einer Barzahlung) und wird die Ausführung dieser Aktion über die Benutzeroberfläche bestätigt, so wird von der Software ein entsprechender Event generiert und der damit verbundene Workflow angestoßen. Ein Benutzer kann auch eine bestimmte Problemsituation erkennen und durch ein Benutzeroberfläche einen entsprechenden Event auslösen.

Nachdem die Events durch Event-Erzeugungsmechanismen generiert und durch Event-Verarbeitungsmechanismen aggregiert oder mit zusätzlichen Informationen angereichert werden, werden sie an die sogenannten *Event-Konsumenten* übergeben. Dieser Ablauf ist in ***Abbildung 3.3*** skizziert. Durch einen gestrichelten Pfeil wurde dabei eine mögliche Rückführung verarbeiteter bzw. im Laufe der Eventverarbeitung neu generierter Events zurück in die Eventverarbeitung, um sie beispielsweise mit anderen Events in Beziehung zu setzen, angedeutet.

Bei den *Event-Konsumenten* handelt es sich um Komponenten, die die verarbeiteten Events empfangen und in Abhängigkeit von der Anwendungslogik auswerten. Die Event-Konsumenten können dabei die zugehörige Event-beschreibende Information beispielsweise für eine spätere Verwendung abspeichern, mit Hilfe spezieller Visualisierungstechniken dem Nutzer präsentieren oder auf einzelne Events mit entsprechenden Aktionen reagieren.

Ähnlich wie die Event-Erzeuger lassen sich auch die Event-Konsumenten in drei Kategorien einteilen: Hardware- bzw. Software-Event-Konsumenten sowie Benutzer als Event-Konsumenten.

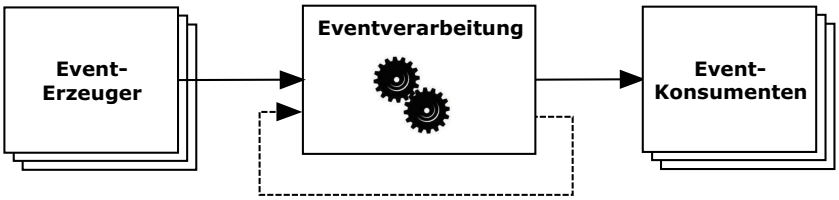


Abbildung 3.3: Eventverarbeitungsablauf (in Anlehnung an [EN10])

Hardware-Event-Konsumenten. Bei den Hardware-Event-Konsumenten handelt es sich um physische Aktoren (bzw. Effektoren) wie beispielsweise Schalter oder Regler, die zur hardwareseitigen Ausführung von Aktionen in Reaktion auf die Events verwendet werden können.

Software-Event-Konsumenten. Zu den Software-seitigen Event-Konsumenten zählen Mechanismen, die beispielsweise das Verändern von Systemeinstellungen ermöglichen oder aber einfach für die Abspeicherung von eingehenden Events in sogenannten *Event-Logs* sorgen. Anschließend können solche Event-Logs für System-übergreifende Auswertungen verwendet werden oder unter Umständen den zuständigen Nutzern präsentiert werden (vgl. Nutzer als Event-Konsument).

Nutzer als Event-Konsumenten. Während die Software-Event-Konsumenten direkt auf eingehende Events reagieren, stellt die bei dieser Art der Event-Konsumenten eingesetzte Software nur ein Werkzeug bereit, welches bei jedem wichtigen Event eine Benachrichtigung des entsprechenden Nutzers ermöglicht. Oft finden dabei sogenannte *Dashboards* Anwendung. Dabei handelt es sich um eine besondere Art der Benutzerschnittstelle, die Informationen aus verschiedenen Quellen konsolidiert und für den Nutzer visualisiert. Insbesondere Überwachungsanwendungen, wie beispielsweise *IBM DB2 Performance Expert* bzw. *IBM Tivoli Monitoring for Databases* (*Abschnitt 3.3.2.2*), greifen oft auf diese Art der Informationsdarstellung zurück.

Bei den vorhergehenden Betrachtungen sind wir primär davon ausgegangen, dass Events beim Erkennen eines ungewöhnlichen Systemverhaltens generiert werden. In der Praxis findet jedoch auch eine weitere Event-Art häufig Anwendung. Dabei handelt es sich um sogenannte *Logging Events*, die bei erwartungsgemäßer (normaler) Funktionsweise von Systemkomponenten generiert werden und somit die Erstellung eines Logs von erfolgreich abgeschlossenen Aktionen ermöglichen. Typische *Logging Events* würden beispielsweise die Ereignisse wie Start von Systemkomponenten, Abschluss von Transaktionen oder Verbindungsaufnahme zu einer Datenbank umfassen. Auch solche Events sind für die Erkennung von Problemen verwendbar. So könnte beispielsweise das Ausbleiben entsprechender *Logging Events* ein Indikator dafür sein, dass das System nicht erwartungsgemäß arbeitet (etwa dass das System „hängt“ oder andere Fehlfunktionen zum Ausbleiben von *Logging Events* geführt haben).

3.3.1 Eventattribute

Ein Event besteht aus mehreren Eventattributen, die die Beantwortung von folgenden Fragen ermöglichen:

- Was ist passiert?
- Wann ist es passiert?
- Welche weiteren Nebenbedingungen spielen dabei eine Rolle?

Die Eventattribute können nach ihrem Einsatzzweck in drei Kategorien eingeteilt werden:

Generische Attribute. Diese Attribute umfassen Metainformationen über den Event. Unter anderem gehören dazu nicht nur Informationen über den Typ des Events (atomar oder zusammengesetzt), sondern auch Informationen über die Eventausprägung (Eventinstanz-ID oder Zeitpunkt des Eventauftretens).

Eventspezifische Attribute. Während die generischen Attribute generische Eventinformationen zusammenfassen, die für Events aller Typen gleich bleiben, beschreiben die eventspezifischen Attribute die tatsächlichen Eventausprägungen. Die Attribute bestimmen sich dabei durch den im Eventheader angegebenen Eventtyp.

Weiterführende Informationen. In diese Kategorie fallen Attribute, die die Informationen zusammenfassen, welche nicht in Attributen der beiden oben genannten Kategorien enthalten sind. Oft sind diese Attribute unstrukturiert (Textattribute).

Die generischen Attribute sind in jedem Event enthalten. Die eventspezifischen Attribute sowie die weiterführenden Informationen sind dagegen optional. Eine ähnliche Unterteilung der Attribute in drei Kategorien (*Header*, *Payload* und *Open Content*) wird auch in [EN10] vorgenommen.

3.3.2 Sensorbasierte Events

Wie bereits in *Abschnitt 2.2* erwähnt, stellt jede im Sinne des *Autonomic Computing* kontrollierte Ressource zwei Arten von Schnittstellen zur Verfügung. Dabei handelt es sich um Sensoren und Effektoren. Während die Effektoren Mechanismen zur Änderung des Zustands der Ressource bereitstellen, ermöglichen die Sensoren die Erfassung von Informationen über den Status dieser Ressource.

Aufbauend auf solchen Sensoren lassen sich sensorbasierte Events definieren. Um eine effektive Erkennung und anschließende Auflösung von Problemen zu ermöglichen, ist es jedoch von Bedeutung, sich nicht ausschließlich auf Problemereignisse eines Systems zu konzentrieren. Stattdessen sollte man auch die Umgebung dieses Systems überwachen, um beispielsweise Betriebssystem- oder Netzwerkprobleme zu erkennen, die wiederum für das fehlerhafte Verhalten der Hauptanwendung verantwortlich sein können. Im Rahmen dieser Arbeit konzentrieren wir uns auf das Datenbank-Management-System *IBM DB2 UDB* und verdeutlichen die Vorgehensweise anhand von DB2-spezifischen Werkzeugen und entsprechenden Events. Dazu werden zunächst in folgenden Abschnitten die Werkzeuge zur Erkennung und Analyse von Datenbankproblemen vorgestellt.

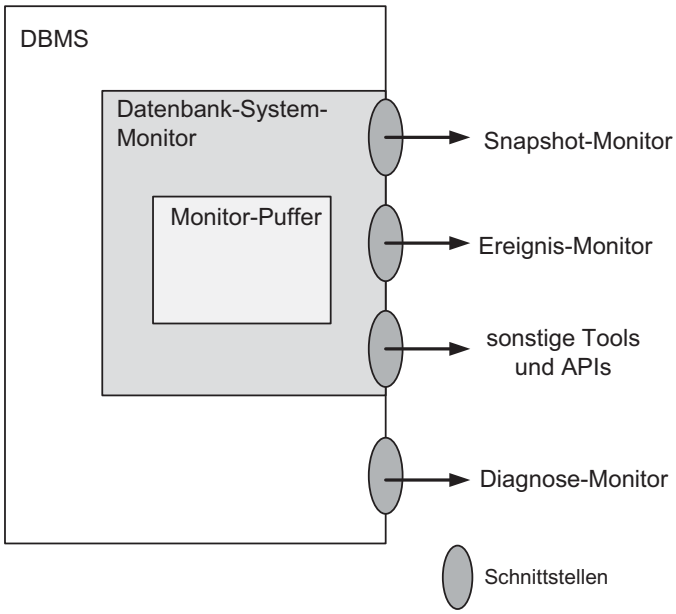


Abbildung 3.4: Der Datenbank-System-Monitor [OGT04]

3.3.2.1 DB2-interne Werkzeuge zur Erkennung und Analyse von Datenbankproblemen

Nachfolgend findet sich eine Auswahl von DB2-internen Überwachungsmechanismen [IBM06h], die primär zum Sammeln von Probleminformationen dienen und daher eine Informationsquelle für die Problemidentifikation bieten. Zur Automatisierung der Problemidentifikation lassen sich auf den Daten bzw. Ausgaben dieser Werkzeuge aufbauend Ereignisse definieren, die für das Anstoßen entsprechender Diagnose- bzw. Tuning-Prozesse verwendet werden können. Auf die Beschreibung von Werkzeugen, wie beispielsweise der *Memory Visualizer* [IBM06h, IBM06e] oder der *Event Analyzer* [IBM06e], die sich auf die Auswertung von gesammelten Informationen konzentrieren, wurde in den nachfolgenden Ausführungen verzichtet.

Datenbank-System-Monitor. Der *Datenbank-System-Monitor* sammelt Informationen zum Datenbankmanager, dessen Datenbanken und den verbundenen Anwendungen. Die gesammelten Informationen werden im Monitorpuffer gespeichert. Für den Zugriff darauf stellen beispielsweise der *Snapshot Monitor*, die *Ereignismonitore* und andere Werkzeuge entsprechende Funktionalitäten bereit (*Abbildung 3.4*).

Der *Snapshot-Monitor*, das Überwachungsprogramm für Momentaufnahmen, kann Informationen zur Datenbank und allen mit ihr verbundenen Anwendungen zu einem bestimmten Zeitpunkt aus dem Monitor-Puffer auslesen. Mit Hilfe dieser Momentaufnahmen kann der Status des Datenbank-Systems ermittelt werden. Werden Momentaufnahmen in regelmäßigen Abständen erstellt, so können mit ihrer Hilfe Trends beobachtet und mögliche Probleme vorhergesehen werden.

Im Gegensatz zum *Snapshot-Monitor* werden die *Ereignis-Monitore* (*Event Monitors*) dazu verwendet, Informationen über die Datenbank und alle verbundenen An-

wendungen beim Auftreten bestimmter Ereignisse aus dem *Monitor-Puffer* auszulesen. Die Ereignisse basieren dabei unter anderem auf den Verbindungen, Deadlocks, Anweisungen oder Transaktionen. Ein *Ereignis-Monitor* definiert, welche Art von Events überwacht werden soll. Ein *Deadlock-Ereignis-Monitor* reagiert etwa (nur) auf Deadlocks und fasst anschließend Informationen über die betroffenen Anwendungen und Sperren, die zum Deadlock geführt haben, zusammen.

Während der *Snapshot-Monitor* hauptsächlich zur Erkennung von Trends und Prävention von Problemen eingesetzt wird, sind die *Ereignis-Monitore* eher dafür geeignet, beim Eintreten von Problemen Informationen zu sammeln und den Administrator zeitnah zu benachrichtigen.

Die Speicherung von gesammelten Daten erfolgt in *Monitor-Elementen*. Jedes *Monitor-Element* speichert Informationen zu einem bestimmten Aspekt des Datenbank-System-Status. Neben einem eindeutigen Namen haben die *Monitor-Elemente* einen bestimmten Informationstyp. Es lässt sich dabei zwischen folgenden Informationstypen unterscheiden:

Zähler (*Counter*). Ein *Zähler* zählt, wie oft eine Aktion stattfand. Die Werte eines *Zählers* steigen während der Überwachung. Als Beispiel für ein Element vom Typ *Zähler* ist `pool_async_data_reads` zu nennen, der die Anzahl asynchron gelesener Seiten in den Bufferpool erfasst.

Wertangabe (*Gauge*, im Deutschen oft als *Pegel* bekannt). Beim Informationstyp *Wertangabe* handelt es sich den aktuellen Wert des entsprechenden Elements. Die Elemente von diesem Typ können hoch- oder heruntergezählt werden. Beispiele für Wertangaben sind die Metriken `pool_hit_ratio` (entspricht der Bufferpool Hitratio) und `locks_held` (zeigt die aktuelle Anzahl gehaltener Sperren an).

Grenzwert (*Water mark*). Ein *Grenzwert* gibt den höchsten (Maximum) oder niedrigsten Wert (Minimum) an, der seit Beginn der Überwachung von entsprechendem Element erreicht wurde. Ein Beispiel für diesen Informationstyp bietet die Metrik `cat_cache_size_top`. Diese gibt die maximal erreichte Größe vom Katalog-Cache an.

Information. Hierbei handelt es sich um weitere Informationen zu den Überwachungsaktivitäten. Dazu zählen beispielsweise die Pfadangaben, Instanznamen etc.

Zeitstempel (*Timestamp*). Ein Informationselement vom Typ *Zeitstempel* gibt den Zeitpunkt (Datum und Uhrzeit) des Auftretens einer Aktivität an. Dabei erfolgt die Zeitangabe in Sekunden und Mikrosekunden, die seit 1. Januar 1970 bis zu dem Zeitpunkt des Auftretens der Aktivität abgelaufen sind.

Zeit (*Time*). Im Gegensatz zu Elementen vom Typ *Zeitstempel*, wo es sich um den Zeitpunkt des Auftretens einer Aktivität handelt, gibt ein Element vom Typ *Zeit* die Dauer einer Aktivität an. Als Beispiel ist an dieser Stelle die Metrik `total_sort_time` zu nennen, die angibt, wie lange alle ausgeführten Sortiervorgänge gedauert haben.

Das Aktivieren von Monitor-Elementen erfolgt mit Hilfe von Monitor-Schaltern. Vor Initialisierung einer Überwachung sollte genau überlegt werden, welche Monitor-Elemente gebraucht werden und die entsprechenden Monitor-Schalter sind zu ak-

tivieren. Dabei ist ein Kompromiss zwischen dem Overhead, falls zu viele Daten gesammelt werden, und dem Nutzen der gesammelten Daten zu treffen.

Monitor-Elemente sammeln Daten einer oder mehrerer logischer Datengruppen. Eine *logische Datengruppe* ist eine Menge von Monitor-Elementen, die Daten für einen bestimmten Bereich der Datenbankaktivitäten sammeln.

Beispiele für die logischen Datengruppen bei der Momentaufnahmeüberwachung sind unter anderem `dbase` (Datenbankebene), `appl` (Anwendungsebene), `stmt` (Anweisungsebene), `bufferpool` (Bufferpoolebene) oder `db_lock_list` (Sperrlistenebene). Einige Monitor-Elemente können gleichzeitig in mehreren Datengruppen enthalten sein. In diesem Fall liefern sie je nach Datengruppe mehr oder weniger aggregierte Informationen, wie es beim Monitor-Element für die Gesamtsortierzeit (`total_sort_time`) der Fall ist, der Informationen auf Datenbankebene, Anwendungsebene und Anweisungsebene auslesen lässt.

Bei Monitor-Elementen von Ereignis-Monitoren handelt es sich unter anderem um folgende logische Datengruppen: `event_bufferpool`, `event_conn`, `event_db`, `event_deadlock` bzw. `event_overflow` für die Eventüberwachung auf Bufferpool-, Daten bankverbindungs-, Datenbank-, Deadlock- bzw. Überlauf-Ebene.

Die Daten der Monitor-Elemente können sowohl in Dateien als auch in SQL-Tabellen gespeichert werden. Anschließend lassen sich die gespeicherten Informationen entweder auf dem Bildschirm anzeigen oder aber auch an Client-Anwendungen zur weiteren Verarbeitung senden [IBM06h].

Diagnose-Monitor. Der *Diagnose-Monitor* (*Health Monitor*) dient einer proaktiven Erkennung von Problemen [IBM06h]. Der Gesundheitszustand des DBMS wird dazu kontinuierlich über Indikatoren überwacht und bei Schwellenwertüberschreitungen Warnungen bzw. Alarme und/oder vordefinierte Aktionen, wie Skripte oder Tasks, ausgelöst.

Während Warnungen bzw. Alarme dazu dienen, den Administrator zu benachrichtigen (Eintrag im *Notification Log*), können die Aktionen dazu beitragen, den Zustand des Systems zu verbessern. Es geht hier also nicht nur darum, das System zu überwachen, sondern auch darum, gegebenenfalls den Zustand des Systems zu verändern, um damit Probleme zu beheben bzw. zu verhindern.

Der Vorteil des *Diagnose-Monitors* liegt darin, dass aufgrund seiner ausnahmegesteuerten Funktionsweise eine permanente Überwachung des ganzen Systems nicht notwendig ist. Basierend auf den *Snapshots* werden nur wenige Informationen den Gesundheitszustand des Systems betreffend gesammelt, was einen geringen Overhead verursacht.

Die Informationen über den Gesundheitszustand des Datenbank-Systems werden mit Hilfe von Gesundheitsindikatoren, den sogenannten *Diagnoseanzeigern*, ermittelt. Diese gibt es auf Instanz-, Datenbank-, Tabellenbereichs- und Tabellenbereichscontainerebene.

Es existieren drei Typen von Diagnoseanzeigern: *schwellenwertbasierte*, *statusbasierte* und *objektgruppenstatusbasierte Diagnoseanzeiger*.

Schwellenwertbasierte Diagnoseanzeiger. Diese Diagnoseanzeiger haben einen fortlaufenden Wertebereich und besitzen die Schwellenwerte „Warning“ und „Alarm“, die diesen Wertebereich in drei Bereiche einteilen. Je nach dem, in welchem Bereich sich der Diagnoseanzeiger befindet, hat er den Zustand „Normal“, „Warning“ oder „Alarm“.

Statusbasierte Diagnoseanzeiger. Diese Indikatoren haben eine endliche Menge von zwei oder mehreren Zuständen eines Datenbankobjekts bzw. einer Datenbankressource, die anzeigen, ob das Objekt/die Ressource einwandfrei funktioniert oder nicht. Ein Zustand wird dabei als normal, alle anderen als nicht normal angenommen. Für diesen Anzeiger sind also zwei Zustände definiert: „Normal“ und „Attention“.

Objektgruppenstatusbasierte Diagnoseanzeiger. Diese Art der Diagnoseanzeiger stellt den zusammengefassten Status eines Objekts oder mehrerer Objekte in der Datenbank dar. Der aggregierte Zustand ergibt sich aus den Einzelzuständen der überwachten Objekte. Befindet sich mindestens ein Objekt in der Gruppe nicht im Zustand „Normal“, so nimmt der Diagnoseanzeiger den Zustand „Attention“ an. Ansonsten nimmt der Indikator den Zustand „Normal“ an.

Die Generierung einer Warnung oder eines Alarms bzw. die Ausführung einer Aktion erfolgt bei einer Statusänderung vom Zustand „Normal“ in einen nicht normalen Zustand wie „Attention“, „Warning“ oder „Alarm“.

SQL-Monitor. Die Mechanismen der *Explain-Einrichtung* (*Explain Facility*) werden oft als *SQL-Monitor* bezeichnet [OGT04]. Mit diesen Mechanismen ist es möglich festzustellen, wie der Datenbankmanager auf Tabellen und Indizies zur Beantwortung einer SQL-Anfrage zugreift, also welchen Zugriffsweg der DB2-Optimizer für diese SQL-Anfrage gewählt hat. Die Zugriffspläne werden dabei in *Explain-Tabellen* gesammelt und stehen für eine Auswertung mittels *Explain-Tools* [IBM06f, IBM06e] wie *Visual Explain* oder *db2expln* zur Verfügung.

Explain-Informationen sind hilfreich, um die Performance von SQL-Anweisungen zu verschiedenen Zeitpunkten (vor und nach der Durchführung von Änderungen am System) miteinander zu vergleichen. Diese Änderungen können beispielsweise das Hinzufügen oder Löschen von Indizes, Aktualisieren der Statistikinformationen, Hinzufügen von *Materialized Query Tables (MQT)* oder Änderungen an den SQL-Anweisungen selbst sein.

Protokolldateien. Eine weitere Informationsquelle für die Problemdiagnose bieten die *Protokolldateien*. Beim Auftreten von DB2-Fehlern wird nach dem *First Failure Data Capture*-Konzept [IBM06i] ein Satz von Diagnoseinformationen automatisch erfasst. Diese Informationen verringern die Notwendigkeit, Fehler zur Beschaffung von Diagnoseinformationen zu reproduzieren. Zu den wichtigen Protokolldateien zählen das *DB2-Diagnoseprotokoll* (*db2diag.log*), die *Systemkerndateien* (*core files*), *Speicherauszugsdateien* (*dump files*) sowie *Trapdateien* (*trap files*):

DB2-Diagnoseprotokolldatei. Die *DB2-Diagnoseprotokolldatei* enthält Diagnoseinformationen zu den in der Instanz festgestellten Fehlern und Warnungen.

Systemkerndatei. Eine *Systemkerndatei* wird durch das System erstellt, wenn ein Programm abnormal (beispielsweise aufgrund Speicheradressverletzungen oder un-

zulässiger Instruktionen) beendet wird. In dieser Datei wird ein Speicherabbild des beendeten Prozesses gespeichert.

Speicherauszugsdatei. *Speicherauszugsdateien* werden erstellt, wenn ein Fehler auftritt, für den zusätzliche Informationen, die bei der Problemdiagnose nützlich sein könnten, verfügbar sind.

Trapdatei. *Trapdateien* werden von DB2 erstellt, wenn aufgrund einer Fehlerunterbrechung (*Trap*), einer Segmentierungsverletzung oder einer Ausnahmebehandlung die DB2-Verarbeitung nicht fortgesetzt werden kann. Die entsprechenden Ausnahmebedingungen, der Funktionsaufrufstack zum Zeitpunkt des Fehlers sowie Informationen zum Status des Prozesses werden dabei in einer *Trapdatei* abgespeichert.

Tracing. Um das Anwendungsverhalten zu analysieren, werden *Traces* verwendet. Diese zeichnen beispielsweise von Anwendungen die Reihenfolge der Aufrufe und die dabei übergebenen Argumente auf. Anhand von Zeitstempeln der Aufrufe lässt sich im Nachhinein feststellen, wo wieviel Zeit verbraucht wurde [IBM06i]. Der mit dem Tracing verbundene Overhead ist allerdings nicht zu verachten, da dadurch die Datenmenge rapide anwächst.

Anwendungen, die über *SQLJ*, *JDBC*, *ODBC* oder das *CLI* mit *DB2 UDB* kommunizieren, können das Tracing aktivieren. Über SQLJ und JDBC besteht die Möglichkeit, sich die Zeitinformationen nach Anwendungs-, Server-, Netzwerk- und Treiberzeit aufzuschlüsseln. Zu beachten ist allerdings, dass SQLJ, JDBC und ODBC die Aufrufe an das CLI weiterleiten. Ist beispielsweise sowohl das JDBC-Tracing als auch das CLI-Tracing aktiviert, werden die Aufrufe „doppelt“ aufgezeichnet. Teilweise kann genau dieses Verhalten erwünscht sein, jedoch verursacht es auch „doppelten“ Overhead [Hen07].

Problem Determination Tool. Das *Problem Determination Tool* (*db2pd*) ist ein Fehlerbestimmungstool, das schnell und direkt Informationen aus dem DB2-Speicher ausliest, ohne dabei Sperrmechanismen oder Ressourcen von DB2 zu beanspruchen. Dadurch kann es allerdings passieren, dass geänderte Pointer festgestellt werden. In solchen Fällen wird mittels einer Signalroutine verhindert, dass *db2pd* abnormal beendet wird. Das Werkzeug ist für die Diagnose nützlich, da es sehr performant arbeitet und vor allem keine Ressourcen beansprucht.

In [Ree05] wird beispielsweise an konkreten Beispielen gezeigt, wie sich dieses Tool zur Performance-Problem-Analyse einsetzen lässt. In [Eat07a, Eat07b, Eat07c, Eat07d, Eat07e, Eat07f] finden sich Beispiele zum Einsatz dieses Werkzeugs zur Überwachung der Bufferpools, Tabellenzugriffen, Log-Auslastung und des Sperrverhaltens.

3.3.2.2 DB2-externe Werkzeuge zur Erkennung und Analyse von Datenbankproblemen

Aufbauend auf den oben genannten DBMS-internen Werkzeugen lassen sich auch DBMS-externe Werkzeuge für die Problemerkennung einsetzen. Diese haben einerseits den Vorteil, dass sie eine bessere Aufbereitung der Überwachungsdaten und gegebenenfalls eine Konsolidierung von Informationen verschiedener Quellen wie DBMS, Anwendungsserver oder

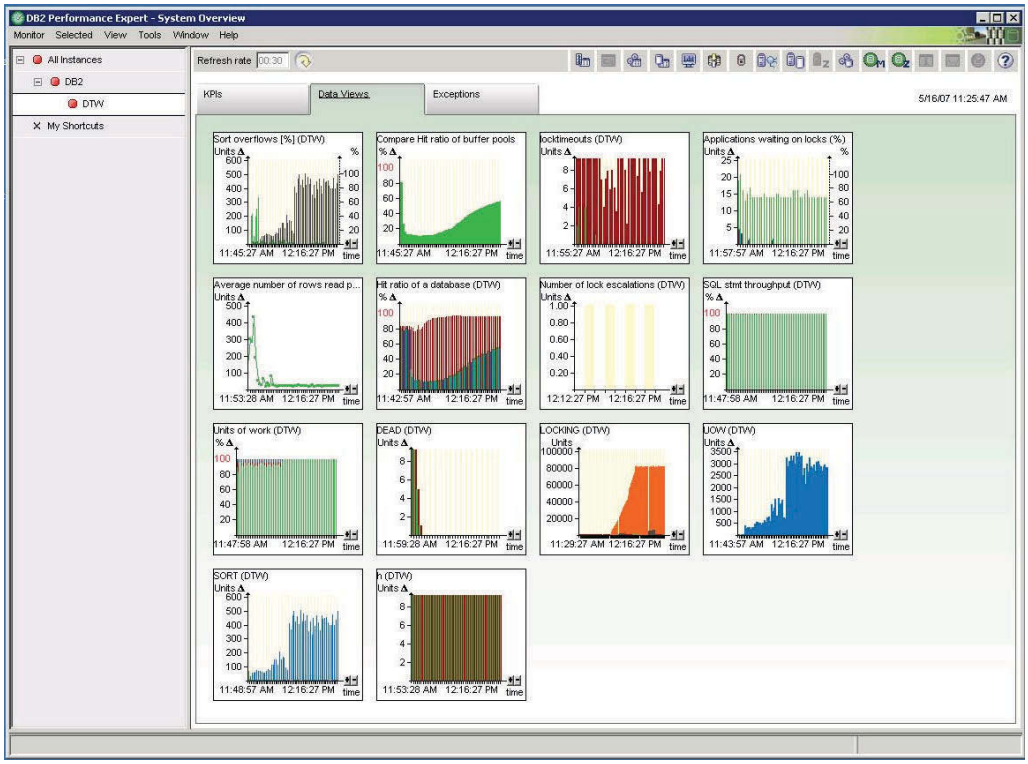


Abbildung 3.5: Performance Expert – Visualisierung von Performance-Metriken

Betriebssystem ermöglichen. Andererseits verursachen sie unter Umständen einen höheren Overhead.

Nachfolgend werden das eigens für *DB2 UDB* entwickelte Überwachungswerkzeug *IBM DB2 Performance Expert* sowie das weitverbreitete universelle Datenbanküberwachungswerkzeug *IBM Tivoli Monitoring for Databases* exemplarisch vorgestellt.

IBM DB2 Performance Expert

IBM DB2 Performance Expert (PE) ist ein Tool zur Überwachung und Analyse der Performance von Datenbanken. Es ermöglicht einen umfassenden Überblick über Leistungsinformationen eines DB2-Datenbank-Systems sowie des zugrunde liegenden Betriebssystems und stellt Funktionalitäten zur Protokollierung, Sicherung und Analyse von Performancedaten bereit. Dabei bietet PE vier Ebenen der Leistungsüberwachung: Echtzeitüberwachung, Kurrzeitüberwachung, Langzeitüberwachung und Ausnahmeverarbeitung [CBMW06].

Echtzeitüberwachung. Die *Echtzeitüberwachung* ermöglicht eine Überwachung des laufenden Betriebs eines DB2-Systems und des zugrunde liegenden Betriebssystems. Die erfassten Daten können dabei entweder in Form von Tabellen oder graphisch dargestellt werden. ***Abbildung 3.5*** veranschaulicht eine graphische Darstellung von ausgewählten Performance-Metriken.

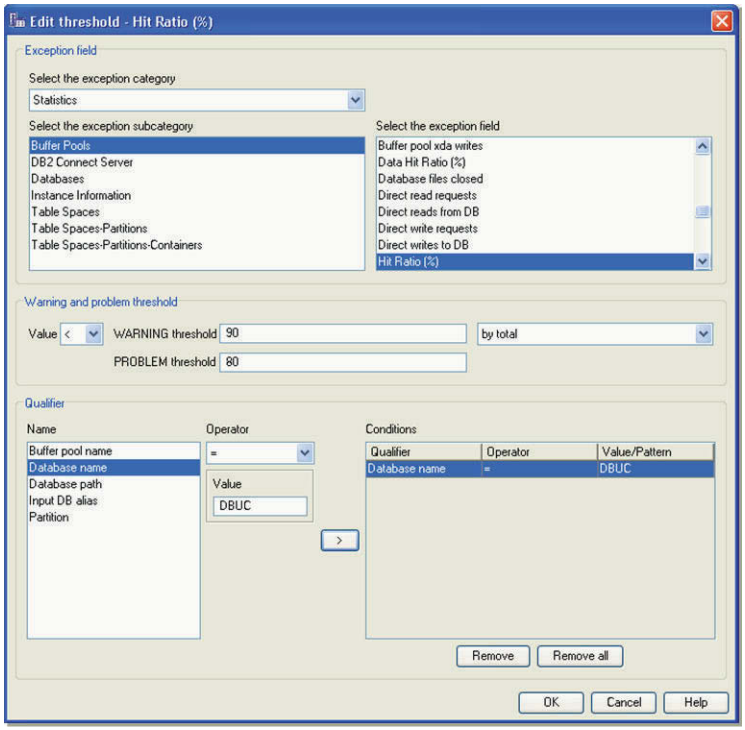


Abbildung 3.6: Performance Expert – Schwellenwertdefinition für die *Bufferpool Hitratio*

Kurzzeitüberwachung. Mit Hilfe einer *Kurzzeitüberwachung* wird der Zugriff auf die in einer „kurzen“ Zeitspanne (Minuten, Stunden, Tage) gesammelten Performance-Daten ermöglicht. Diese Daten können ausgewertet und Deadlocks, langlaufende SQL-Anweisungen, *Timeouts*, Sperrerskalationen etc. erkannt werden. Wie lange die gesammelten Daten gespeichert werden, kann vom Administrator konfiguriert werden. Ähnlich wie bei der Echtzeitüberwachung können die erfassten Daten in Tabellenform oder graphisch dargestellt werden, um im Nachhinein Probleme zu diagnostizieren oder Trends zu erkennen.

Langzeitüberwachung. Die *Langzeitüberwachung* dient der Sicherung von Daten über einen langen Zeitraum. Die Datenspeicherung erfolgt dabei in einem sogenannten *Performance Warehouse*. Die gesammelten Daten können für Trendanalysen oder spätere Auswertungen genutzt werden. Eine Trendanalyse ermöglicht nicht nur das Sammeln von Erkenntnissen, wie sich das System während des normalen bzw. Hochbetriebs verhält, sondern unterstützt den Administrator auch dabei, sich entwickelnde Probleme proaktiv zu erkennen und vorzubeugen.

Ausnahmeverarbeitung. Die *Ausnahmeverarbeitung* (*Exception Processing*) ermöglicht eine reaktive bzw. proaktive[8] Überwachung von Datenbanken. Dabei wird zwischen zwei Arten des *Exception Processings* unterschieden: *Event Exception Processing* und *Periodic Exception Processing*.

[8] Die proaktive Überwachung kann mittels vorgelagerter Schwellenwerte umgesetzt werden, indem Warnungen kurz vor Erreichen des kritischen Zustands generiert werden.

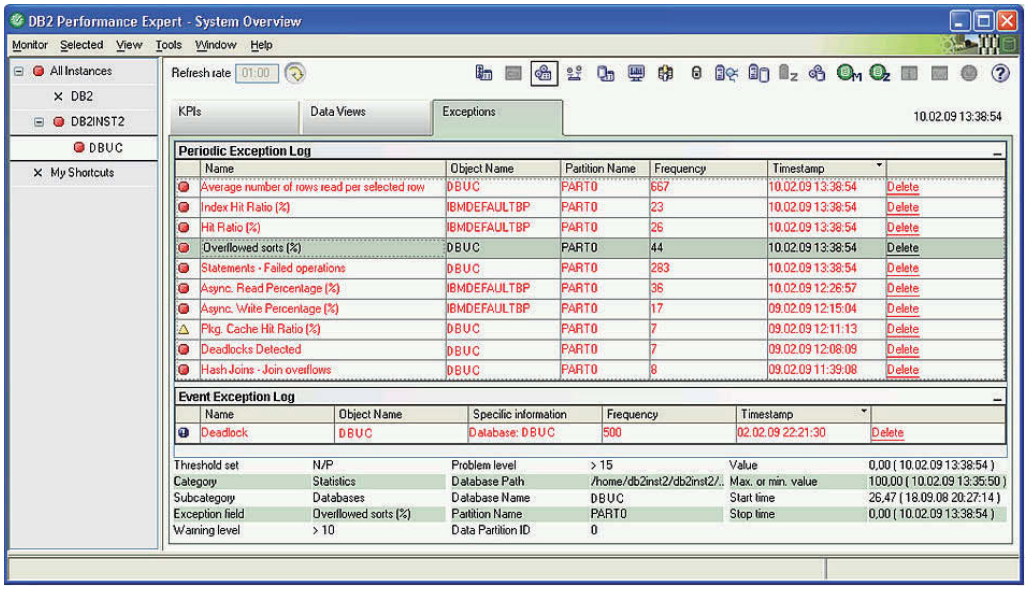


Abbildung 3.7: Performance Expert – Darstellung von Exceptions

Während das *Event Exception Processing* auf dem *Ereignis-Monitor* des DB2 aufbaut und aktuell ausschließlich die Erkennung von *Deadlock Events* ermöglicht, basiert das *Periodic Exception Processing* auf dem *Snapshot-Monitor* und kann somit für die Überwachung einer Vielzahl von Performance-Metriken eingesetzt werden (vgl. *Abschnitt 3.3.2.1*). Beim *Periodic Exception Processing* können vor- bzw. selbstdefinierte Schwellenwerte für Metriken verwendet werden. Beim Über- bzw. Unterschreiten dieser Schwellenwerte können Exceptions in Form von Warn- bzw. Problemmeldungen (*Warning* bzw. *Problem*) generiert werden. ***Abbildung 3.6*** enthält ein Beispiel zur Schwellenwertdefinition. Es erfolgt hier die Definition einer Exception aufbauend auf der Bufferpool-Hitrate (*Bufferpool Hitratio*). Fällt diese unter den Warning-Schwellenwert von 90%, so wird eine Warnmeldung generiert. Fällt die Bufferpool Hitratio aber weiter unter 80%, so wird eine entsprechende Problemmeldung generiert. Neben den Schwellenwerten müssen dabei weitere Metadaten, wie beispielsweise der Name der überwachten Datenbank oder der Name des Bufferpools, angegeben werden.

Die im Rahmen des *Exception Processing* erzeugten Exceptions können über das GUI des PE-Clients ausgegeben werden. Ein Beispiel für die Darstellung von *Periodic* und *Event Exceptions* ist in ***Abbildung 3.7*** zu finden. Neben dem Namen der erkannten Exceptions werden dabei weitere Metadaten wie die verletzten Schwellenwerte sowie die entsprechenden Zeitangaben angezeigt.

Weiterhin können die erzeugten Exceptions an E-Mail-Adressen gesendet oder an eine spezielle Routine (*User Exit*) übergeben werden. Diese Routine ermöglicht den Austausch von Exceptions mit anderen Anwendungen oder eine automatische Ausführung von Aktionen[9].

[9]Diese Funktionalität findet bei der prototypischen Umsetzung des *Autonomic Tuning Expert* (*Abschnitt 5.2*) Anwendung.

IBM Tivoli Monitoring for Databases

Während der im vorangegangenen Abschnitt vorgestellte *IBM DB2 Performance Expert* sich auf die Überwachung der Performance-Kennzahlen des Datenbank-Systems sowie des zugrunde liegenden Betriebssystems konzentriert, ermöglicht die Tivoli-Software[10] von IBM die Verwaltung, Überwachung und Optimierung der gesamten IT-Landschaft. Zur umfangreichen Produktpalette von Tivoli zählen unter anderem Werkzeuge zur durchgängigen Netzwerk- und Systemüberwachung, zur automatischen Software-Verteilung bzw. zum Service-Management. Die für die einzelnen Problemstellungen konzipierten Komponenten können in die Tivoli-Gesamtinfrastruktur integriert werden, so dass eine komponentenübergreifende Funktionsweise ermöglicht wird. Insbesondere kann davon auch die Eventdefinition bzw. -verarbeitung profitieren.

Speziell auf die Überwachung von Datenbank-Systemen spezialisiert sich die Tivoli-Komponente namens *IBM Tivoli Monitoring for Databases*[11] [IBM08c]. Hier wird genauso wie in anderen Systemüberwachungsprodukten begrifflich zwischen *Situationen* und *Ereignissen* unterschieden. Als Situationen werden Überwachungsspezifikationen bzw. Bedingungsdefinitionen bezeichnet (z. B. „pool_hit_ratio $<$ 50%“). Diese werden den einzelnen überwachten Systemen zugeordnet und von der Tivoli-Infrastruktur periodisch ausgewertet [IBM08b]. Dabei wird anhand der mit Hilfe von Überwachungsagenten auf den entsprechenden verwalteten Systemen erfassten Werte überprüft, ob die Situationsbedingung erfüllt ist und, falls dies der Fall ist, ein entsprechendes Ereignis generiert.

Die einzelnen Überwachungsagenten stellen Sätze an vordefinierten Situationen bereit. Zu den DB2-spezifischen vordefinierten Situationen von *Tivoli Monitoring for Databases* zählen beispielsweise folgende [IBM07]:

UDB_Appl_BP_Hit_Ratio_Low. Ausgabe einer Warnung, wenn Bufferpool Hitratio für eine Anwendung unter 50% fällt.

UDB_Appl_CatCache_Hit_Low. Ausgabe einer Warnung, wenn die Katalogcache-Hitratio einer Anwendung unter 50% fällt.

UDB_Database_Lock_Warning. Ausgabe einer Warnung, wenn auf die überwachte Datenbank mindestens eine der folgenden Bedingungen zutrifft:

- Mehr als 10 Deadlocks
- Mehr als 10 Überschreitungen des Zeitlimits für Sperren (*lock timeouts*)
- Mehr als 20 Wartezustände für Sperren (*lock waits*)

UDB_DB_Sort_Overflow_High. Ausgabe einer Warnung, wenn bei einer überwachten Datenbank mehr als 30% an Überläufen bei Sortiervorgängen (*sort overflows*) auftreten.

UDB_DB_SQL_Fail_High. Ausgabe einer Warnung, wenn bei einer überwachten Datenbank mehr als 40% der SQL-Anweisungen fehlschlagen.

[10] http://www.ibm.com/software/de/tivoli/

[11] *IBM Tivoli Monitoring for Databases* ist inzwischen zu einem Bestandteil von *IBM Tivoli Composite Application Manager for Applications* geworden.

Die vordefinierten Situationen basieren auf den bewährten Schwellenwerten. Offensichtlich ist jedoch, dass unter Umständen die Notwendigkeit bestehen kann, diese Werte anzupassen bzw. eigene Situationen zu definieren. Dazu bietet die Tivoli-Umgebung entsprechende Werkzeuge.

Bei der Definition von eigenen Situationen lassen sich Teilausdrücke mittels der booleschen Operatoren *AND* und *OR* miteinander verknüpfen. In die Situationsdefinitionen lassen sich neben den klassischen Attributvergleichen auch andere Situationen einbinden. Weiterhin lassen sich Einstellungen bezüglich der Häufigkeit der Bedingungsauswertung (*sampling interval*) bzw. der Dauer des Vorliegens der Situation, bis der entsprechende Event generiert wird (*situation persistence*), vornehmen [IBM08b].

Die Mächtigkeit der Situationsbeschreibung ist hier offenbar deutlich höher als bei der Event-Definition in *IBM DB2 Performance Expert*. Dennoch lassen sich Beispiele finden, in denen die angebotene Mächtigkeit der Situationsbeschreibung nicht ausreicht. Abhilfe schafft hier die Verarbeitung komplexer Ereignisse (*Abschnitt 3.4*), die eine Zusammenfassung der in diesem Abschnitt vorgestellten atomaren Events und somit eine ereignisübergreifende Auswertung ermöglicht.

Der Hauptanwendungszweck der *Tivoli-Monitoring*-Produktfamilie ist die Bereitstellung einer Administrationskonsole, die den Zugriff auf die wichtigen den Zustand des überwachten Systems repräsentierenden Kennzahlen gewährt und dem Administrator weiterhin das Anstoßen von Administrationsroutinen ermöglicht. Die Tivoli-Infrastruktur stellt jedoch auch Mechanismen bereit, um auf das Auftreten von Events automatisch mit Aktionen reagieren zu können. Dazu werden zwei Stufen der Automatisierung umgesetzt.

Die *Reflex-Automatisierung* ermöglicht beim Auftreten eines Events die Ausführung einer Anweisung bzw. eines Skripts anzustoßen. Das Ergebnis der Ausführung wird dabei nicht überwacht und kein Feedback diesbezüglich wird zurückgeliefert.

Mit den Mechanismen der *erweiterten Automatisierung* lassen sich dagegen komplexe Administrationsabläufe (bezeichnet als *Policies*) umsetzen. Nach der Ausführung eines solchen Administrationsablaufs wird ein *Return Code* zurückgeliefert. Die Überwachungsagenten stellen bereits vordefinierte Administrationsabläufe bereit, der Nutzer hat jedoch auch die Möglichkeit, eigene Administrationsabläufe umzusetzen. Weitere Ausführungen zu der Mächtigkeit der erweiterten Automatisierung von Tivoli finden sich in *Abschnitt 4.1.3*.

3.3.3 Nutzerinitiierte Events

Die Ansätze zur automatischen Problemerkennung schlagen häufig fehl, so dass ca. 50% der durch die Administratoren zu lösenden Probleme von den Endnutzern erkannt und an die zuständigen Administratoren weitergeleitet werden [GBS+05]. Sichtbar für den Endnutzer werden die Probleme typischerweise dabei nicht nur, wenn die Ausführung der Anwendungen fehlschlägt, sondern auch, wenn Abweichungen vom typischen Anwendungsverhalten festgestellt werden. Zu solchen Abweichungen zählt beispielsweise die plötzlich steigende Antwortzeit einer bestimmten Anwendung.

Die Systemantwortzeit ist ein wichtiger Indikator für die Performance eines Systems. Sie ist definiert als die Zeit, die zwischen dem Absetzen einer Anfrage durch den Nutzer

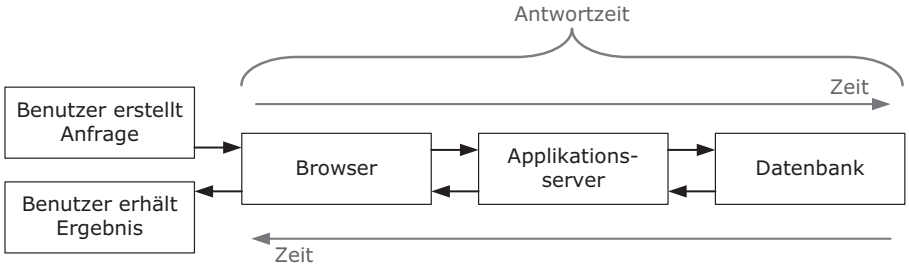


Abbildung 3.8: Zusammensetzung der Antwortzeit [Aut07]

(oder ein Computersystem) und dem Zeitpunkt des Zurücklieferns des Ergebnisses vergeht [MH03, HN01]. Die Antwortzeit setzt sich dabei aus den Einzelzeiten für die Bearbeitung der Anfrage bzw. des Anfrageergebnisses im Browser bzw. auf dem Anwendungsserver sowie der innerhalb der Datenbank für Nutzerprozesse aufgewendeten Zeit (Datenbankzeit) [DRS+05] zusammen (***Abbildung 3.8***).

Die meisten Datenbankperformance-Probleme wirken sich negativ auf die Datenbankzeit und somit die Antwortzeit aus. Dabei setzen sich die Auswirkungen eines Performance-Problems von der ursprünglich von dem Performance-Problem betroffenen Ressource über alle Ressourcen, die mit dieser Ressource kommunizieren bzw. diese benutzen, fort.

Dieser Sachverhalt lässt sich an einem vereinfachten Beispiel der Anfragebearbeitung in DB2 verdeutlichen. Bei der Bearbeitung einer von der Datenbankanwendung abgesetzten Anfrage werden zahlreiche Datenbank-System-Komponenten durchlaufen und verschiedene Ressourcen verwendet (***Abbildung 3.9***). Das Fehlverhalten einer der Komponenten bzw. Ressourcen hätte Auswirkungen auf den gesamten Prozess der Anfragebearbeitung und würde sich anschließend in einem von dem Endbenutzer wahrgenommenen Problem äußern: Ein fehlender Index würde also in einer längeren Bearbeitungszeit einer entsprechenden Anfrage resultieren, was zu einer längeren Wartezeit des Nutzers auf das Anfrageergebnis führt.

Grundsätzlich lässt sich ein Performance-Problem als eine nicht tolerierbare Abweichung des Ist-Zustands von einem geforderten Soll-Zustand definieren. Der Zustand eines Systems lässt sich dabei durch eine Menge von Performance-Indikatoren (Performance-Metriken) beschreiben. Zu solchen Performance-Metriken zählen einerseits für den Endnutzer verständliche, abstrakte Maße wie die Angaben bezüglich der Antwortzeiten, der Durchsatzraten bzw. der Systemverfügbarkeit. Andererseits können die Performance-Metriken auf einer tieferen Ebene beispielsweise in Form von Angaben bezüglich der Auslastungsgrade einzelner Komponenten, Füllgrade der Speicherbereiche bzw. des Verhältnisses von angefragten und im Cache vorgefundenen Datenseiten zu der Gesamtanzahl von angefragten Datenseiten (*Cache Hitratio*) beschrieben werden.

Es besteht offensichtlich eine Abhängigkeit zwischen den auf hoher Ebene definierten abstrakten und den auf tieferer Ebene spezialisierten Performance-Metriken. Da die Plattenzugriffe deutlich langsamer als die Cache-Zugriffe sind, impliziert beispielsweise eine niedrige Cache Hitratio eine längere Bearbeitungszeit der Anfrage und somit eine niedrigere Durchsatzrate bzw. eine höhere Antwortzeit.

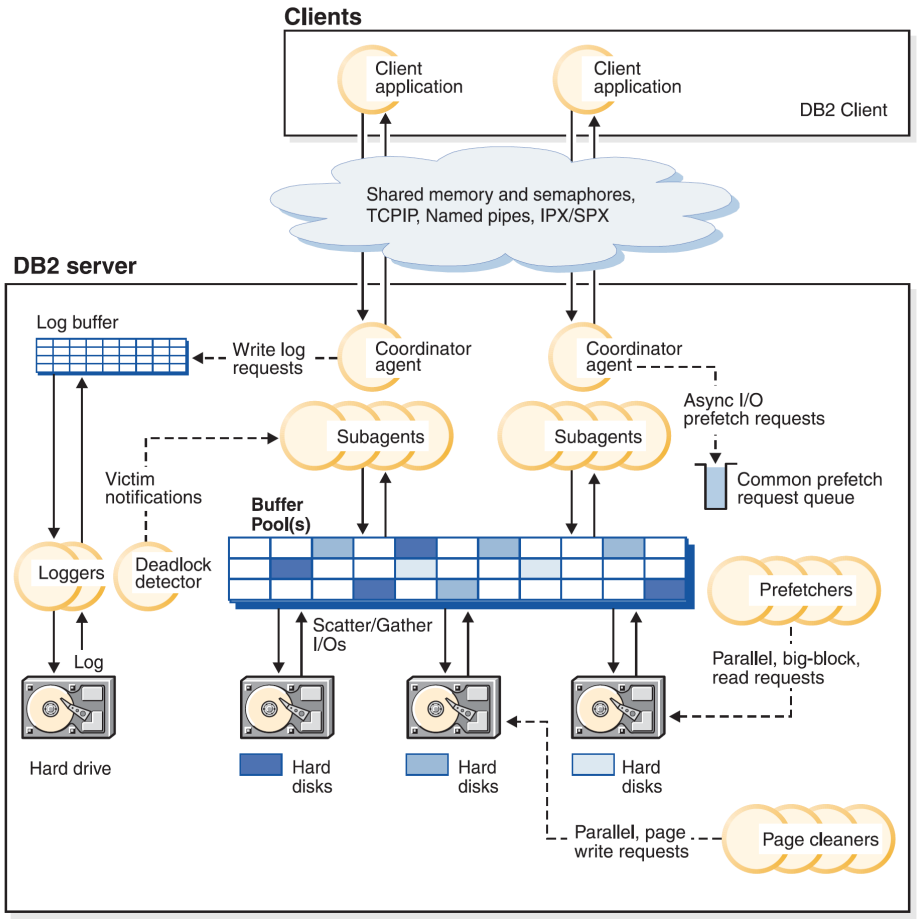


Abbildung 3.9: *IBM DB2 UDB* – Architektur- und Prozessüberblick [IBM06f]

Wie eingangs erwähnt, spielen neben den automatisch generierten sensorbasierten Events die durch die Nutzer erkannten Probleme eine wichtige Rolle beim Tuning von IT-Systemen. Typischerweise vergeht jedoch einige Zeit, bis ein Problem durch den Nutzer erkannt, dem zuständigen Administrator mitgeteilt und anschließend behoben werden kann. Es besteht daher die Notwendigkeit für die Entwicklung von Mechanismen, die eine automatische Erkennung von nutzerinitiierten Events und somit eine schnelle Problemauflösung ermöglichen. Dabei lassen sich beispielsweise Data-Mining-Verfahren einsetzen, um basierend auf den Performance-Metriken den normalen Systemzustand zu erlernen und bei starken Abweichungen von diesem, noch bevor der Benutzer diese Abweichungen bemerkt, entsprechende Events zu generieren. Weitere Betrachtungen dieses Ansatzes sind jedoch nicht Gegenstand dieser Arbeit.

3.4 Komplexe Events

Zur Reduzierung der Eventflut durch die Zusammenfassung ähnlicher Events sowie zur Berücksichtigung des Problemkontexts greifen wir auf die Konzepte der Verarbeitung kom-

plexer Ereignisse zurück und setzen die ankommenden primitiven Events mit Hilfe von Eventoperatoren zu komplexen Events zusammen.

Dabei lassen sich nicht nur die nicht interessierenden Events eliminieren bzw. aggregieren. Wichtige Events können auch mit Informationen zum Kontext des aufgetretenen Problems angereichert werden. Zu solchen Informationen, die einen Problemkontext beschreiben, zählen beispielsweise die Informationen über die Systemumgebung, über die ausgeführten Anwendungen bzw. über die aktiven Benutzer.

Den nachfolgenden Ausführungen bezüglich der Operatoren zur Bildung komplexer Events (*Abschnitt 3.4.1*) sowie der exemplarischen Vorstellung von Konzepten zur Ereignisverarbeitung (*Abschnitt 3.4.2*) liegen unter anderem [DG00, EN10, Jäg08] zugrunde.

3.4.1 Eventoperatoren

In Anlehnung an [EN10] lassen sich die Operatoren der Eventverarbeitung, wie in ***Abbildung 3.10*** angegeben, kategorisieren. Die Auswahl der nachfolgend aufgeführten und in die entsprechenden Kategorien eingeordneten Eventverarbeitungsoperatoren beschränkt sich auf die wichtigsten, in der Praxis weit verbreiteten Operatoren.

Filterung. Diese Operatoren ermöglichen die Eliminierung nicht interessierender Events anhand einer entsprechenden Filterbedingung. Insbesondere bei Event-Erzeugern, die eine hohe Anzahl an Events in kurzer Zeit produzieren, ist die Verwendung von Filterungsoperatoren sinnvoll, um die Eventflut zu reduzieren. Typischerweise ist der Filterungsoperator *zustandslos* und die Auswertung der Filterungsbedingung erfolgt ausschließlich basierend auf dem Eventinhalt. Ein Beispiel hierfür ist ein Test, ob der Zeitpunkt des Eventauftretens in einem bestimmten Intervall liegt oder nicht. Nur wenn das der Fall, wird der Event an den entsprechenden Event-Konsumenten durchgelassen.

In einigen Fällen können jedoch auch *zustandsbehaftete Filterungsoperatoren* definiert werden. Zu solchen zustandsbehafteten Filterungsoperatoren zählen beispielsweise folgende Operatoren: *First m*, *First m out of n*, *Last m* sowie *Last m out of n*. Diese Operatoren werten den eingehenden Eventstrom aus während eines bestimmten Zeitfensters bzw. bis eine bestimmte Schranke für die Anzahl der ausgewerteten Events erreicht ist und ermitteln die ersten bzw. die letzten m Events im betrachteten Intervall. Anhand von zusätzlichen Attributbedingungen lassen sich die Attributwerte der Events in die Filterung miteinbeziehen. So können beispielsweise nur Events mit bestimmten Eigenschaften gezählt werden.

Bei der Ermittlung der ersten m Events können die einzelnen Events gleich nach ihrem Eintreffen weitergegeben werden, während für die Ermittlung der letzten m Events zunächst bis zum Ende des betrachteten Intervalls abgewartet werden muss und die während des Zeitintervalls angekommenen Events zwischengespeichert werden müssen. Die bei der Filterung ermittelten Events werden anschließend an die entsprechenden Event-Konsumenten weitergeleitet. Die restlichen Events werden dagegen herausgefiltert. Je nach Umsetzung des Filterungsoperators können die herausgefilterten Events entweder gelöscht oder an einen speziell für die herausgefilterten Events prädestinierten Event-Konsumenten übergeben werden.

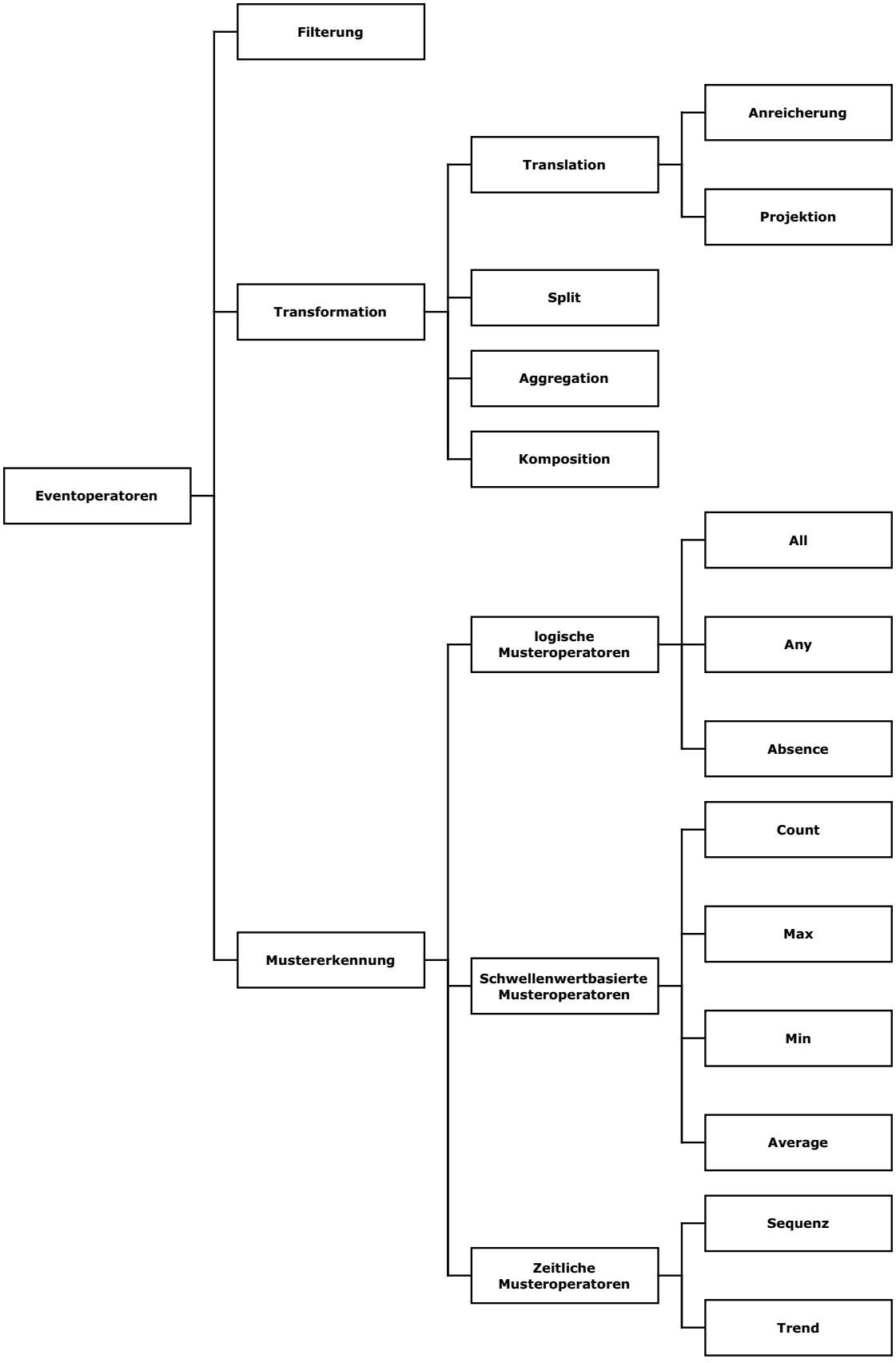


Abbildung 3.10: Kategorisierung von Eventoperatoren

Transformation. Die Transformationsoperatoren verändern den Inhalt der eingehenden Events. In Abhängigkeit von der Kardinalität des Inputs bzw. Outputs dieser Operatoren und davon, welche Art von Veränderungen dabei stattfindet, lassen sich die Transformationsoperatoren weiter in *Translation*, *Split*, *Aggregation* und *Komposition* unterteilen.

Translation. Ohne Berücksichtigung der Vorgänger- bzw. Nachfolgerevents sorgen die Translationsoperatoren für eine Veränderung der Inhalte der eingehenden Events. Zu jedem eingehenden Event wird dabei ein ausgehender Event generiert. Bei der Generierung ausgehender Events können bestimmte Informationen entfernt (*Projektion*) bzw. hinzugenommen (*Anreicherung*) oder modifiziert werden. Bei den nachfolgend erklärten Operatoren für die Event-Anreicherung bzw. -Projektion handelt es sich um häufig in der Praxis vorkommende Spezialfälle des Translationsoperators.

- **Anreicherung.** Durch die Anreicherung von Events lassen sich ihre Inhalte um zusätzliche Informationen erweitern. Zu einem mittels eines eingehenden Events angedeuteten Problem lassen sich beispielsweise entsprechende Behandlungsmaßnahmen aus einer Datenbasis auslesen und in den Event einpflegen.
- **Projektion.** Mit Hilfe dieses Operators lassen sich bestimmte Informationen aus dem eingehenden Event ausblenden. Der ausgehende Event enthält somit alle Informationen aus dem eingehenden Event unter der Ausnahme der ausgeblendeten Informationen.

Split. Dieser Operator erzeugt aus einem eingehenden Event mehrere ausgehende Events, die beispielsweise für verschiedene Event-Konsumenten gedacht sein können. Dabei lässt sich beispielsweise ein Event mit umfangreichem Inhalt auf mehrere Events mit jeweils einem relevanten Ausschnitt an Daten aufteilen und die erzeugten Events an die jeweils zuständigen Event-Konsumenten verteilen. Auch die mittels des Kompositionsoperators zusammengesetzten Events können mittels des Split-Operators wieder in einzelne Events aufgeteilt werden.

Aggregation. Mit Hilfe eines Aggregationsoperators lassen sich mehrere Events eines Eventstroms zu einem einzelnen Event zusammenfassen. Bei der Erzeugung eines neuen Events können verschiedene Aggregationsfunktionen wie beispielsweise *min*, *max*, *average*, *count* oder *concat* auf die Attribute eingehender Events angewendet werden.

Komposition. Während der Aggregationsoperator auf Events eines einzelnen Eventstroms angewendet wird, ermöglicht der Kompositionsoperator die Zusammenführung mehrerer Eventströme zu einem einzigen Eventstrom. Ansonsten ist die Funktionsweise des Kompositionsoperators in gewissem Umfang mit der Funktionsweise des Aggregationsoperators vergleichbar.

Mustererkennung (Pattern Detection). Die Mustererkennungsoperatoren analysieren die eingehende Menge von Events und überprüfen, ob eine Teilmenge der Events einem vorgegebenen Muster entspricht und somit die Musterbedingung erfüllt. Ist dies der Fall, so wird ein entsprechender komplexer Event (Datensatz) generiert, welcher unter Umständen alle der Musterbedingung entsprechenden Events oder aber auch andere Inhalte enthalten kann.

Es kann unter anderem zwischen *logischen*, *schwellenwertbasierten* und *zeitlichen* Musteroperatoren unterschieden werden.

Logische Musteroperatoren. Diese Operatoren sorgen für die Umsetzung verschiedener logischer Funktionen wie die Konjunktion, Disjunktion bzw. Negation.

- **All.** Beim Event-Muster *All* handelt es sich um eine Umsetzung der logischen Konjunktion. Das *All*-Muster ist erfüllt, wenn jedes der in der Parameterliste des Operators aufgeführten Events in einem vordefinierten Zeitfenster tatsächlich mindestens einmal aufgetreten ist. Die Reihenfolge des Auftretens ist dabei nicht von Bedeutung.
- **Any.** Dabei handelt es sich um eine Umsetzung der logischen Disjunktion. Tritt also mindestens eins der Events, die in der Operatorparameterliste angegeben sind, auf, so gilt das *Any*-Muster als erfüllt.
- **Absence.** Das *Absence*-Muster wird erkannt, wenn keines der aufgelisteten Events in einem vorgegebenen Zeitintervall aufgetreten ist. Es handelt sich hierbei also um die Umsetzung einer Negation.

Schwellenwertbasierte Musteroperatoren. Bei diesen Musteroperatoren wird zunächst eine Aggregationsfunktion auf eine Menge von Events bzw. ihren Attributen angewendet. Anschließend wird der ermittelte Wert mit einem vorgegebenen Schwellenwert verglichen. Bei Erfüllung der Vergleichsbedingung gilt das Muster als erfüllt.

- **Count.** Mit diesem Operator wird die Anzahl von Event-Instanzen eines bestimmten Typs gezählt.
- **Max.** Der *Max*-Operator ermittelt den maximalen Wert eines bestimmten Eventattributs aus einer Menge von Events.
- **Min.** Ähnlich wie der *Max*-Operator ist der *Min*-Operator für die Bestimmung des kleinsten Attributwerts zuständig.
- **Average.** Dieser Operator wertet alle Werte eines bestimmten Eventattributs aus einer Menge von Events aus und ermittelt daraus einen Durchschnittswert.

Zeitliche Musteroperatoren. Die zeitlichen Musteroperatoren ermöglichen die Berücksichtigung des zeitlichen Aspekts bei der Ereigniskorrelation. Der weit verbreitetste Vertreter dieser Klasse von Operatoren ist der Sequenzoperator. Aber auch der Trendoperator findet in Ereignisverarbeitungsanwendungen oft Anwendung.

- **Sequenz.** Der *Sequenz*-Operator ähnelt dem *All*-Operator mit dem Unterschied, dass hier die Reihenfolge des Eventauftretens eine wichtige Rolle spielt. Das Sequenz-Muster ist erfüllt, wenn unter Berücksichtigung eines vordefinierten Zeitfensters jeder der in der Parameterliste enthaltenen Events in der vorgegebenen Reihenfolge mindestens einmal aufgetreten ist.
- **Trend (steigend, fallend, stabil, gemischt).** Betrachtet man den Werteverlauf eines bestimmten Attributs über einen vorgegebenen Zeitraum, so lässt sich eine Aussage bezüglich des Trends dieses Werteverlaufs treffen. Mit Hilfe des Trendmusters kann eine Auswertung dieses Trends erfolgen. Entspricht der erkannte Werteverlaufstrend dem vorgegebenen Trendmuster (steigend, fallend,

Operatorart	Zustandslos	Zustandsbehaftet
Filterung	+	+
Transformation	+	+
Translation	+	−
Anreicherung	+	−
Projektion	+	−
Split	+	−
Aggregation	−	+
Komposition	−	+
Mustererkennung	−	+

Tabelle 3.1: Zustandslose und zustandsbehaftete Eventoperatoren

stabil bzw. gemischt – also teilweise steigend, teilweise fallend und teilweise stabil), so gilt das Muster als erfüllt.

Die durch die Anwendung eines Eventoperators erzeugten Events können entweder direkt an die Eventkonsumenten weitergeleitet oder aber auch im Rahmen weiterführender Eventverarbeitung weiteren Eventoperationen unterzogen werden.

Wie bereits erwähnt, kann zwischen *zustandlosen* und *zustandsbehafteten Eventoperatoren* unterschieden werden. Während sich die zustandslosen Eventoperatoren ausschließlich auf die Verarbeitung einzelner Events konzentrieren und ihre Verarbeitungslogik nicht von den vorangegangenen bzw. nachfolgenden Events beeinflusst wird, werden bei der Auswertung von zustandsbehafteten Eventoperatoren mehrere eingehenden Events in die Verarbeitung miteinbezogen. ***Tabelle 3.1*** fasst die soeben vorgestellten Eventoperatorarten zusammen und gibt für die einzelnen Operatorarten an, ob es sich hierbei um zustandslose oder zustandsbehaftete Eventoperatoren handelt.

3.4.2 Verarbeitung von Events

Insbesondere im Zusammenhang mit der Erkennung von Eventmustern in bestimmten Zeitfenstern stellen sich einige Fragen bezüglich der Eventauswertung. So kann beispielsweise zwischen mehreren Arten von Zeitfensterdefinitionen unterschieden werden (*Abschnitt 3.4.2.1*). Weiterhin ist im Rahmen der Eventverarbeitung zum einen festzulegen, wie sich die Reihenfolge von Events bestimmt (*Abschnitt 3.4.2.2*). Zum anderen spielen die Zeitpunkte, zu denen die Ereignisauswertung durchgeführt wird, eine entscheidende Rolle (*Abschnitt 3.4.2.3*).

Schließlich ist festzulegen, welche Instanzen eines Ereignisses bei der Erkennung eines komplexen Ereignisses verbraucht werden (*Abschnitt 3.4.2.4*). Im Kontext von aktiven Systemen, wo die Ereigniserkennungsmechanismen typischerweise Anwendung finden, stellt sich weiterhin die Frage, mit welchen Aktionen (Regeln) den erkannten Ereignissen begegnet wird. Ausführungen dazu finden sich in *Abschnitt 3.4.2.5*.

3.4.2.1 Definition der Zeitfenster

Bei der Definition von Zeitintervallen kommen unterschiedliche Vorgehensweisen in Betracht (vgl. [EN10]).

Zeitbasiertes Intervall. Ein zeitbasiertes Intervall wird durch eine absolute Zeitangabe geöffnet. Das Ende eines solchen Zeitfensters kann entweder ebenfalls absolut oder durch die Angabe einer Gesamtdauer angegeben werden. Beispiele für zeitbasierte Intervalldefinitionen sind:

- *Migration DB2v8 to DB2v9 (start = 24.11.2010 19:00, end = 25.11.2010 11:00)*
- *Migration DB2v8 to DB2v9 (start = 24.11.2010 19:00, duration = 16 hours).*

Eventbasiertes Intervall. In diesem Fall wird das Zeitfenster durch Auftreten von einem bestimmten Event geöffnet. Das Schließen des Zeitfensters kann entweder mit einem anderen Event bzw. automatisch nach Ablauf einer bestimmten Zeit (Gesamtdauer) erfolgen. Zu Beispielen für diese Intervallart zählen:

- *Bufferpool-Tuning (start = „Bufferpool Hitratio außerhalb des normalen Wertebereichs“, end = „Bufferpool Hitratio (wieder) im normalen Wertebereich“)*
- *Index-Tuning (start = „schlechte Indexausnutzung“, duration = 10 minutes).*

Neben den oben genannten fixen Intervallarten können auch gleitende Intervalle definiert werden. Im Gegensatz zu fixen Zeitintervallen erfolgt hier die Eröffnung des Zeitfensters nicht absolut, sondern relativ in Abhängigkeit vom vorhergehenden Zeitfenster. Das neue Zeitfenster kann dabei ausgehend vom Anfang bzw. Ende des vorhergehenden Zeitfensters nach Ablauf einer bestimmten Zeit (*gleitendes zeitbasiertes Intervall*) oder nach Auftreten einer bestimmten Anzahl bestimmter Events (*gleitendes eventbasiertes Intervall*) eröffnet werden.

3.4.2.2 Bestimmung der Eventreihenfolge

Hinsichtlich der zeitlichen Reihenfolge von Events lassen sich verschiedene Vorgehensweisen identifizieren. Nach [ME07, EN10] eignen sich insbesondere folgende Aspekte zur Bestimmung der Eventreihenfolge:

Zeitpunkt des Eventauftretens. Die Bestimmung der Eventreihenfolge erfolgt basierend auf einem entsprechenden Zeitstempelattribut, welches den Zeitpunkt des Eventauftretens angibt.

Zeitpunkt der Eventerkennung. Im Gegensatz zum oben genannten Fall erfolgt hier die Bestimmung der Eventreihenfolge anhand eines Attributs mit dem Zeitpunkt des Eventerkennens durch eine entsprechende Komponente. Typischerweise lässt sich der Zeitpunkt der Eventerkennung relativ leicht bestimmen, da die Systemkomponenten die Events mit einem entsprechenden Zeitstempel versehen können. Der davor liegende Zeitpunkt des tatsächlichen Eventauftretens dagegen bleibt unter Umständen unbekannt und kann nur mit viel Aufwand ermittelt werden.

Zeitpunkt der Eventverarbeitung. Auch die Reihenfolge, in der die primitiven Events die für die Erkennung der Eventmuster zuständige Eventverarbeitungskomponente erreichen und somit verarbeitet werden, kann als Grundlage für die Eventreihenfolge verwendet werden. Die Ordnung der Events ergibt sich somit aus der Reihenfolge der Events im eingehenden Eventstrom der Eventverarbeitungskomponente.

Nutzerdefiniertes Attribut. Die Bestimmung der Eventreihenfolge kann auch mit Hilfe eines nutzerdefinierten Eventattributs erfolgen. Ein solches Attribut kann beispielsweise Zeitpunktangaben oder aber auch lediglich eine eindeutige Eventkennung, auf der sich eine Ordnung definieren lässt, beinhalten.

3.4.2.3 Zeitpunkte der Eventauswertung

Es ist weiterhin wichtig zu klären, zu welchen Zeitpunkten die Auswertung der Eventmuster stattfinden soll. Dabei kommen zwei Ansätze in Frage:

Sofortige Auswertung. Die Überprüfung der vorliegenden Eventmuster erfolgt sofort mit jedem neu hinzugekommenen Event. Der Vorteil dieser Vorgehensweise liegt in einer unmittelbaren Mustererkennung, während der durch die somit häufige Musterüberprüfung entstehende Aufwand das System stark belastet und als nachteilig einzustufen ist.

Verzögerte Auswertung. In diesem Fall werden die Events erst am Ende eines Zeitfensters ausgewertet. Handelt es sich dabei um ein sehr langes Zeitfenster, so wird die Eventerkennung durch diese Vorgehensweise stark verzögert. Durch die Festlegung hinreichend kurzer („mittellanger“) Zeitfenster können sowohl die Verzögerung als auch der Verarbeitungsaufwand etwas reduziert werden.

3.4.2.4 Eventauswahl und -verbrauch

Erfolgt die Auswertung der eingetroffenen Events erst am Ende des Zeitfensters, so stellt sich beim Vorliegen mehrerer Eventinstanzen von gleichem Typ die Frage, welche Eventinstanzen für das komplexe Ereignis konsumiert werden. In [CKAK94] wird in diesem Kontext zwischen folgenden Vorgehensweisen unterschieden:

Chronologische Eventauswahl (chronicle). Für das komplexe Ereignis werden die primitiven Ereignisinstanzen entsprechend ihrer Reihenfolge (vgl. *Abschnitt 3.4.2.2*) verwendet, das heißt, die jeweils älteste Ereignisinstanz wird herangezogen. Diese Vorgehensweise wird auch als *first come, first used* bezeichnet. Dieser Verbrauchsmodus entspricht der Vorgehensweise bei einer sofortigen Auswertung eingehender Events.

Umgekehrt-chronologische Eventauswahl (recent). Für das zusammengesetzte Ereignis wird die jeweils letzte, also jüngste, primitive Ereignisinstanz verbraucht. Die früheren Instanzen werden ignoriert.

Fortlaufende Eventauswahl (continuous). Jede eingehende primitive Eventinstanz wird in die Mustererkennung einbezogen. Die bereits in einen komplexen Event eingeflossenen primitiven Eventinstanzen gelten dabei zunächst nicht als verbraucht und können für die Erkennung weiterer komplexer Events verwendet werden.

Kumulative Eventauswahl. Bei diesem Verbrauchsmodus erfolgt der Verbrauch aller Instanzen eines primitiven Ereignisses E_i, sobald eine der Instanzen von E_i in die Erkennung eines komplexen Events einbezogen wurde. Keine der zu diesem Zeitpunkt vorliegenden Eventinstanzen von E_i kann somit für die Erkennung weiterer komplexer Ereignisse verwendet werden.

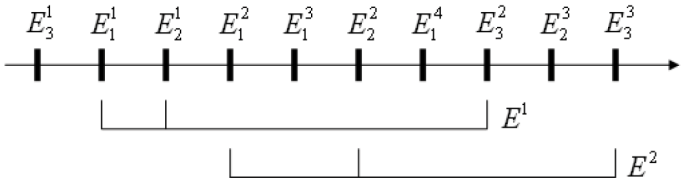
Das zusammengesetzte Ereignis $E = \mathit{sequence}(\mathit{all}(E_1, E_2), E_3)$ dient nachfolgend als Beispiel zur Veranschaulichung der oben genannten Verbrauchsmodi (***Abbildung 3.11***). Das Ereignis E tritt genau dann ein, wenn zunächst E_1 und E_2 in beliebiger Reihenfolge eingetreten sind und anschließend E_3 eingetreten ist. Weiterhin nehmen wir an, dass folgende Ereignisinstanzen am Ende eines Zeitfensters für die Mustererkennung vorliegen: $E_3^1; E_1^1; E_2^1; E_1^2; E_1^3; E_2^2; E_1^4; E_3^2; E_2^3; E_3^3$ [Jäg08]. Dabei kennzeichnet E_x^y das y-te Auftreten von E_x im betrachteten Intervall.

Im Fall, dass jeweils die ältesten Instanzen der Ereignisse E_1, E_2 und E_3 zur komplexen Ereigniserkennung herangezogen werden (chronologische Eventauswahl, ***Abbildung 3.11a***), wird das Ereignis E zweimal signalisiert: Zum einen, wenn das Ereignis E_3 zum zweiten Mal eintritt (E_3^2), und zum anderen, wenn das Ereignis E_3 zum dritten Mal eintritt (E_3^3).

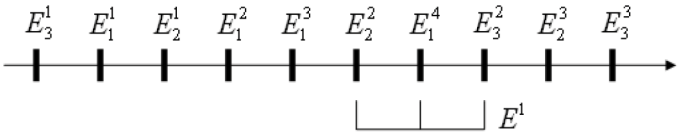
Handelt es sich um eine umgekehrt-chronologische Eventauswahl (***Abbildung 3.11b***), wird das Ereignis E nur einmal signalisiert, nämlich beim zweiten Auftreten von E_3 (E_3^2). Beim dritten Vorkommen von E_3 (E_3^3) kann das Ereignis E nicht erneut signalisiert werden, da die ersten drei Eintreten von E_1 ignoriert und das vierte Auftreten von E_1 (E_1^4) bereits verbraucht wurde.

Bei der fortlaufenden Eventauswahl (***Abbildung 3.11c***) wird das Ereignis E sechsmal ausgelöst. Diese sechs Instanzen des zusammengesetzten Ereignisses $E = \mathit{sequence}(\mathit{all}(E_1, E_2), E_3)$ entstehen dabei folgendermaßen: Die Ereignisinstanz E_3^1 wird zunächst ignoriert, da zur Erkennung des zusammengesetzten Ereignisses E zunächst das Eintreten von sowohl E_1 als auch E_2 (in beliebiger Reihenfolge) und erst dann das Eintreten von E_3 benötigt werden. Das zweite eingetretene Ereignis E_1^1 bildet den ersten Bestandteil des Eventmusters $\mathit{all}(E_1, E_2)$. Das nachfolgend eingetretene Ereignis E_2^1 erfüllt einerseits dieses Eventmuster und bildet zugleich den ersten Bestandteil einer neuen Instanz des gleichen Musters aufgrund der Tatsache, dass die Ereignisinstanzen mehrmals zur Detektion zusammengesetzter Ereignisse verwendet werden können. Die zunächst eingetretene Ereignisinstanz E_1^2 trägt beispielsweise dazu bei, dass das durch E_2^1 begonnene Eventmuster $\mathit{all}(E_1, E_2)$ einerseits abgeschlossen wird und andererseits eine weitere Instanz dieses Eventmusters begonnen wird. Die nachfolgend eingetretenen Instanzen der Events E_1 bzw. E_2 gehen analog mehrfach in das Eventmuster $\mathit{all}(E_1, E_2)$ ein. Wird nun anschließend das Ereignis E_3^2 signalisiert, können die ersten fünf bisher begonnenen Ereignismuster $\mathit{sequence}(\mathit{all}(E_1, E_2), E_3)$ vervollständigt werden, wodurch das Ereignis E fünfmal erkannt wird. Beim Eintreten von E_3^3 wird schließlich die sechste Ereignissequenz beendet und somit das zugehörige Ereignis E zum sechsten Mal signalisiert.

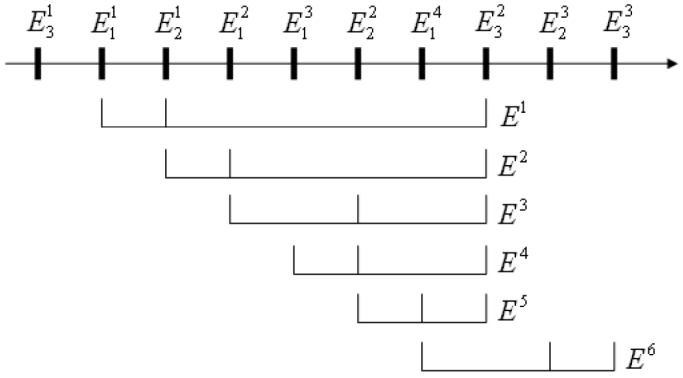
Anschließend wird bei der kumulativen Eventauswahl (***Abbildung 3.11d***) beim Eintreten der Ereignisinstanz E_3^2 das Eventmuster $\mathit{sequence}(\mathit{all}(E_1, E_2), E_3)$ erkannt und somit das



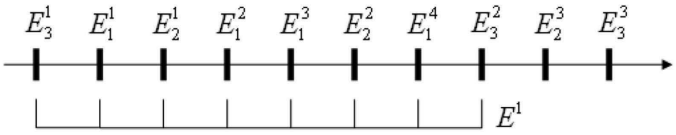
(a) Chronologische Eventauswahl



(b) Umgekehrt-chronologische Eventauswahl



(c) Fortlaufende Eventauswahl



(d) Kumulative Eventauswahl

Abbildung 3.11: Eventauswahl am Beispiel $E = \mathit{sequence}(\mathit{all}(E_1, E_2), E_3)$

Ereignis E signalisiert. Zugleich werden alle bis zu diesem Zeitpunkt vorliegenden Eventinstanzen von E_1 (E_1^1, E_1^2, E_1^3 und E_1^4), E_2 (E_2^1 und E_2^2) sowie E_3 (E_3^2) als konsumiert gekennzeichnet.

Je nach Anwendungsbereich erweisen sich einzelne Vorgehensweisen als sinnvoll. Die chronologische Eventauswahl ist beispielsweise typisch für Anwendungen, die kausale Abhängigkeiten zwischen den einzelnen Ereignissen analysieren. In solchen Anwendungen wird aufgezeigt, welche Ereignisse immer nach bestimmten anderen Ereignissen eintreten. Die umgekehrt-chronologische Eventauswahl kommt dagegen vorzugsweise bei sensorbasierten Anwendungen wie beispielsweise Krankenhausüberwachungssystemen zum Einsatz. Dort werden Ereignisse schnell hintereinander ausgelöst, wodurch atomare Ereignisse desselben Typs sehr oft auftreten. In Anwendungsbereichen, in denen die komplexen Ereignisse innerhalb eines beweglichen Zeitfensters erkannt werden (beispielsweise Trendanalysen), spielt währenddessen die fortlaufende Eventauswahl eine wichtige Rolle.

Die oben beschriebenen Ereignisverbrauchsmodi lassen sich entweder pro zu erkennenden komplexen Event oder für die Eventverarbeitungskomponente und somit für alle damit zu erkennenden Events festlegen.

In [CKAK94, PD99] werden die hier vorgestellten Verbrauchsmodi anhand von Beispielen erläutert. Neben diesen vier Verbrauchsmodi sind noch weitere denkbar, die jedoch nicht Gegenstand dieser Arbeit sind.

3.4.2.5 Regelauswahl und -ausführung

Ereigniserkennungsmechanismen finden typischerweise in *aktiven Systemen* Anwendung. Die erkannten Ereignisse werden dabei zur Auslösung von bestimmten Regeln eingesetzt. Solche aktiven Mechanismen bringen einerseits eine hohe Flexibilität (Änderung des Systemverhaltens durch Einpflegen von neuen zu erkennenden Events und entsprechenden Aktionen) mit und ermöglichen andererseits eine unmittelbare Reaktion auf Veränderungen, die anderenfalls nur umständlich mit Hilfe von Polling-Mechanismen umzusetzen wären. Jedoch hält sich typischerweise die Bereitschaft der Nutzer, sich bei der Umsetzung ihrer Geschäftslogik auf aktive Mechanismen zu verlassen, in Grenzen [CCW00].

Einer der wichtigsten Gründe für die eingeschränkte Bereitschaft der Nutzer zum Einsatz aktiver Mechanismen liegt in den eventuellen Abhängigkeiten unter den Regeln und der sich daraus ergebenden Schwierigkeit, eine Vorhersage bezüglich des Systemverhaltens sowie der Performance zu treffen [BDR04]. Die durch die erkannten Ereignisse ausgelösten Regeln können wiederum neue Ereignisse auslösen, welche in der Ausführung weiterer Regeln resultieren.

In diesem Zusammenhang sind insbesondere Themen wie die Terminierung und die Konfluenz von Bedeutung.

Terminierung. Wie bereits erwähnt, können Regeln weitere Ereignisse und somit weitere Regeln auslösen. Dabei kann es unter Umständen zu direkten bzw. indirekten Zyklen kommen. Werden solche Zyklen nicht erkannt bzw. vermieden, so können sie in einer unendlichen Ausführung betroffener Regeln resultieren. Dieses Problem ist in der Welt der Aktiven Datenbanken unter dem Begriff der *Terminierung* bekannt und wurde bereits in zahlreichen Forschungsarbeiten adressiert [AWH92, Wei97, BW00,

CT03, BDR04]. Da dieses Problem nach [BDR98] im Allgemeinen nicht entscheidbar ist, konzentrieren sich die Forschungsansätze auf die Abdeckung bestimmter Spezialbereiche.

Nach [BDR04] gibt es drei Vorgehensweisen zur Adressierung des Terminierungsproblems. Dabei handelt es sich um die etwa aus praktischer Sicht verbreitete Beschränkung der maximalen Anzahl von Zyklusdurchläufen, um die Einführung syntaktischer Einschränkungen sowie um die statische Analyse, die zum Zeitpunkt der Regeldefinition aus den Abhängigkeiten unter den Regeln mögliche Zyklen ableitet.

Konfluenz. Ein weiteres bekanntes Problem bei der Regelausführung ist unter dem Namen *Konfluenz* bekannt [AWH92, Wei97]. Dabei geht es um die Fragestellung, in welcher Reihenfolge die ausgelösten Regeln ausgeführt werden sollen, falls zum Zeitpunkt der Regelausführung mehrere Regeln angestoßen wurden, weil sie beispielsweise an das gleiche Ereignis gekoppelt sind oder weil mehrere Ereignisse zum gleichen Zeitpunkt erkannt wurden.

Als *konfluent* wird eine Regelmenge genau dann bezeichnet, wenn unabhängig von der Reihenfolge der Regelausführung der gleiche Endzustand des Systems erreicht wird. Anderenfalls wird die Regelmenge als nicht konfluent angesehen und muss aufgelöst werden, indem bestimmt wird, in welcher Reihenfolge die Ausführung stattfindet.

Eine mögliche Vorgehensweise ist die Priorisierung von Regeln [CT03]. Auch im Rahmen der in der vorliegenden Arbeit beschriebenen Forschungsarbeiten wurde dieser Weg gewählt. Detaillierte Ausführungen zu unserem Ansatz zur Regelpriorisierung finden sich in *Abschnitt 6.2*.

Eine weitere Möglichkeit zur Lösung des Konfluenz-Problems ist die Ermittlung der Kommutativität von Regeln [AWH92]. Sind alle Regelpaare einer Regelmenge kommutativ, so ist diese Regelmenge konfluent. [Wei97] gibt einen Algorithmus zur statischen Analyse der Regeln an, der feststellt, ob alle Regeln der vorgegebenen Regelmenge konfluent sind und isoliert gegebenenfalls die Problemfälle. Die somit identifizierten nicht kommutativen Regeln können anschließend beispielsweise mittels einer partiellen Ordnung priorisiert werden.

3.4.3 *IBM Tivoli Active Correlation Technology*

Eines der Ziele der vorliegenden Arbeit ist die Konzeption und prototypische Umsetzung einer Infrastruktur zum autonomen Tuning von datenbankbasierten Anwendungen. Bei der Auswahl eines in den Prototyp einzubindenden Eventkorrelationsprodukts spielten daher Aspekte wie die Universalität, eine leichte Integrierbarkeit in ein bestehendes Softwaresystem und die Leichtgewichtigkeit eine entscheidende Rolle. Bei den meisten in der Welt der Aktiven Datenbanken bekannten Systemen wie beispielsweise *HiPAC* [DBB+88], *Snoop* [CKAK94], *SAMOS* [GD93, DG00], *REACH* [BBKZ93] oder *Oracle Rules Manager* [Ora07] handelt es sich um speziell auf bestimmte Problembereiche oder Architekturen zugeschnittene Prototypen bzw. Produkte. Aufgrund mangelnder Universalität bzw. fehlender Verfügbarkeit kamen sie für den Einsatz in unserem Prototyp nicht in Betracht, Nebenbedingungen ergaben sich dabei auch durch die Einbindung in den Projektkontext.

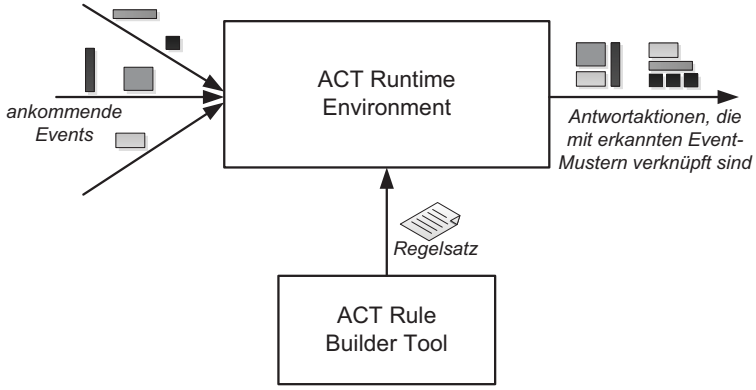


Abbildung 3.12: Architektur von *Active Correlation Technology* (in Anlehnung an [BG05])

Eines der ersten Systeme, die sich eingehend mit der Eventkorrelationstechnologie beschäftigt haben, war *Tivoli Event Correlation Engine* [Bez10]. Diese mit Prolog implementierte Eventkorrelationsengine ist in *IBM Tivoli Enterprise Console* [IBM03b] integriert. Aufgrund der hohen Mächtigkeit und Komplexität dieses Produkts setzt sein Betrieb hohe Systemressourcen voraus und eignet sich daher nicht für unsere Zwecke.

Die über die Jahre gesammelten Erfahrungen auf dem Gebiet der Ereigniskorrelation flossen unter anderem auch in die *Active Correlation Technology (ACT)*, eine von *Tivoli* entwickelten Technologie zur Verarbeitung und Korrelation eingehender Events, die mittlerweile in einigen IBM-Produkten wie beispielsweise dem *Tivoli Directory Integrator* [IBM10] Anwendung findet. Im Gegensatz zu den meisten anderen verfügbaren Korrelationsengines wird ACT als eine Library bereitgestellt und lässt sich somit mit einem geringen Programmieraufwand in ein bestehendes System integrieren. Nicht zu vernachlässigen ist außerdem die Tatsache, dass ACT zur Laufzeit kaum Performance-Overhead verursacht. Aus diesen Gründen fand die *Tivoli Active Correlation Technology* Einsatz bei der prototypischen Entwicklung der autonomen Tuning-Infrastruktur (*Abschnitt 5.2*) und wird nachfolgend zur exemplarischen Veranschaulichung der Funktionsweise eines Ereignisverarbeitungssystems verwendet.

In ACT werden die ankommenden atomaren Ereignisse in einer Queue angesammelt und in Echtzeit mit einem Satz von vordefinierten Mustern (*Korrelationsregeln*) analysiert. Erkannte Muster werden zu komplexen Ereignissen zusammengesetzt und stoßen anschließend die Ausführung der damit verknüpften Antwortaktionen an [BG05]. Bei den auszuführenden Antwortaktionen handelt es sich dabei um einen beliebigen Java-Code.

Die Gesamtarchitektur von ACT findet sich in ***Abbildung 3.12***. Im Mittelpunkt der Architektur steht eine Laufzeitumgebung (*ACT Runtime Environment*), in der die Korrelationstechnologie eingebettet ist. Ein weiterer wichtiger Bestandteil von ACT ist die XML-basierte Regelsprache, die zur Definition der Regeln verwendet wird. Bei der Regeldefinition kann der Benutzer durch den zur ACT-Architektur gehörenden Regelerzeuger (*Rule Builder*) unterstützt werden.

Zur Definition zusammengesetzter Ereignisse bietet ACT sieben verschiedene Regelarten an. Der angebotene Regelmusterumfang hat sich bewährt, um die meisten Ereigniskorrela-

tionssituationen zu erfassen [BG05]. Zur Modellierung komplexerer Ereignismuster lassen sich die Korrelationsmuster durch das Zurückführen korrelierter Events in die ACT-Queue miteinander kombinieren. Die von ACT unterstützten Korrelationsmuster sind *Filter Patterns*, *Collection Patterns*, *Duplicate Patterns*, *Computation Patterns*, *Threshold Patterns*, *Sequence Patterns* sowie *Timer Patterns*.

Filter Pattern. Bei diesem Korrelationsmuster wird bei jedem eingehenden Ereignis überprüft, ob es mit der Regeldefinition übereinstimmt. Jedes somit identifizierte Ereignis wird aus dem Eventstrom herausgefiltert und löst unmittelbar nach seiner Erkennung eine vordefinierte Regelantwort aus. Bei diesem Korrelationsmuster handelt es sich um ein zustandsloses Korrelationsmuster, so dass die Vorgänger- bzw. Nachfolgerereignisse bei der Ereignisidentifikation nicht berücksichtigt werden können. Mit der Filterungsregel lassen sich beispielsweise einfache Ereignisse wie *„Die Bufferpool Hitratio ist kleiner als 80%“* erkennen.

Timer Pattern. Mit Hilfe dieses Musters lässt sich ein einmaliger bzw. ein fortlaufender Timer definieren. Nach Ablauf einer vorgegebenen Zeit wird eine entsprechende Antwortaktion initiiert. Bei einem fortlaufenden Timer erfolgt unmittelbar nach Auslösen der Antwortaktion ein Timer-Neustart. Hilfreich ist dieses Korrelationsmuster insbesondere bei der Umsetzung periodisch wiederkehrender Aktionen.

Collection Pattern. Wie der Name dieses Korrelationsmusters es vermuten lässt, werden hier die Ereignisse, die der Regeldefinition entsprechen, über einen bestimmten Zeitraum gesammelt und zu einem komplexen Ereignis zusammengefasst. Am Ende dieses Zeitraums wird eine entsprechende Regelantwort ausgelöst. Aufbauend auf dieser Musterart gibt es zwei weitere Musterarten: das *Duplicate Pattern* und das *Computation Pattern*.

Duplicate Pattern. Die Funktionsweise dieses Korrelationsmusters ähnelt der Funktionsweise des *Collection Patterns* mit dem Unterschied, dass das erste Vorkommen eines erkannten Events bereits die entsprechende Regelantwort auslöst und alle nachfolgenden Events vom gleichen Typ bis zum Ende des vorgegebenen Zeitraums zwar mitgezählt werden, aber keine weitere Antwortaktion mehr auslösen. Dieses Verhalten ist beispielsweise dann sinnvoll, wenn nach Auslösen einer Antwortaktion zunächst geprüft werden soll, ob diese bereits den gewünschten Effekt erzielt hat. Am Ende des Zeitintervalls lässt sich eine weitere abschließende Antwortaktion initiieren. Will man beispielsweise auf das Ereignis *„Die Bufferpool Hitratio ist kleiner als 80%“* mit einer Vergrößerung der Bufferpool-Größe reagieren, so ist zu berücksichtigen, dass je nach Systemkonfiguration das Ereignis mehrfach in sehr kurzen Zeiträumen erkannt werden kann. Eine Reaktion auf jedes der erkannten Ereignisse würde zu einer Überallokation des Speichers für den Bufferpool führen, was sich wiederum hinderlich auf die System-Performance auswirkt. Solche Überreaktionen lassen sich mit Hilfe eines Duplicate-Musters vermeiden, indem beispielsweise nur beim ersten Eventauftreten darauf reagiert wird und alle nachfolgenden Instanzen des erkannten Events ignoriert werden, bis eine bestimmte Zeit verstrichen ist.

Computation Pattern. Auch dieses Korrelationsmuster hat ein ähnliches Verhalten wie das *Collection Pattern*: Ereignisse, die der Regeldefinition entsprechen, werden gesammelt und die zugehörige Antwortaktion wird einmal am Ende des Zeitintervalls

eingeleitet. Zusätzlich wird jedoch beim Erkennen jedes Teilereignisses eine Berechnungsfunktion angewendet. Damit lassen sich beispielsweise bestimmte Attribute der erkannten Teilereignisse auswerten und in die Berechnung einbeziehen.

Threshold Pattern. Dieses Muster ermöglicht das Sammeln von bestimmten Ereignissen in einem vorgegebenen Zeitraum. Dabei kann entweder die Anzahl der gesammelten Ereignisse bestimmt werden bzw. eine Auswertung ihrer Attribute mittels einer Berechnungsfunktion erfolgen. Das Ergebnis kann anschließend mit einem vordefinierten Schwellenwert abgeglichen werden. Wird dieser Schwellenwert im vorgegebenen Zeitraum erreicht, so wird unmittelbar danach die zugehörige Antwortaktion ausgeführt. Eine Ausnahmeaktion kann eingeleitet werden, falls der Schwellenwert innerhalb des vorgegebenen Zeitraums nicht erreicht wurde. Unter Zuhilfenahme dieses Musters lässt sich beispielsweise das Konfigurieren der Sperrlistengröße (*locklist*) umsetzen. Treten beispielsweise innerhalb einer bestimmten Zeit mehrere Sperreskalationen auf, so lässt sich die Größe der Sperrliste automatisch anpassen (vgl. *Abschnitt 3.1.1*).

Sequence Pattern. Mit dem *Sequence Pattern* lässt sich eine Menge bzw. eine Sequenz von bestimmten Teilereignissen in einem vordefinierten Zeitfenster erkennen und anschließend eine entsprechende Antwortaktion einleiten. Mittels einer Ausnahmeaktion kann dabei auf das Nichteintreten des komplexen Ereignisses reagiert werden. Weiterhin kann eine Aktion eingeleitet werden, falls das komplexe Ereignis nur zum Teil eingetreten ist, d.h. nur ein Teil der Teilereignisse im vorgegebenen Zeitraum erkannt wurde.

Die Mächtigkeit der von ACT unterstützten Korrelationsmuster wurde in [Jäg08] eingehend untersucht. Dabei wurden die in *Abschnitt 3.4.1* vorgestellten Eventoperatoren mit Hilfe der ACT-Korrelationsmuster modelliert. Einige Korrelationsoperatoren lassen sich unmittelbar mit entsprechenden Regelmustern abbilden. So können beispielsweise die Sequenzen bzw. die Konjunktion (Eventoperator *All*) mittels des Sequenzmusters und die Disjunktion (Eventoperator *Any*) mittels des Filtermusters umgesetzt werden. Für einige Eventoperatoren findet sich jedoch keine direkte Entsprechung in Form eines ACT-Korrelationsmusters. In solchen Fällen muss auf Kombinationen der ACT-Korrelationsmuster zurückgegriffen werden.

3.5 Zusammenfassung

Performance-Probleme können durch ein Zusammenspiel einer Vielzahl von Faktoren entstehen und resultieren in einem Systemverhalten, welches von dem üblichen bzw. dem durch einen Administrator vorgegebenen Verhalten abweicht. Die Behebung eines solchen Problems setzt eine genaue Identifikation der Problemursachen voraus.

Grundsätzlich können für das zu tunende 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 können und nur die (un-)mittelbaren Auswirkungen spüren (z. B. Anstieg der Antwortzeit), ist diese Art der Problemidentifikation nur bedingt zur automatischen Administration einsetzbar [RWRA08]. Die in *Abschnitt 3.3.3*

kurz skizzierte Vorgehensweise zur automatischen Erkennung von nutzerinitiierten Events mittels Data-Mining-Techniken kann dabei Abhilfe schaffen.

Im Rahmen der vorliegenden Arbeit wurde eine Integration von typischerweise in komplexen IT-Infrastrukturen bereits vorhandenen, sich über mehrere Schichten der IT-Infrastruktur erstreckenden Überwachungswerkzeugen in den Problemerkennungsprozess angestrebt, um dadurch unter anderem den durch den Problemerkennungsprozess verursachten Mehraufwand weitgehend zu reduzieren. Zur Beschreibung der Problemsituationen wird hier daher auf einen zweistufigen Ansatz zurückgegriffen.

Für die einzelnen integrierten Überwachungswerkzeuge betrachten wir im ersten Schritt die den Systemzustand repräsentierenden Metriken und definieren „ungesunde“ Werte bzw. Wertebereiche (durch Schwellenwerte), die auf ein Problem hindeuten und daher im Rahmen weiteres Problemdiagnose- bzw. Tuning-Vorgehens adressiert werden sollten. Zum einen können jedoch solche Werte bzw. Wertebereichsüberschreitungen in einer komplexen Systemumgebung häufig auftreten. Zum anderen beziehen sich solche atomaren Ereignisse typischerweise auf einzelne Systembereiche (wie Netzwerk, Application Server bzw. Datenbank-Management-System), in denen sie durch entsprechende Überwachungswerkzeuge isoliert voneinander erkannt werden.

Im zweiten Schritt erfolgt daher eine Konsolidierung dieser Ereignisse aus sämtlichen aktiven Überwachungswerkzeugen und eine mittels Eventkorrelationsmechanismen darauf durchgeführte Erkennung von vorab definierten Ereignismustern. Dadurch kann nicht nur die Eventflut reduziert, sondern insbesondere die einzelnen Problemsituationen miteinander in Beziehung gebracht und die Ursachen der aufgetretenen Probleme durch die Berücksichtigung des entsprechenden Problemkontexts genau eingegrenzt werden.

Erst wenn eine mögliche Problemursache identifiziert ist, kann eine entsprechende Gegenmaßnahme (*Tuning-Plan*) eingeleitet werden. Gelingt jedoch eine genaue Identifikation der Problemursache mittels der Eventkorrelation nicht, da beispielsweise bestimmte Daten nicht vorliegen und zuerst gesammelt werden müssen, so sind weitere Untersuchungen notwendig. Die im nachfolgenden *Kapitel 4* beschriebenen *Tuning-Pläne* stellen Aktivitäten zur Unterstützung der Problemdiagnose bzw. zum Datenbank-Tuning bereit und ermöglichen eine visuelle, intuitive Programmierung von entsprechenden Abläufen. Diese Abläufe lassen sich in Reaktion auf bestimmte (korrelierte) problemrepräsentierende Ereignisse ausführen.

Kapitel 4

Beschreibung und Modellierung von bewährten Datenbank-Tuning-Praktiken

Über die Jahre sind die IT-Systeme bezüglich ihrer Mächtigkeit, aber auch ihrer Komplexität stark gewachsen, so dass ihre Administration und Wartung zu einem sehr wichtigen Bestandteil der *Total Cost of Ownership (TCO)* geworden sind. Das zu einer effizienten Administration und Wartung notwendige Wissen steht den Administratoren heutzutage beispielsweise in der Fachliteratur, aber auch in elektronischer Form in Produkthandbüchern, Online-Magazinen, Foren oder IT-Blogs zur Verfügung. Diese Informationen sind jedoch zum größten Teil unstrukturiert, also in Form von Freitext verfügbar, und sind somit auch nicht maschinell auswertbar.

In dem vorliegenden Kapitel wird ein Formalisierungsvorschlag für Datenbank-Tuning-Praktiken erarbeitet, der einerseits eine strukturierte Modellierung der Datenbank-Tuning-Vorgehensweisen und andererseits eine automatische Ausführung dieser ermöglicht und die Administratoren somit bei ihrer alltäglichen Arbeit unterstützt und entlastet.

Zwar bieten viele Überwachungstools wie beispielsweise der DB2-Diagnosemonitor (*Abschnitt 3.3.2.1*) Funktionalitäten zum probleminitiierten Anstoßen von beliebigen Skripten, was einen ersten Schritt zur Automatisierung von Aufgaben der Administratoren macht und die Administratoren bei ihren Tätigkeiten unterstützt. Die Erstellung von solchen Skripten ist jedoch ein langwieriger und fehleranfälliger Prozess, dem sich jeder Administrator immer wieder unterziehen muss. Je höher die gewünschte Mächtigkeit des zu erstellenden Skripts, umso höher auch die Komplexität und der Aufwand bei seiner Erstellung und Wartung. Zur Umsetzung einiger Funktionalitäten müssen oft sogar ganze Anwendungen programmiert werden, die dann in die Skripte eingebunden werden können, so dass Anpassungen solcher Skripte nur noch sehr schwer möglich sind.

Eine eigens für das Datenbank-Tuning entwickelte, leicht erlernbare, intuitive Sprache, die sowohl die Beschreibung von Tuning-Vorgehensweisen als auch ihre Ausführung ermöglicht, hat, wie jede domänenspezifische Sprache, eine Reihe von Vorteilen (vgl. [vKV00]). Sie ermöglicht die Verwendung der Terminologie und der Abstraktionsebene der Domäne des Datenbank-Tuning, wodurch einerseits der Prozess der Tuning-Plan-Erstellung und andererseits die Verständlichkeit der erstellten Tuning-Pläne erleichtert werden. Durch die Abstraktion von der eigentlichen technischen Realisierung und Einschränkung der Mächtigkeit der Sprache auf die für die betrachtete Domäne tatsächlich notwendigen Funktionalitäten können nicht nur die Benutzerfreundlichkeit, sondern insbesondere auch die Produktivität erhöht und die Fehleranfälligkeit reduziert werden. Auch die Validierung

sowie Optimierung von erstellten Tuning-Vorgehensweisen gestaltet sich unter der Verwendung einer entsprechenden domänenspezifischen Sprache als wesentlich einfacher und effektiver.

Nach einem kurzen Überblick über ausgewählte Ansätze zur Beschreibung von Arbeitsabläufen (*Abschnitt 4.1*), wird nachfolgend zunächst die Vorgehensweise der Datenbankadministratoren beim Tuning von Datenbanken näher untersucht und anhand von einigen Beispielen erläutert (*Abschnitt 4.2*). Der darauffolgende *Abschnitt 4.3* beschäftigt sich mit der Identifikation von möglichen Effektoren eines DB2-Systems, die bei der Konzeption einer Sprache zur Beschreibung von Tuning-Abläufen eine entscheidende Rolle spielen.

In *Abschnitt 4.4* wird anschließend ein Ansatz zur Konzeption einer auf das Datenbank-Tuning zugeschnittenen Ablaufbeschreibungssprache vorgestellt. Die in dieser Sprache erstellten Tuning-Abläufe werden als *Tuning-Pläne* bezeichnet. Zur Modellierung von Tuning-Plänen erscheint die Wahl eines visuellen, Workflow-orientierten Ansatzes vielversprechend. In *Abschnitt 4.5* wird daher zunächst am Beispiel von *IBM WebSphere Integration Developer* veranschaulicht, wie sich ein herkömmliches Workflow-Management-System zur Modellierung und Ausführung von Tuning-Plänen einsetzen lässt. Anschließend folgt eine kurze Vorstellung der entsprechenden Untersuchungen mit einem leichtgewichtigen und leicht erweiterbaren, zur Erstellung von dynamischen Internetanwendungen konzipierten Modellierungs- und Ausführungswerkzeug namens *IBM WebSphere sMash*. Eine Zusammenfassung des in diesem Kapitel diskutierten Ansatzes und der Ergebnisse findet sich anschließend in *Abschnitt 4.6*.

4.1 Bekannte Ansätze zur Beschreibung von Arbeitsabläufen

Nachfolgend werden ausgewählte Ansätze zur Beschreibung von Arbeitsabläufen, die entweder aus der Forschung oder aber auch aus ihrem industriellen Einsatz bekannt sind, vorgestellt.

4.1.1 Workflow-Management-Systeme

Durch die *Workflow-Management-Systeme (WfMS)* wird eine vielversprechende Technologie bereitgestellt, die die Realisierung prozessorientierter, verteilter Anwendungen ermöglicht [LR97]. Die Separierung der Prozessablauflogik von dem eigentlichen Anwendungscode und dessen Fachsemantik ist dabei nicht nur für eine schnellere Anwendungserstellung vorteilhaft, sondern reduziert dabei auch die Fehleranfälligkeit bedeutend. Auch die durch die eventuellen Veränderungen der Anforderungen notwendigen Anpassungen der Prozesslogik lassen sich durch diese Trennung mit einem geringen Aufwand vornehmen, was zu einer Senkung der Entwicklungs- und Wartungskosten beiträgt.

Im Kontext des Workflow-Managements werden die Begriffe Geschäftsprozess und Workflow oft als Synonyme verwendet. Zwar beschreiben beide Begriffe Prozesse in Unternehmen, jedoch unterscheiden sie sich im Wesentlichen durch ihren Detaillierungsgrad [GG99].

Ein *Geschäftsprozess* beschreibt die zeitliche Abfolge betriebswirtschaftlicher Aufgaben. Aus einer betriebswirtschaftlich-strategischen Gesamtsicht auf den Prozess beschreibt also ein Geschäftsprozess, „*was*“ gemacht werden muss.

Im Gegensatz zu einem Geschäftsprozess hat ein *Workflow* eine technische Sicht auf den Prozess und beschreibt, „*wie*“ der Geschäftsprozess auszuführen ist. Die Workflow-Beschreibung ist dabei technisch und detailliert genug, um eine Ausführung auf einem *Workflow-Management-System (WfMS)* zu ermöglichen. Bei einem WfMS handelt es sich um ein Werkzeug, welches die Modellierung, Simulation, Ausführung und Überwachung von Workflows unterstützt [GG99].

Zur Beschreibung von Workflows existieren zahlreiche Sprachen und Standards[12]. Die Workflow-Modellierungssprachen dienen oft verschiedenen Zwecken und unterscheiden sich daher bezüglich ihrer Eigenschaften. So kann beispielsweise in der Regel zwischen ausführbaren und nicht ausführbaren Sprachen unterschieden werden. Die nicht ausführbaren Sprachen dienen vor allem der Konzeption von Workflow-Prozessen und ihrem Austausch zwischen unterschiedlichen Systemen. Die ausführbaren Sprachen hingegen ermöglichen die Simulation sowie die Ausführung von Workflow-Prozessen und legen typischerweise weniger Wert auf die Lesbarkeit und die Austauschbarkeit der Prozesse. Eine bekannte Ausnahme bildet *Business Process Model and Notation (BPMN)*. In seiner aktuellen Version (*BMPN 2.0*) verfolgt es das Ziel, sowohl die Workflow-Modellierungs- als auch -Ausführungs-Aspekte in sich zu vereinen [OMG11].

Zu den wichtigen Vertretern von ausführbaren Workflow-Modellierungssprachen zählen beispielsweise *Business Process Execution Language (BPEL)* und *XML Process Definition Language (XPDL)*. Während *XPDL* [WFM08] eine graphenorientierte Sprache ist und somit den Austausch von graphischen Workflow-Modellen ermöglicht, verfolgt *BPEL* [OAS07b, OAS07a] einen blockorientierten Ansatz, weshalb es keine Speicherung von graphischen Informationen unterstützt und für den Austausch von Workflow-Diagrammen kaum geeignet ist. Vielmehr handelt es sich bei BPEL um eine Sprache zur Orchestration von *Web Services*. Eine starke Web-Service-Orientierung von BPEL ist sogar aus *WS-BPEL* – der offiziellen Bezeichnung der BPEL-Spezifikation – zu entnehmen.

Eine weitere bekannte Workflow-Sprache ist *YAWL* [vT03]. Hierbei handelt es sich um eine ausführbare Sprache, die auf den Konzepten der Petri-Netze aufbaut und entsprechend die formal nachweisbaren Eigenschaften von Petri-Netzen besitzt. Diese Eigenschaften können speziell zur Simulation bzw. Korrektheitsbeweisen von mit *YAWL* definierten Abläufen verwendet werden. Bei der Konzeption von *YAWL* wurde weiterhin versucht, alle zu dem Zeitpunkt bekannten Workflow-Patterns[13] zu integrieren, weshalb *YAWL* als eine sehr ausdrucksstarke Workflow-Modellierungssprache gilt.

[12] Zu den für die Weiterentwicklung und Standardisierung der Sprachen verantwortlichen Organisationen zählen *Workflow Management Coalition (WfMC)*, *Object Management Group (OMG)*, *Business Process Management Initiative (BPMI)* und *OASIS*.

[13] Als *Workflow-Patterns* werden in Workflow-Projekten oft wiederkehrende Ablaufmuster bezeichnet [vTKB03].

4.1.2 Ansätze aus der medizinischen Informatik

Bereits vor über 20 Jahren hat das Gesundheitswesen die Notwendigkeit erkannt, die Qualität von Dienstleistungen zu verbessern und die Behandlungskosten zu reduzieren. Durch die Erstellung von Standardarbeitsanweisungen (*Standard Operating Procedures, SOPs*) zur Diagnose und Behandlung von Patienten hat man versucht, medizinisches Personal bei der Arbeit in ihrem hoch komplexen Arbeitsumfeld zu unterstützen. Die Bereitstellung von solchen SOPs erfolgt dabei typischerweise als strukturierter Freitext. Seltener finden sich aber auch Flussdiagramme, die den logischen Ablauf der Diagnose und der Therapie ausgewählter Krankheiten beschreiben, deren Bedeutung jedoch wiederum im Freitext erfasst ist [Sed08].

Aufbauend auf solchen in unstrukturierten Formaten vorliegenden SOPs wurden zahlreiche Ansätze zur formalen Erfassung und zum Teil auch einer semi-automatischen Abarbeitung dieser entwickelt[14]. Der Fokus der Ansätze ist hierbei jedoch nicht die vollständige Automatisierung der Behandlung, sondern deren Qualitätsunterstützung bzw. -sicherung [RWRA08]. Nachfolgend werden zwei Ansätze exemplarisch vorgestellt. Neben einer kurzen Einführung in die *Arden Syntax*, einen der ältesten und bekanntesten Ansätze, konzentrieren wir uns dabei auf das *Guideline Interchange Format*, welches über einen langen Zeitraum weiterentwickelt wurde und die Grundlage für viele weitere Forschungsansätze bildet.

Arden Syntax. Eines der ältesten und bekanntesten Ansätze ist die *Arden Syntax* [HCP+93]. Das Ziel dieser Sprache ist die Dokumentation des klinischen Wissens in einer regelbasierten Form. Das Ablegen des Wissens erfolgt in der Arden Syntax in einzelnen Modulen (*Medical Logic Module, MLM*). Per Definition muss dabei das Wissen eines MLMs zum Treffen einer medizinischen Entscheidung ausreichend sein. Solche MLMs umfassen neben einigen unstrukturiert abgespeicherten Metadaten ein Event, welches die Ausführung des Algorithmus initiiert, und die eigentlichen, den Algorithmus beschreibenden Regeln. Zur automatischen Abarbeitung von MLMs lassen sich diese in eine Programmiersprache umwandeln und anschließend ausführen. Zu den Nachteilen von *Arden Syntax* zählt beispielsweise die Tatsache, dass keine Verkettung einzelner Module unterstützt wird. Auch eine Kombination von Events (vgl. *Abschnitt 3.4*) ist in der *Arden Syntax* nicht möglich. Die einzelnen Events, die die Ausführung eines MLM initiieren, sind ODER-verknüpft.

GLIF. Das *Guideline Interchange Format (GLIF)* ist ursprünglich mit dem Ziel entstanden, den Austausch formalisierter menschenlesbarer Leitlinien zu ermöglichen und dabei unter anderem die Nachteile der *Arden Syntax* zu beheben. So eignet sich GLIF beispielsweise im Gegensatz zur Arden Syntax zum Formalisieren von Leitlinien, die aus mehreren Schritten bestehen. In [OMGM+98] wurde das GLIF-Modell unter dem Namen *GLIF2* veröffentlicht. Es ermöglichte die Modellierung von Leitlinien als Abläufe von klinischen Entscheidungen. Die einzelnen Aktivitäten dieser Abläufe wurden dabei als *Guideline Steps* bezeichnet. Dabei wurde zwischen folgenden Arten von *Guideline Steps* unterschieden: *Action Step*, *Conditional Step*, *Branch Step* und *Synchronization Step*.

[14] http://www.openclinical.org

Während *Action Steps* die auszuführenden klinischen Aktionen beschreiben und neben der Aktion jeweils eine Verknüpfung mit dem nachfolgenden Schritt beinhalten, lässt sich mit Hilfe von *Conditional Steps* der Kontrollfluss in Abhängigkeit von Zwischenergebnissen verändern. Je nach Auswertungsergebnis (*true* oder *false*) der im *Conditional Step* angegebenen Bedingung kann mit dieser Schrittart der nachfolgende auszuführende *Guideline Step* bestimmt werden. Zur Umsetzung der parallelen Ausführung können die *Branch Steps* (Verzweigung) sowie *Synchronization Steps* (Zusammenführung) verwendet werden.

Aufgrund der Ausrichtung von GLIF2 auf die Erstellung *menschenlesbarer* Leitlinien war der Großteil der Elemente nicht formalisiert und in Form von Freitext erfasst, so dass eine automatische Abarbeitung solcher Modelle nicht möglich war.

Bei der Konzeption des Nachfolgemodells *GLIF3* wurde daher eine zusätzliche Anforderung der *Maschinenlesbarkeit* gestellt [PBO+00, BPT+04]. Die in GLIF bzw. GLIF2 als Freitext erfassten Elemente wie Entscheidungskriterien, klinische Aktionen und Patientendaten lassen sich in GLIF3 nun formal definieren. Des Weiteren wurde GLIF3 so konzipiert, dass die Erstellung von SOPs auf drei Abstraktionsstufen ermöglicht wird. Auf der konzeptuellen Ebene erfolgt die Modellierung von SOPs als Flussdiagramme. Diese werden im zweiten Schritt um ausführungsrelevante Informationen wie Datendefinitionen, Entscheidungskriterien oder Kontrollflüsse erweitert. Anschließend erfolgt eine Abbildung der Aktionen und der Datenobjekte auf ein konkretes Zielsystem, welches für die Ausführung der SOPs verantwortlich ist.

Für die automatische Abarbeitung von Instanzen des GLIF3-Modells wurde die *GLIF Execution Engine (GLEE)* [WPT+04] entwickelt. Dabei wurde der Ansatz eines Interpreters gewählt. Die GLIF3-Modellinstanzen werden also zur Laufzeit schrittweise interpretiert und abgearbeitet.

Sedlmayr et al. verfolgen dagegen in in [SRR+07, Sed08] einen Kompilierungsansatz. Mit Hilfe einer Abbildungsassistenten transformieren sie die GLIF3-Modellinstanzen in Workflows, die anschließend mit einem Workflow-Management-System ausgeführt werden können. In ihrem Ansatz erfolgt eine Transformation der SOPs in das jPDL-Format (*jBPM Process Definitional Language, jPDL*). Die im jPDL-Format vorliegenden Workflows können anschließend mit dem *Java Business Process Management Framework* der *JBoss Gruppe*[15] ausgeführt werden. Durch das Zurückgreifen auf bereits existierende, ausgereifte Workflow-Management-Systeme zur Ausführung von GLIF3-Modellinstanzen kann dabei von den Vorteilen, wie beispielsweise einer breiten Tool-Unterstützung bei der Überwachung der Ausführung bzw. bei der Simulation, profitiert werden.

Es ist dabei zu erwähnen, dass trotz einer automatischen Abarbeitung von SOPs der Großteil von Aktionen nicht automatisch ausgeführt werden kann. Ein medizinisches Informationssystem unterbreitet also dem medizinischen Personal lediglich Handlungsvorschläge, die von diesem Personal angenommen oder abgelehnt werden können. Aufgrund dieser Tatsache unterscheiden sich die hier vorgestellten Ansätze zur Automatisierung von SOPs von unserem, in der vorliegenden Arbeit vorstellten Ansatz einer automatischen Abarbeitung von Datenbank-Tuning-Abläufen.

[15] http://www.jbpm.org

4.1.3 *IBM Tivoli Monitoring for Databases*

Zu einem im Kontext der vorliegenden Arbeit besonders wichtig erscheinenden Ansatz zur Beschreibung von Arbeitsabläufen zählt der Ansatz des Softwareprodukts *Tivoli Monitoring*. Ein erster Einblick in die Funktionsweise von *Tivoli Monitoring* wurde bereits in *Abschnitt 3.3.2.2* gegeben. Hauptaugenmerk lag dabei auf der Überwachungsfunktionalität und der Definition von Ereignissen. Auf die im Rahmen der Überwachung mittels *Tivoli Monitoring*-Produkten erkannten Ereignisse kann jedoch auch automatisch mit Aktionen reagiert werden. Die Tivoli-Infrastruktur setzt dazu zwei Stufen der Automatisierung um.

Während die *Reflex-Automatisierung* beim Auftreten eines Events lediglich das Anstoßen eines Skripts bzw. die Ausführung einer einzelnen Anweisung ermöglicht, bietet das Konzept der *erweiterten Automatisierung* deutlich mehr Mächtigkeit. Es lassen sich damit komplexe Administrationsabläufe (bezeichnet als *Policies*) umsetzen. Eine *Policy* besteht aus einer Menge von Aktivitäten, die durch entsprechende Kontrollstrukturen zu einem Ablauf verknüpft sind[16].

Tivoli Monitoring bringt einen Satz an Standardaktivitäten mit. Dazu zählen neben wenigen anderen folgende kontrollflussspezifische Aktivitäten [IBM08b]:

- `Wait until a situation is true`
- `Evaluate a situation now`
- `Make a choice`
- `Start/Stop a policy`
- `Start/Stop a situation`
- `Suspend execution`
- `Take action or Write message`

Die Überwachungsagenten stellen des Weiteren Sätze an vordefinierten, produktspezifischen Aktivitäten bereit. So werden vom DB2-Überwachungsagenten des *Tivoli Monitoring for Databases* folgende Aktivitäten angeboten [IBM07]:

- `Backup Database`
- `Rebind Package`
- `Reorg Table`
- `Run Statistics`
- `Start DB2`
- `Stop DB2`
- `Update Database Configuration`
- `Update Database Manager Configuration`

[16] Es ist zu beachten, dass sich die Verwendung des Begriffs *Policy* im Kontext des *IBM Tivoli Monitoring* grundsätzlich von unserer Verwendung des Begriffs beispielsweise in *Abschnitt 6.2* unterscheidet.

Die durch den DB2-Agenten von *Tivoli Monitoring for Databases* bereitgestellten Aktivitäten ermöglichen die Automatisierung einiger Basisadministrationsaufgaben. Zur Erstellung von komplexeren Administrationsabläufen reicht jedoch die Mächtigkeit bereitgestellter Aktivitäten in der Regel nicht aus, so dass die Definition eigener Aktivitäten unabdingbar ist und durch die Tivoli-Infrastruktur auch ermöglicht wird. Der Hauptnachteil dieses Ansatzes ist dennoch die hohe Komplexität der gesamten Infrastruktur und die damit verbundene hohe Systembelastung zur Laufzeit. Die Tatsache, dass der Großteil der zur Datenbank-Administration und zum -Tuning notwendigen Aktivitäten von Nutzern (DBA) selbst erstellt werden muss, ist ein weiterer großer Nachteil. Auch eine Möglichkeit zum Austausch von erstellten Abläufen unter den Nutzern ist nicht gegeben. Die soeben angesprochenen Probleme werden mit dem nachfolgend vorgestellten Ansatz adressiert und weitgehend behoben. Insbesondere wird in unserem Ansatz außerdem die Möglichkeit geboten, die Ausführung der Tuning-Abläufe einerseits durch Metadaten bzw. gesammelte Statistiken und andererseits durch die Vorgaben des Administrators zu beeinflussen (vgl. *Kapitel 6*).

4.2 Aufgaben und typische Diagnose- bzw. Tuning-Vorgehensweisen der Datenbankadministratoren

Das Spektrum der typischen Aufgaben eines Datenbankadministrators reicht von nahezu Echtzeitanforderungen (Ad-hoc-Maßnahmen, „erste Hilfe“) bis hin zu langfristigen Aufgaben und erstreckt sich unter anderem über folgende Bereiche [WR07b]:

- Initiale Systemkonfiguration und -verteilung
- Logischer und physischer Datenbankentwurf
- Laufende Systemverwaltung
- Gewährleistung der *Service Level Agreements* (Sicherung der hohen Verfügbarkeit, *Disaster Recovery* etc.)

Den größeren Anteil ihrer Arbeitszeit verbringen die Administratoren dabei mit den Aufgaben der laufenden Systemverwaltung. Diese lassen sich weiter in *reaktive* und *präventive* Tätigkeiten unterteilen. Während die zeitkritischen, reaktiven Administrationstätigkeiten Aufgabenbereiche wie Erkennen von Problemen, Bestimmen ihrer Ursachen und Einleiten von lindernden/behebenden Maßnahmen umfassen, handelt es sich bei den präventiven Tätigkeiten um Maßnahmen, die dem Auftreten von Problemen möglichst vorbeugen sollen. Dazu zählen Aufgaben wie die Verwaltung von Datenbankobjekten, Aktualisieren von Statistiken, Speicherplatz- sowie Benutzerverwaltung und Defragmentierung von Daten.

Die in diesem Kapitel vorgestellten Konzepte zur Erstellung und probleminitiierten Ausführung von Tuning-Abläufen decken zwar alle oben angesprochenen Bereiche ab, konzentrieren sich jedoch insbesondere auf die Aufgaben der laufenden Systemverwaltung, da ihre Automatisierung die höchste Entlastung der DBA verspricht.

Das Tuning von IT- bzw. Datenbank-Systemen ist ein iterativer Prozess, der sich in mehrere Schritte unterteilen lässt. Da die einzelnen Schritte von einander abhängig sind und somit Auswirkungen aufeinander haben, empfiehlt sich nach [Ora01] in der Regel das Einhalten folgender Tuning-Reihenfolge:

1. Geschäftslogik
2. Initiale Datenbankkonfiguration
3. Anwendungsentwurf
4. Logischer Datenbankentwurf
5. SQL-Tuning
6. Zugriffspläne
7. Speicherbereiche
8. I/O und physische Strukturen
9. Ressourcenkonflikte
10. Systemumgebung (Betriebssystem, Netzwerk etc.)

Nach erfolgten Änderungen in einzelnen Schritten sind dabei unter Umständen Wiederholungen einiger vorhergehender Schritte notwendig. So müssen beispielsweise nach Veränderungen des logischen Datenbankentwurfs entsprechende Anpassungen in darauf operierenden Anwendungsprogrammen vorgenommen werden. Nach einer Auflösung von Ressourcenkonflikten sind wiederum gegebenenfalls die physischen Strukturen anzupassen.

Wird ein Problem anhand von Symptomen festgestellt, so sind zunächst seine Ursachen zu identifizieren (Problemdiagnose), damit sie im nächsten Schritt behoben werden können (Problemauflösung). Nachfolgend werden einige Techniken zur Diagnose von Performance-Problemen exemplarisch vorgestellt [Hen07]:

Kennzahlen-Analyse. Mittels der Kennzahlen-Analyse erfolgt die Untersuchung von Kennzahlen, die die Performance eines DBMS beschreiben [Sch03]. Diese können von den Ressourcenattributen abgeleitet werden. Die Ressource Bufferpool hat beispielsweise Kennzahlen wie Größe, *Extent*-Größe, *Bufferpool Hitratio* etc. Diese Kennzahlen können zu bestimmten Zeitpunkten ausgelesen bzw. über einen Zeitablauf aufgezeichnet werden. Anhand von Daumenregeln und durch ihre langjährige Erfahrung können DBA unter Einbeziehung weiterer Informationen über die Systemumgebung und die anliegende Workload die Performance der betrachteten Ressource einschätzen. Eine niedrige *Bufferpool Hitratio* von ca. 70% kann beispielsweise bei einer OLAP-Workload akzeptabel sein, während sie bei einer OLTP-Workload entsprechende Tuning-Maßnahmen erfordert.

Bottleneck-Analyse. In Anbetracht der Tatsache, dass Prozesse entweder aktiv sind (und ihre Aufgabe ausführen) oder auf die Gewährung von Ressourcen wie CPU oder ähnliches warten, konzentriert sich die Bottleneck-Analyse auf die Identifikation der Ressourcen, die die höchsten Wartezeiten verursachen [Sch03]. Werden solche Ressourcen gefunden, so ist festzustellen, ob ihr Durchsatz durch Tuning-Maßnahmen verbessert und die Wartezeit reduziert werden können.

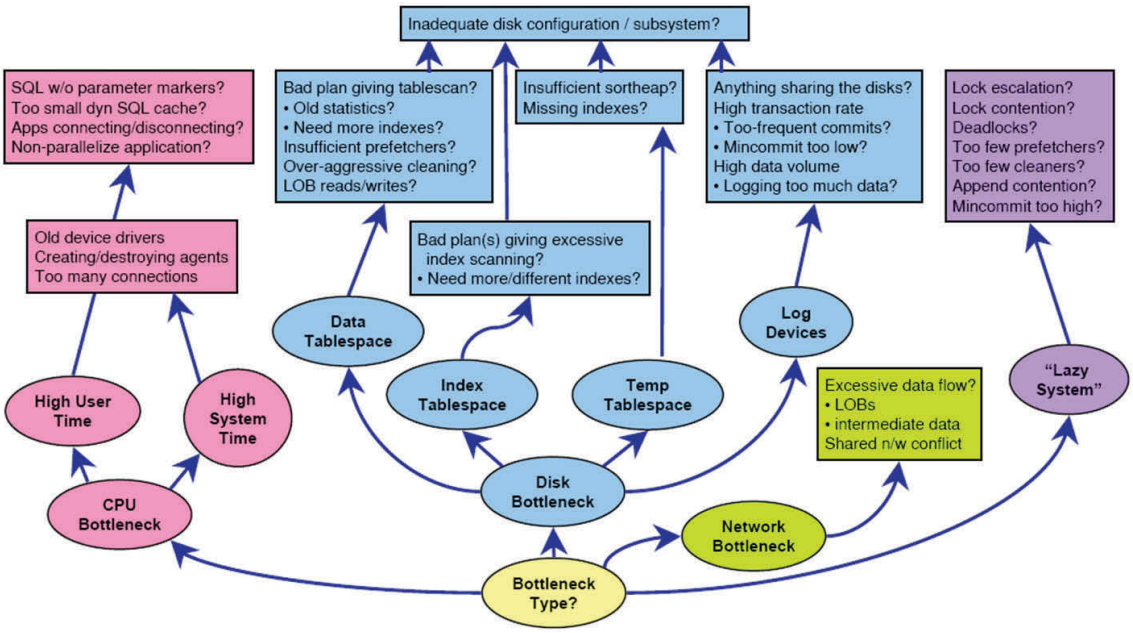
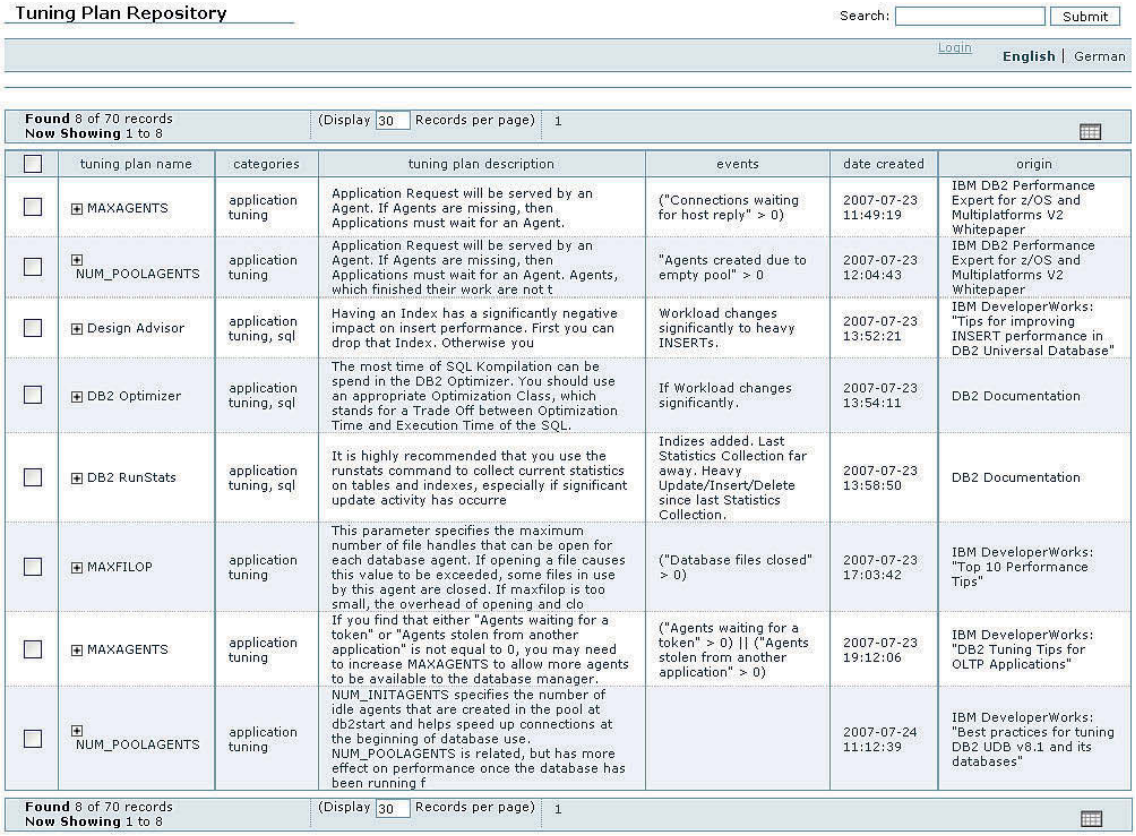


Abbildung 4.1: Eine bewährte Vorgehensweise bei der Problemdiagnose bzw. dem Tuning [Ree07]

Workload-Analyse. Mittels der Workload-Analyse lassen sich Anwendungen oder sogar einzelne Anweisungen identifizieren, die aufgrund eines hohen Ressourcenverbrauchs das System zu stark beanspruchen. Bei einer Top-Down-Vorgehensweise lassen sich beispielsweise zuerst die „teuersten“ Anwendungen bzw. Anweisungen bestimmen und anschließend die von ihnen beanspruchten Ressourcen ermitteln [Sne03]. Die Ermittlung der „teuren“ Anwendungen bzw. Anweisungen kann dabei beispielsweise nach Kriterien wie Häufigkeit bzw. Dauer der Ausführung, I/O-Zeit oder CPU-Zeit erfolgen. Wird der Durchsatz dieser stark ausgelasteten Ressourcen durch Tuning-Maßnahmen erhöht, so hat dies hohe Performance-Auswirkungen auf die darauf operierenden Anwendungen. Das Verfahren lässt sich durch das Gruppieren der Anweisungen hinsichtlich ihrer Ähnlichkeit optimieren [Hay05].

Zur Steigerung der Qualität der Problemdiagnose lassen sich die soeben vorgestellten Diagnosetechniken miteinander kombinieren [Sch03]. Die Kennzahlen-Analyse verschafft dem Administrator einen schnellen Überblick über wichtige Performance-Metriken und bietet Anhaltspunkte für eine genauere Eingrenzung des Problems. Die Bottleneck-Analyse zeigt hingegen auf, wo die eigentlichen Engpässe im System bestehen. Mit der Workload-Analyse bekommt man anschließend einen Überblick über die Probleme verursachenden Anwendungen.

Ein Beispiel für eine solche Kombination der oben vorgestellten Diagnosevorgehensweisen findet sich in [Ree07] und wurde in [Hen07] zusammengefasst. Insbesondere ist bei der Vorgehensweise von Steeve Rees sehr gut erkennbar, welche Informationen zu welchen Zeitpunkten auf welchen Ebenen benötigt werden. Eine ähnliche Vorgehensweise wird auch in [Ora08] beschrieben.



Tuning Plan Repository

Search: Submit

Login English | German

Found 8 of 70 records
Now Showing 1 to 8

(Display 30 Records per page) 1

	tuning plan name	categories	tuning plan description	events	date created	origin
☐	⊞ MAXAGENTS	application tuning	Application Request will be served by an Agent. If Agents are missing, then Applications must wait for an Agent.	("Connections waiting for host reply" > 0)	2007-07-23 11:49:19	IBM DB2 Performance Expert for z/OS and Multiplatforms V2 Whitepaper
☐	⊞ NUM_POOLAGENTS	application tuning	Application Request will be served by an Agent. If Agents are missing, then Applications must wait for an Agent. Agents, which finished their work are not t	"Agents created due to empty pool" > 0	2007-07-23 12:04:43	IBM DB2 Performance Expert for z/OS and Multiplatforms V2 Whitepaper
☐	⊞ Design Advisor	application tuning, sql	Having an Index has a significantly negative impact on insert performance. First you can drop that Index. Otherwise you	Workload changes significantly to heavy INSERTs.	2007-07-23 13:52:21	IBM DeveloperWorks: "Tips for improving INSERT performance in DB2 Universal Database"
☐	⊞ DB2 Optimizer	application tuning, sql	The most time of SQL Kompilation can be spend in the DB2 Optimizer. You should use an appropriate Optimization Class, which stands for a Trade Off between Optimization Time and Execution Time of the SQL.	If Workload changes significantly.	2007-07-23 13:54:11	DB2 Documentation
☐	⊞ DB2 RunStats	application tuning, sql	It is highly recommended that you use the runstats command to collect current statistics on tables and indexes, especially if significant update activity has occurre	Indizes added. Last Statistics Collection far away. Heavy Update/Insert/Delete since last Statistics Collection.	2007-07-23 13:58:50	DB2 Documentation
☐	⊞ MAXFILOP	application tuning	This parameter specifies the maximum number of file handles that can be open for each database agent. If opening a file causes this value to be exceeded, some files in use by this agent are closed. If maxfilop is too small, the overhead of opening and clo	("Database files closed" > 0)	2007-07-23 17:03:42	IBM DeveloperWorks: "Top 10 Performance Tips"
☐	⊞ MAXAGENTS	application tuning	If you find that either "Agents waiting for a token" or "Agents stolen from another application" is not equal to 0, you may need to increase MAXAGENTS to allow more agents to be available to the database manager.	("Agents waiting for a token" > 0) || ("Agents stolen from another application" > 0)	2007-07-23 19:12:06	IBM DeveloperWorks: "DB2 Tuning Tips for OLTP Applications"
☐	⊞ NUM_POOLAGENTS	application tuning	NUM_INITAGENTS specifies the number of idle agents that are created in the pool at db2start and helps speed up connections at the beginning of database use. NUM_POOLAGENTS is related, but has more effect on performance once the database has been running f		2007-07-24 11:12:39	IBM DeveloperWorks: "Best practices for tuning DB2 UDB v8.1 and its databases"

Found 8 of 70 records
Now Showing 1 to 8

(Display 30 Records per page) 1

Abbildung 4.2: Tuning-Plan-Repository, kategorienbasierte Übersicht von Tuning-Plänen

Zur Ermöglichung eines Vergleichs zwischen einem „gesunden“ und einem problematischen Systemzustand ist das kontinuierliche Sammeln von Informationen über das DBMS und seine Umgebung notwendig [Wie11]. Diese Informationen umfassen unter anderem Art und Version des verwendeten Betriebssystems sowie des DBMS, Konfigurationseinstellungen der Datenbank und des Datenbankmanagers, Registry-Einstellungen, Informationen zum Datenbank-Design bzw. -Schema sowie Laufzeitinformationen wie Zugriffspläne, Performancemetriken etc.

Nach Auftreten eines Problems ist nach [Ree07] zuerst zu überprüfen, ob irgendwelche Veränderungen dieses Problem verursacht haben können, was erfahrungsgemäß oft der Fall ist. Zu solchen Veränderungen zählen beispielsweise neue Datenbankanwendungen, neue Applikationen auf dem gleichen System, drastischer Anstieg des Datenvolumens bzw. der Nutzeranzahl, Veränderungen an der DBMS-Konfiguration bzw. an der verwendeten Hardware etc. Ist eine von solchen Veränderungen tatsächlich die Ursache für das aufgetretene Problem, so ist zu überprüfen, ob diese Veränderung rückgängig gemacht werden kann/darf oder ob sie beabsichtigt war und das Problem entsprechend anderweitig beseitigt werden muss.

Im nächsten Schritt wird versucht, anhand von mit Hilfe von entsprechenden Analysewerkzeugen ermittelten Symptomen auf der Betriebssystemebene festzustellen, welche der Betriebssystemressourcen (CPU, Hauptspeicher, Festplatte, Netzwerk) einen Engpass auf-

weisen (***Abbildung 4.1***). Für eine so ermittelte Ressource wird anschließend versucht, die Ursache des Engpasses zu bestimmen. Für ein Festplattenproblem (Disk Bottleneck) wird beispielsweise geprüft, ob es sich bei einem der Tablespaces um einen Hotspot handelt. Für diesen (Daten-)Tablespace lassen sich die entsprechenden Tabellen ermitteln. Anhand dieser Tabellen können die darauf operierenden Anweisungen bestimmt werden, die beispielsweise die meisten Tabellenzeilen lesen oder die meiste CPU-Zeit verbrauchen. Durch eine Analyse dieser Anweisungen bzw. durch die Überprüfung relevanter Konfigurationseinstellungen lassen sich nun die möglichen Ursachen (z. B. veraltete Statistiken, fehlende Indizes, ungenügende Anzahl von Prefetcher-Prozessen etc.) für die vorliegenden Symptome feststellen und beseitigen.

Bei der soeben vorgestellten Diagnose- sowie Tuning-Vorgehensweise handelt es sich um eine bewährte Tuning-Maßnahme (*Best Practice*). Im vorliegenden Kapitel werden Konzepte zur intuitiven, visuellen Programmierung von solchen Tuning-Best-Practices vorgestellt. Diesen Ausführungen liegen die Ergebnisse eines Kooperationsprojekts des *Lehrstuhls für Datenbanken und Informationssysteme* der *Friedrich-Schiller-Universität Jena* mit der *IBM Deutschland Research & Development GmbH* zugrunde, im Rahmen dessen unter anderem zahlreiche bewährte Vorgehensweisen zum Tunen von DB2-basierten Softwarestacks und dazugehörige Problemdefinitionen, die mit Hilfe von korrelierten Events modelliert wurden, identifiziert und in einem entsprechenden webbasierten System zusammengetragen wurden. In ***Abbildung 4.2*** findet sich ein Screenshot des *Tuning Plan Repository*, in welchem eine Übersicht von Tuning-Plänen, die der Kategorie „Application Tuning“ zugeordnet sind, zu sehen ist.

Zu einem der wichtigen im Rahmen der vorliegenden Arbeit (und des IBM-Projekts) betrachteten Anwendungsszenarien gehört das Diagnostizieren und Beheben des Sperreskalationen-Problems. Die in ***Abbildung 4.3*** skizzierte Vorgehensweise wurde größtenteils aus *IBM Data Studio Administration Console (DSAC)* entnommen, ausführlich analysiert und um einige Spezialfälle erweitert [Roe08, Are09]. Die Abbildung enthält vier Arten von Knoten: Die Wurzel bildet das aufgetretene Problem, welches es zu beheben gilt. Die Vierecke mit leicht gerundeten Ecken repräsentieren die Bedingungen, die erfüllt sein müssen, damit der darunter liegende Teilbaum ausgewertet wird. Die Vierecke mit stark gerundeten Ecken repräsentieren die Annahmen, die aufgrund der bis dahin erfüllten Bedingungen getroffen werden können. Bei den Blättern des Baums handelt es sich um Empfehlungen, die zu befolgen sind, um das vorliegende Problem zu lösen bzw. um entsprechende Aktionen. Die Kanten sind als alternative Pfade zu verstehen, die nach der Überprüfung der in den Knoten angegebenen Bedingungen eingeschlagen werden können.

Im Zuge einer Sperreskalation werden die Sperreinträge von Tabellen mit den meisten Sperren so lange eskaliert, bis der Sperrlistenverbrauch unter 50% der durch den Parameter `maxlocks` festgelegten Größe fällt. Die Gründe für die Sperreskalationen in DB2 wurden bereits in *Abschnitt 3.1.1* kurz erörtert. So kann beispielsweise eine Anwendung mit ihren Sperren einen Teil der Sperrliste belegen, der größer ist als der mittels des Parameters `maxlocks` festgelegte Wert (*Knoten 1*). Es können aber auch mehrere Anwendungen aktiv sein, so dass die Gesamtheit ihrer Sperren die Sperrliste füllt, obwohl keine der einzelnen Anwendungen die durch `maxlocks` definierte Schranke überschreitet (*Knoten 2*). In diesem Fall liegt die Vermutung nahe, dass die Sperrliste einfach zu klein ist (*Knoten 5*). Um diese

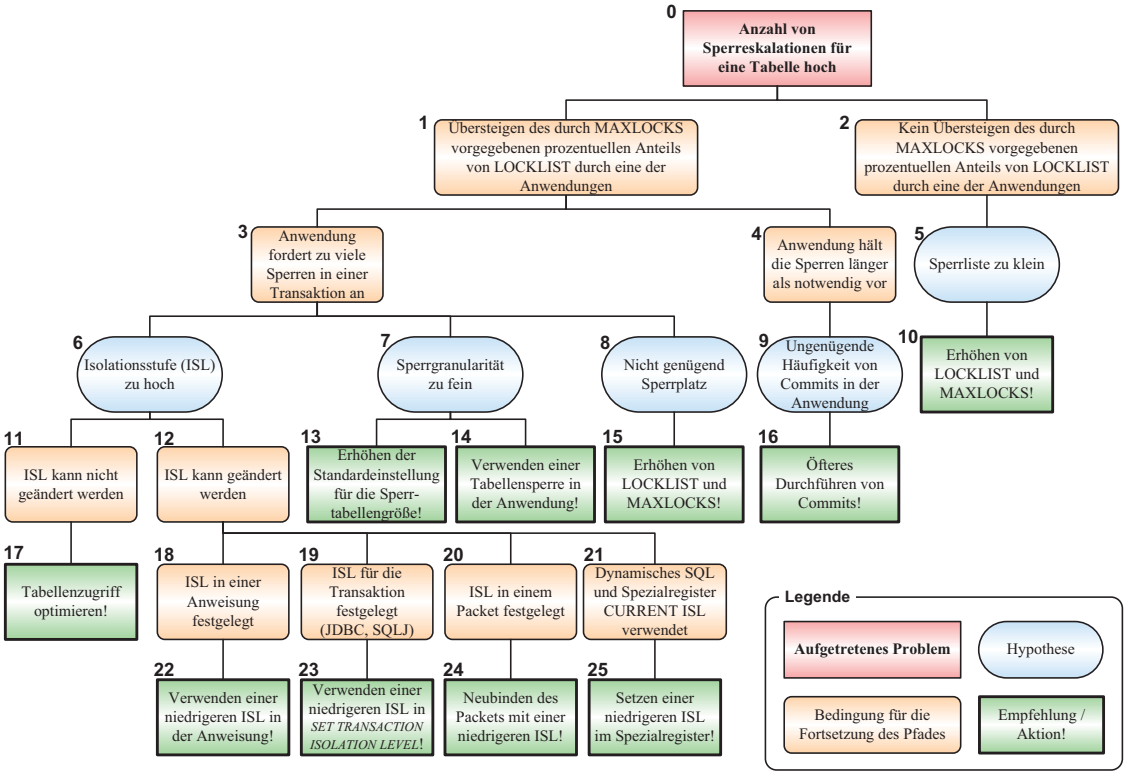


Abbildung 4.3: Diagnose und Auflösung des Sperreskalationen-Problems

aufgestellte Hypothese zu bestätigen und das Problem zu lösen, müssen die Sperrliste vergrößert sowie der Parameter `maxlocks` entsprechend erhöht werden (*Knoten 10*).

Bei einer `maxlocks`-Überschreitung muss dagegen zunächst zwischen zwei Situationen unterschieden werden: Liegen die Sperreskalationen an einer durch eine Anwendung innerhalb einer Transaktion angeforderten hohen Anzahl von Sperren (*Knoten 3*) oder an der Tatsache, dass die Sperren über einen langen Zeitraum angefordert und nicht freigegeben wurden (*Knoten 4*)? Werden die Sperren zu lange vorgehalten, so liegt es daran, dass der Anwendungsprogrammierer zu wenige Commit-Befehle verwendet hat. Dies trägt dazu bei, dass Sperren unter Umständen über lange Zeiträume nicht freigegeben werden und beeinträchtigt den Mehrbenutzerbetrieb. In diesem Fall muss die Anwendung nach Möglichkeit entsprechend abgeändert werden (*Knoten 16*). Im Rahmen eines autonomen Datenbank-Tuning kann dieser Schritt (Eingriff in die Anwendungssemantik) nicht durchgeführt werden, so dass das Problem zwar identifiziert, aber nicht automatisch aufgelöst werden kann. Für die Änderung der Anwendung ist der Anwendungsentwickler bzw. der Datenbankadministrator zuständig.

Die Ursache dafür, dass eine Anwendung zu viele Sperren in einer Transaktion anfordert, kann eine zu hohe Isolationsstufe (*Knoten 6*), eine zu feine Sperrgranularität (*Knoten 7*) bzw. nicht genügend Sperrplatz aufgrund einer zu kleinen Sperrliste bzw. einem falsch eingestellten Parameter `maxlocks` (*Knoten 8*) sein.

Ähnlich wie im Fall der mangelnden Anzahl von Commits würde eine automatische Veränderung der Isolationsstufe eine Veränderung der Semantik des Anwendungsprogramms verursachen und ist daher nicht automatisch vorzunehmen. Verzichtet man gänzlich auf die Veränderung der Isolationsstufe, so kann nur noch der Tabellenzugriff optimiert werden (*Knoten 11, 17*). Ist jedoch eine Veränderung der Isolationsstufe beispielsweise durch Unterbreiten eines entsprechenden Vorschlags dem Anwendungsentwickler bzw. dem Datenbankadministrator möglich (*Knoten 12*), so muss nur noch überprüft werden, wie diese festgelegt wurde und eine entsprechende Empfehlung bzw. Änderung vorgenommen werden (*Knoten 18-21* und *Knoten 22-25*). Ist die verwendete Sperrgranularität zu fein, so lässt sich die Standardeinstellung für die Sperrtabellengröße verändern bzw. auf die Verwendung einer Tabellensperre anstatt von feingranularen Sperren zurückgreifen (*Knoten 7, 13, 14*).

Eine Erhöhung der Sperrlistengröße (*Knoten 15*) bringt in der Regel einen Rückgang der Sperreskalationenanzahl mit sich. Es ist jedoch zu berücksichtigen, dass dadurch oft die eigentliche Problemursache nicht behoben wird und eine zu große Sperrliste sich negativ auf die System-Performance auswirken kann [Roe08]. Der Weg des Tuning der Sperreskalationenanzahl über die Erhöhung der Sperrlistengröße ist daher nach Möglichkeit erst dann einzuschlagen, wenn andere Versuche fehlgeschlagen sind.

Wie bereits angedeutet, enthält das in *Abbildung 4.3* dargestellte Diagramm eine Vielzahl von möglichen alternativen Pfaden von der Wurzel bis zu den Blättern. Oft bringt das Befolgen von Empfehlungen, die in den Blättern des Baums enthalten sind, nicht die gewünschte Wirkung. Die vorgenommenen Änderungen sind dann unter Umständen wieder rückgängig zu machen und ein anderer Pfad mit entsprechenden Empfehlungen ist zu wählen (*Backtracking*). Es handelt sich hier also um ein iteratives Vorgehen. Die Pfade werden nach und nach abgearbeitet, bis das vorliegende Problem (hier: Sperreskalationen) hoffentlich behoben wird. Nähere Erläuterungen zu der Umsetzung der soeben beschriebenen Pfadabarbeitungslogik finden sich in *Abschnitt 6.3*. Nachfolgend befassen wir uns in *Abschnitt 4.4* mit den Aspekten einer formalisierten Beschreibung von solchen Tuning-Vorgehensweisen. Vorher werden jedoch in *Abschnitt 4.3* Möglichkeit zum Tunen von DB2-Datenbanken aufgezeigt, da wir uns im umgesetzten Prototyp zunächst auf das DB2-spezifische Tuning konzentrieren.

4.3 Konfigurationsmöglichkeiten und Effektoren eines DB2-Systems

Nachdem im *Abschnitt 4.2* eine typische Vorgehensweise bei der Diagnose und Auflösung von Problemen exemplarisch vorgestellt wurde, bietet dieser Abschnitt einen Überblick über die möglichen Effektoren, die im Rahmen des System-Tuning zur Problemauflösung verwendet werden können. Zu solchen Effektoren zählen dabei nicht nur zahlreiche Konfigurationsparameter des Betriebssystems, des Anwendungsservers sowie des Datenbank-Management-Systems, sondern auch Anpassungen des logischen und physischen Datenbankentwurfs oder Optimierung von Anwendungsprogrammen und ihren Datenbankanfragen.

Die Schwierigkeit besteht also darin, eine Folge von Aktionen zu ermitteln, die das aufgetretene Problem eingrenzen und beheben, ohne dabei andere Probleme und Oszillationen in der System-Performance zu verursachen.

Zur Erreichung dieses Ziels wurde in [Gan06] ein Versuch der Modellierung eines DBMS-Ausschnitts am Beispiel von *IBM DB2* unternommen. Ziel dieses Vorgehens war die Erfassung der Zusammenhänge zwischen den Effektoren und den Sensoren des Systems und ihre Abbildung in einem mathematischen Modell, um damit eine optimale Systemkonfiguration entsprechend den vorgegebenen Zielsensorwerten identifizieren zu können. Unter Generierung einer Workload und einer ständigen Überwachung von Systemsensoren wurden zunächst die Systemparameter kontinuierlich verändert. Nach Auswertung von aufgezeichneten Effektor- sowie Sensorwerten konnten anschließend Abhängigkeiten unter den Effektoren und Sensoren festgestellt werden.

Anhand von diesen Daten wurde in [Exn07] mit Hilfe der nicht-linearen Regression eine Approximierung von Performance-Funktionen durchgeführt. Der für die Aufstellung des Systemmodells entwickelte Prototyp sowie eine Auswahl von identifizierten Performance-Funktionen wurden in [RW07] vorgestellt. Zwar sind die Ergebnisse dieses Ansatzes vielversprechend und die damit ermittelten Abhängigkeiten stimmen mit den aus der Literatur bekannten Abhängigkeiten überein, jedoch ist der Aufwand für die Erstellung eines solchen Systemmodells als sehr hoch einzustufen und steigt erheblich mit wachsender Anzahl an betrachteten Effektoren. Außerdem wirken die oft vertraulichen, systeminternen Mechanismen und Automatismen auf der Herstellerseite diesem Ansatz entgegen. Initiiert ein solcher Mechanismus beispielsweise bei Erreichen eines Schwellenwertes unter bestimmten Rahmenbedingungen eine entsprechende Aktion, so erschwert dies die Modellerstellung bzw. die Ermittlung von Performance-Funktionen. Da die heutigen IT-Systeme immer mehr auf die Verwendung von solchen systeminternen Mechanismen zurückgreifen, wurde auf eine Weiterverfolgung dieses Ansatzes verzichtet.

Stattdessen wurde ein Schwerpunkt der vorliegenden Arbeit auf die Entwicklung von Konzepten gelegt, die den Datenbankadministratoren eine intuitive Beschreibung ihrer Vorgehensweisen zur Behebung von bestimmten Problemen ermöglichen. Bevor aber die Konzepte zur Beschreibung von typischen Tuning-Abläufen vorgestellt werden, verschaffen wir uns einen Überblick über die Konfigurationsmöglichkeiten von *IBM DB2*.

4.3.1 Konfigurationsmöglichkeiten

Wie bereits in *Abschnitt 2.1.2* erläutert, ermöglicht die Architektur eines DB2-Systems den Betrieb mehrerer Datenbanken. Diese ordnen sich in die Hierarchie der DB2-Datenbankobjekte ein. Auf der obersten Ebene dieser Hierarchie findet sich das *DB2-System*, welches einer DB2-Installation entspricht und sich grundsätzlich auf der Betriebssystem-Ebene konfigurieren lässt. Einem DB2-System können mehrere *DB2-Instanzen* zugeordnet werden. Diese sind jeweils für die Verwaltung mehrerer ihnen zugeordneter Datenbanken zuständig und sind daher auch als *Datenbankmanager (DBM)* bekannt. Sie stellen logische Datenbankserverumgebungen dar und können mit Hilfe der Datenbankmanager-Konfigurationsparameter einzeln konfiguriert werden.

Die Einrichtung mehrerer Instanzen bietet einerseits Vorteile wie beispielsweise Trennung der Entwicklungsumgebung von der Geschäftsumgebung, Begrenzen des Zugriffs auf sensi-

ble Daten, Optimieren der Datenbankmanager-Konfiguration für jede Instanz oder Begrenzen der durch den Ausfall einer Instanz entstehenden Auswirkungen durch Verteilung der Daten auf mehrere Instanzen. Andererseits sind die für jede Instanz zusätzlich benötigten Systemressourcen sowie der Verwaltungsaufwand nicht zu vernachlässigen. Einer Datenbankinstanz können wiederum mehrere *Datenbanken (DB)* zugeordnet werden. Für diese können weitere Einstellungen mittels der Datenbank-Konfigurationsparameter vorgenommen werden. Die Datenbanken lassen sich partitionieren bzw. in einzelne Tabellenbereiche aufteilen (vgl. *Abschnitt 3.1.1*).

4.3.1.1 Registry- und Umgebungsvariablen

Die Steuerung der plattformabhängigen DB2-Datenbankumgebung erfolgt mittels der *DB2-Registry-* und der *Umgebungsvariablen*. Während Änderungen von Umgebungsvariablen auf der Betriebssystemebene mit einem Neustart des Systems verbunden sind, bieten die DB2-Registry-Variablen den Vorteil, dass die Änderungen sofort übernommen und für später gestartete Instanzen auch sofort wirksam werden. Die zum Zeitpunkt der Änderung aktiven Instanzen müssen jedoch zur Übernahme der neuen Einstellungen neu gestartet werden. Durch die Verwendung der DB2-Registry-Variablen wird eine zentrale Steuerung der Umgebungsvariablen ermöglicht. DB2-Registry-Variablen können dabei auf verschiedenen Ebenen definiert werden. Während die meisten DB2-Registry-Variablen auf der Instanzebene definiert werden und somit nur für eine entsprechende Instanz gültig sind, gelten die auf der globalen Ebene definierten DB2-Registry-Variablen für alle Instanzen, sofern sie nicht durch entsprechende Definitionen auf der Instanzebene überschrieben werden. In Umgebungen mit partitionierten Datenbanken lassen sich auch Einstellungen auf der Instanzknotenebene vornehmen. Diese gelten ausschließlich für die entsprechende Datenbankpartition und überschreiben somit die Einstellungen auf der Instanz- bzw. der globalen Ebene.

4.3.1.2 Konfigurationsparameter

Zwar lassen sich einige Einstellungen eines DB2-Systems mit Hilfe von Umgebungsvariablen bzw. DB2-Registry-Variablen vornehmen, die meisten Konfigurationsänderungen werden jedoch mit entsprechenden Konfigurationsparametern erzielt. Die Konfigurationseinstellungen können dabei auf zwei Ebenen vorgenommen werden. Die Datenbankmanager-Konfigurationseinstellungen gelten für die entsprechende Instanz und haben hauptsächlich Einfluss auf die Zuweisung der Ressourcen dieser Instanz oder konfigurieren die Einrichtung des Datenbankmanagers und der verschiedenen Kommunikationssubsysteme. Die Datenbank-Konfigurationseinstellungen erstrecken sich dagegen über die jeweilige Datenbank und beeinflussen die Zuweisung der Ressourcen, die dieser Datenbank zugeordnet sind.

Die DBM- bzw. DB-Konfigurationseinstellungen werden in entsprechenden Konfigurationsdateien gespeichert. Diese werden beim Anlegen einer Instanz bzw. einer Datenbank automatisch erstellt und mit Standardwerten initialisiert. Unter Umständen könnten die

Standardwerte jedoch den Anforderungen an das System nicht genügen. So können beispielsweise beim Betrieb umfangreicher Datenbanken, bei einer großen Anzahl von gleichzeitigen Verbindungen bzw. bei hohen Leistungsanforderungen für bestimmte Anwendungen Veränderungen der entsprechenden Konfigurationsparameter notwendig sein.

Auf die Konfigurationsdateien kann nicht direkt zugegriffen werden. Zur Anzeige bzw. Änderung der Konfigurationseinstellungen sind entweder die speziell dafür bereitgestellten *Anwendungsprogrammierschnittstellen (API)* oder die auf diesen API agierenden Tools zu verwenden. Zur Umsetzung einer automatisierten Konfiguration dieser Parameter sind dabei insbesondere die bereitgestellten Befehlszeilenprozessorbefehle (*Abschnitt 4.3.2.2*) interessant. Für die Konfigurationsparameter des Datenbankmanagers handelt es sich dabei beispielsweise um folgende Anweisungen:

- Auslesen der aktuellen Datenbankmanager-Konfigurationseinstellungen mit `get database manager configuration` (oder `get dbm cfg`)
- Ändern von Datenbankmanager-Konfigurationseinstellungen mit `update database manager configuration` (oder `update dbm cfg`) `using <parametername> <wert>`
- Zurücksetzen aller Parameter des Datenbankmanagers auf ihre Standardwerte mit `reset database manager configuration` (oder `reset dbm cfg`)

Vor der Durchführung entsprechender Operationen auf der Datenbankmanager-Konfiguration ist eine Verbindung zur entsprechenden Instanz mittels `attach to <instanzname>` notwendig. Anschließend ist die Instanzverbindung mittels der Anweisung `detach` wieder aufzulösen.

Während einige Parameter im laufenden Betrieb geändert werden können (online konfigurierbare Konfigurationsparameter), setzen andere einen Neustart des Datenbankmanagers bzw. das Trennen aller Verbindungen zur Datenbank voraus, damit die durchgeführten Änderungen wirksam werden. Diese Tatsache ist beim automatisierten Tunen zu berücksichtigen, da solche Neustarts das Systemverhalten beeinflussen und unter Umständen von Nutzern des Systems als sehr störend wahrgenommen werden können.

Eine detaillierte Übersicht der DBM- sowie DB-Konfigurationsparameter findet sich in [IBM06f]. Für einzelne Parameter wird dabei angegeben, welchen Einfluss sie auf die System-Performance haben. Im Rahmen des Performance-Tuning spielen naheliegenderweise die Konfigurationsparameter mit *hoher* bzw. *mittlerer* Auswirkung auf die Performance eine übergeordnete Rolle.

In [Hen07] findet sich eine exemplarische Vorstellung der Konfigurationsparameter, geordnet nach entsprechenden Hardwareressourcen CPU, Hauptspeicher, Festplatte und Netzwerk. Die Verwaltung dieser Parameter-Ressourcen-Zuordnungen ist in einem System zum autonomen Datenbank-Tuning von großer Bedeutung, da sich somit beispielsweise im Rahmen der Bottleneck-Analyse zu den identifizierten Problemressourcen die entsprechenden Konfigurationsparameter schnell bestimmen lassen.

4.3.1.3 Performance-relevante Datenbankobjekte

Nicht nur die klassischen, soeben vorgestellten Konfigurationsmöglichkeiten eines DB2-Systems, sondern insbesondere auch die zahlreichen DB2-Datenbankobjekte haben einen Einfluss auf die System-Performance. Durch Anlegen neuer bzw. Löschen oder Verändern existierender, ungeeigneter Datenbankobjekte lässt sich die Performance eines DB2-Systems steigern. Zu solchen Performance-relevanten Datenbankobjekten zählen beispielsweise:

- Partitionen
- Tabellenbereiche (*Tablespaces*)
- *Materialized Query Tables (MQTs)*
- *Multidimensional Clustering Tables (MDCs)*
- Indizes
- Bufferpools
- *Stored Procedures*
- *User-defined Functions*

Für die Erläuterungen zu den einzelnen Datenbankobjekten sei hier auf *Abschnitt 3.1.1* verwiesen.

4.3.2 Effektoren

Im vorangegangenen *Abschnitt 4.3.1* wurde gezeigt, welche Parameter bzw. Datenbankobjekte Einfluss auf die Performance eines DB2-Systems haben. Zur Ermöglichung eines probleminitiierten Tuning ist jedoch eine genaue Untersuchung der entsprechenden zur Verfügung gestellten Effektoren zur Veränderung der Parameterwerte bzw. zum Anlegen, Löschen oder Modifizieren entsprechender Datenbankobjekte notwendig. Der Vorstellung dieser Effektoren widmet sich der vorliegende Abschnitt.

Grundsätzlich stellt DB2 zwei Arten von Befehlen bereit, die im Zusammenhang mit dem autonomen Datenbanktuning interessant sind: die *Systembefehle* und die *Befehlszeilenprozessorbefehle* [IBM06e]. Während die DB2-Befehle (System-, Befehlszeilenprozessorbefehle) hauptsächlich für das Konfigurieren eines DB2-Systems verantwortlich sind, kommen für die Veränderungen des Datenbank-Designs *SQL-Anweisungen* zum Einsatz. Nachfolgend findet sich eine kurze Vorstellung dieser drei Effektorarten von DB2.

4.3.2.1 Systembefehle

Die *DB2-Systembefehle* können über die Eingabeaufforderung des Betriebssystems abgesetzt werden. Mit Hilfe eines DB2-Systembefehls kann beispielsweise der Datenbankmanager gestartet (`db2start`) oder angehalten (`db2stop`) werden. Auch der *DB2 Design Advisor* (`db2advis`) – ein Werkzeug, welches anhand der übergebenen Workload Vorschläge bezüglich der zu erstellenden oder zu löschenden Indizes, MQTs, MDCs bzw. Partitionen unterbreitet – bzw. die *DB2 Explain Facility* (`db2expln`) – ein Werkzeug, welches ausführliche Informationen über den Zugriffsplan bereitstellt, den der Optimizer zur Ausführung einer SQL-Anweisung ausgewählt hat – lassen sich mit den entsprechenden Systembefehlen aufrufen. Der schreibende bzw. lesende Zugriff auf die DB2-Registry-Variablen erfolgt mittels des entsprechenden Systembefehls `db2set`.

4.3.2.2 Befehlszeilenprozessorbefehle

Der *DB2-Befehlszeilenprozessor (command line processor, CLP)* dient der Ausführung von Datenbank-Utilities, dem Absetzen von SQL-Anweisungen bzw. dem Zugriff auf die DB2-Hilfefunktionen. Zur Ausführung von Befehlen des Befehlszeilenprozessors muss zunächst der Befehlszeilenprozessor gestartet werden. Es ist entweder durch das Voranstellen von `db2` vor jedem CLP-Befehl (Kommandozeilen-Modus) oder durch das Wechseln in die interaktive CLP-Umgebung möglich. Zu den Beispielen für die DB2-CLP-Befehle zählen unter anderem die Befehle zum Auslesen, Verändern bzw. Zurücksetzen von Konfigurationseinstellungen des Datenbankmanagers bzw. der Datenbank: `get database (manager) configuration`, `update database (manager) configuration`, `reset database (manager) configuration` (vgl. *Abschnitt 4.3.1.2*). Weitere Beispiele für Befehlszeilenprozessorbefehle sind `runstats` zur Aktualisierung von Statistiken, `reorg` zur Durchführung einer Tabellenreorganisation bzw. `rebind` zum erneuten Binden der Datenbankpakete (vgl. *Abschnitt 3.1.1*).

4.3.2.3 SQL-Anweisungen

Auch mit Hilfe von SQL-Anweisungen lassen sich gewisse Veränderungen eines DB2-Systems vornehmen, die einen Einfluss auf die Performance haben. So kommen beispielsweise beim Anlegen, Löschen bzw. Verändern von Datenbankobjekten entsprechende SQL-Anweisungen zum Einsatz. Beispiele hierfür sind `create tablespace`, `create index`, `drop index`, `create bufferpool`, `alter bufferpool` etc.

4.3.2.4 Betriebssystembefehle

Wie bereits in *Abschnitt 4.3.1.1* erwähnt, lassen sich ausgewählte Konfigurationseinstellungen mit Hilfe von Umgebungsvariablen auf der Betriebssystemebene vornehmen. Entsprechende Anweisungen zum Setzen von Umgebungsvariablen gehören somit ebenfalls zu den Effektoren eines DB2-Systems.

4.4 Formalisierte Abbildung der Tuning-Abläufe durch Tuning-Pläne

Als ein *Tuning-Plan (TP)* wird in der vorliegenden Arbeit ein auf die Lösung eines bestimmten Problems bzw. einer Problemklasse zugeschnittener Workflow bezeichnet, dessen Aktivitäten atomare Datenbank-Tuning-Aufgaben umsetzen und der mit entsprechenden Metadaten angereichert ist [RWRA08]. Aus logischer Sicht besteht ein Tuning-Plan somit aus einem *Header*, welcher die Metadaten des Tuning-Plans enthält, sowie einen den eigentlichen Tuning-Algorithmus enthaltenden *Body*.

Die im Tuning-Plan-Header enthaltenen Metadaten umfassen einerseits die den Tuning-Plan beschreibenden Daten. Anhand von diesen deskriptiven Metadaten lassen sich unter anderem die zur Lösung eines vorliegenden Problems am besten geeigneten Tuning-Pläne zur Laufzeit bestimmen. Andererseits gehören dazu aber auch die Statistiken bezüglich der vergangenen Ausführungen, ihrer Ausführungskosten und ihrer Effektivität. Diese Ausführungsmetadaten helfen bei der Optimierung der Tuning-Plan-Auswahl (*Abschnitt 6.1*).

Tuning-Pläne werden den Events zugeordnet, die das Problem repräsentieren. Beim Auftreten eines entsprechenden (atomaren oder komplexen) Events wird der damit verknüpfte Tuning-Plan initiiert. Liegen mehrere Tuning-Pläne vor, die dem aufgetretenen Event zugeordnet sind, so kann anhand der Metadaten einzelner Tuning-Pläne bestimmt werden, welcher einzelner Tuning-Plan bzw. welche Tuning-Pläne und in welcher Reihenfolge zur Behebung des erkannten Problems ausgeführt werden soll(en). Weiterführende Informationen zur Auswahl bzw. Ausführung von Tuning-Plänen zur Laufzeit finden sich in *Kapitel 6*.

Für die Umsetzung des Tuning-Plan-Body kann grundsätzlich eine beliebige Programmiersprache eingesetzt werden. Wir beschränken jedoch die Komplexität und Erhöhen die Benutzerfreundlichkeit bei der Erstellung von Tuning-Plänen durch Konzeption und Bereitstellung einer Menge von vordefinierten Datenbank-Tuning-spezifischen Aktivitäten, die Basisaktionen des Datenbank-Tuning repräsentieren. Instanzen dieser Aktivitäten bezeichnen wir im Folgenden als *Tuning-Schritte*. Diese können DBA mit Hilfe von Kontrollstrukturen wie Sequenzen, Verzweigungen oder Schleifen zu komplexen Tuning-Abläufen (Tuning-Plänen) verknüpfen.

4.4.1 Tuning-Aktivitäten

In den nachfolgenden Unterabschnitten erfolgt eine Vorstellung von Tuning-Aktivitäten aus zwei Sichtweisen. Die Basisaktivitäten bilden die Grundbausteine eines jeden Datenbank-Tuning-Ablaufs (*Abschnitt 4.4.1.1*). Durch die Zusammenfassung dieser in einzelne Schnittstellen, die für jede Systemkomponente zu definieren sind, wird eine leichte Erweiterbarkeit des Systems gewährleistet. Die auf den Basisaktivitäten aufbauende Schicht mit semantisch angereicherten Tuning-Aktivitäten unterstützt die Planungs- sowie die Ausführungsphase und erhöht die Benutzerfreundlichkeit durch eine Gruppierung der Tuning-Aktivitäten nach ihren Anwendungsgebieten (*Abschnitt 4.4.1.2*). Reicht die Mächtigkeit der vordefinierten Aktivitäten nicht aus, so können die Benutzer eigene Aktivitäten definieren (*Abschnitt 4.4.1.3*) und unter Umständen Tuning-Plan-übergreifend verwenden oder sogar an andere Community-Mitglieder (*Kapitel 7*) weitergeben.

4.4.1.1 Basisaktivitäten

Wie in den vorangegangen Abschnitten gesehen, lässt sich der Großteil des Tuning eines DB2-Systems mit Hilfe von DB2-Befehlen, SQL-Anweisungen sowie Befehlen des entsprechenden Betriebssystems durchführen. Eine zu konzipierende Datenbank-Tuning-Sprache hat also insbesondere diese Befehlsarten zu unterstützen und entsprechende Aktivitäten dafür bereitzustellen.

Oft lassen sich jedoch die erkannten Probleme nicht automatisch beseitigen, da beispielsweise Eingriffe in die Anwendungsprogramme (z. B. bei mangelnden *Commits*), Semantikverändernde Aktionen (z. B. Änderung der Isolationsstufen) oder beispielsweise das Anschließen bzw. Einrichten zusätzlicher Hardware notwendig sind (vgl. *Abschnitt 4.2*). Für solche Fälle sind Werkzeuge bereitzustellen, um Administratoren in den Entscheidungs- bzw. Tuning-Prozess mit einbeziehen zu können. Dabei können die Administratoren einerseits über die erkannten Probleme (beispielsweise per E-Mail oder SMS) benachrichtigt werden. Der Benachrichtigung können unter Umständen Empfehlungen zur Problemlösung beigefügt werden. Andererseits kann der Administrator zur Freigabe von Tuning-Aktionen aufgefordert werden.

Des Weiteren kann im Rahmen des Datenbank-Tuning die Notwendigkeit zum Zugriff auf Schnittstellen weiterer Komponenten des zu tunenden Software-Stacks (z. B. *Application Server*) bzw. anderer Tools (z. B. *IBM DB2 Performance Expert* bzw. *IBM Tivoli Monitoring for Databases*) entstehen.

Aufgrund der hohen Dynamik bezüglich der eingesetzten Software in den IT-Infrastrukturen ist die Datenbank-Tuning-Sprache so zu konzipieren, dass die Einführung neuer Tuning-Aktivitäten zum Zugriff auf neu hinzugekommene Systemkomponenten einfach möglich ist. Durch die Kapselung von Tuning-Aktivitäten in entsprechende für jede Systemkomponente zu definierende Schnittstellen, welche jeweils die Sensor- und Effektor-Funktionalitäten umsetzen, kann diese Forderung erfüllt und die Übersichtlichkeit gewahrt werden. Zur Integration einer neuen Schicht in die Infrastruktur muss somit lediglich eine entsprechende Schnittstelle mit Tuning-Aktivitäten zum lesenden sowie schreibenden Zugriff auf die Konfigurationsparameter der neuen Systemkomponente definiert und in das System integriert werden.

Nachfolgend werden die für unser System (vgl. *Abschnitt 5.2*) identifizierten Schnittstellen exemplarisch beschrieben:

Betriebssystem (z. B. Microsoft Windows, WIN): Die Betriebssystemschnittstelle umfasst unter anderem Aktivitäten zum:

- Lesen und Setzen von Umgebungsvariablen,
- Lesen und Schreiben von Dateien und
- Ausführen von Skripten.

IBM DB2 Universal Database (DB2): Die DB2-Schnittstelle deckt Anweisungen zum Ausführen folgender Befehlsarten ab:

- DB2-System-Befehle (z. B. `db2start`/`db2stop`, `db2advis`, `db2expln`, `db2set`);
- DB2-CLP-Befehle (z. B. `update dbm`/`db cfg`, `runstats`/`reorg`/`rebind`);

- 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): Diese Schnittstelle ist für die Tuning-Aktivitäten zur Einbeziehung von Administratoren zuständig. Dazu zählen beispielsweise E-Mail- bzw. SMS-Benachrichtigung der Administratoren bzw. ihre Aufforderung zur Handlung.

IBM DB2 Performance Expert (PE): Die PE-Schnittstelle ermöglicht den Zugriff auf den in unserer Architektur (*Abschnitt 5.2*) eingesetzten und im *Abschnitt 3.3.2.2* vorgestellten PE, um zur Laufzeit das Überwachungsverhalten des PE den Gegebenheiten anzupassen. So kann beispielsweise die PE-Ausnahmeverarbeitung konfiguriert bzw. angestoßen werden.

ATE: Neben den soeben vorgestellten Schnittstellen spielt im *Autonomic Tuning Expert (ATE)* (*Abschnitt 5.2*) die Schnittstelle zum Zugriff auf die Metadaten und die Konfigurationseinstellungen des ATE selbst eine wichtige Rolle. Damit lassen sich zur Laufzeit (z. B. in Reaktion auf bestimmte Ereignisse) Änderungen des Tuning-Verhaltens des ATE beispielsweise durch Modifikationen der Events und der Tuning-Pläne erreichen.

Die beiden Schnittstellen für den Zugriff auf die Daten bzw. Konfigurationseinstellungen des ATE und PE ermöglichen eine Modifikation des Tuning- bzw. des Überwachungs-Verhaltens zur Laufzeit. Tuning-Pläne, die solche Änderungen mit Hilfe entsprechender Schnittstellen vornehmen, bezeichnen wir als *Meta-Tuning-Pläne*.

4.4.1.2 Semantikreiche Aktivitäten

Bei den im vorangegangenen Abschnitt vorgestellten Tuning-Aktivitäten handelt es sich um generische Basisaktivitäten. Auf (gegebenenfalls komplexe) Abfolgen dieser lässt sich der Großteil typischer Datenbank-Tuning-Aufgaben zurückführen. Die Teilaufgaben, die sich mit den einzelnen Aktivitäten umsetzen lassen, sind dabei sehr vielseitig. Bei einer SQL-Anweisung kann es sich beispielsweise um eine SELECT-Anweisung zum Auslesen bestimmter Daten aus dem Datenbankkatalog handeln. Andererseits können mit Hilfe einer SQL-Anweisung eine Tabelle, ein Index oder eine *Stored Procedure* erstellt werden.

Zur Unterstützung der Community-Funktionen (*Kapitel 7*) und insbesondere zur Ermöglichung der Planung zur Laufzeit ist eine Erfassung der Semantik einzelner Tuning-Schritte bzw. gesamter Tuning-Pläne unabdingbar. Diese deskriptiven Informationen können auch bei der Ausführung der Tuning-Pläne berücksichtigt werden (vgl. *Abschnitt 6.2*).

Viele Informationen über die Verwendung einzelner Aktivitäten lassen sich durch das Parsen dieser ermitteln. So kann beispielsweise automatisch festgestellt werden, *welche* Datenbankobjekte, *wie* (lesend oder schreibend) und *mit welchen Nebenbedingungen* (z. B. bei welchen Attributen ein Lese- oder Schreibvorgang stattgefunden hat) zugegriffen werden. Zur Steigerung der Planungseffizienz bzw. zur Erhöhung der Mächtigkeit der Aktivitätsbeschreibung ist es jedoch unter Umständen von Bedeutung, weitere Informationen über die einzelnen Aktivitäten zu erfassen. Außerdem sind oft zur Erledigung einer einfachen Tuning-Aufgabe, wie beispielsweise einem relativen Verändern eines Konfigurationswerts,

mehrere Aktivitäten notwendig: So muss zum einen der aktuelle Wert des Konfigurationsparameters ausgelesen werden. Erst dann können der neue Wert berechnet und die Belegung des Konfigurationsparameters geändert werden.

Es erscheint daher aus der Benutzersicht die Bereitstellung einer Zwischenschicht sinnvoll, die auf den Basis-Tuning-Aktivitäten aufbaut und für einzelne Anwendungsbereiche entsprechende Aktivitäten definiert. Durch das Zurückgreifen auf die vordefinierten atomaren Tuning-Aktivitäten bei der Definition von Tuning-Abläufen werden dabei Rückschlüsse auf die Semantik dieser Aktivitäten und somit die Semantik der gesamten Tuning-Pläne ermöglicht.

Des Weiteren erfolgt eine Unterstützung der Administratoren durch eine Abstraktion von der DBMS-spezifischen Syntax der eigentlichen Anweisungen und ihre Kapselung in entsprechende Aktivitäten. Nachfolgend findet sich eine kurze Zusammenstellung der nach ihren Anwendungsgebieten kategorisierten, semantisch reichen Tuning-Aktivitäten (vgl. [RWRA08]):

Monitoring-Aktivitäten zum aktiven Sammeln von Überwachungsdaten aus verschiedenen Quellen bzw. zum Anstoßen von Datensammlungsprozessen.

- Zugriff auf den Datenbankkatalog;
- Monitoring-spezifische SQL-Statements;
- Zugriff auf bereits gesammelte Monitoring-Daten bzw. Anstoßen der Datensammlung
- Zugriff auf verschiedene Datenquellen wie *IBM DB2 Performance Expert*, *Tivoli Monitoring for Databases* oder das in [Wie11] eingeführte Performance Warehouse – eine Infrastruktur zur Erfassung sämtlicher Performance-relevanter Metriken;
- (De-)Aktivieren vom Logging/Tracing sowie Zugriff auf die Log-/Trace-Dateien;
- Aufruf von DB2-internen (`db2eva`, `db2diag`, `db2pd`) bzw. -externen (`iostat`, `perfmon`) Tools;
- Anlegen von Tablespaces, Erstellen von Tabellen oder ähnliches zum Ermöglichen einer temporären Speicherung von Daten.

Datenbank-Design-Aktivitäten zum Erstellen/Löschen von Tuning-spezifischen Datenbankobjekten (z. B. Indizes, materialisierte Sichten, Partitionen, Tablespaces, Bufferpools); Aufruf der *Explain Facility* sowie Verwertung ihrer Ergebnisse; Aufruf des *DB2 Design Advisor*.

Konfigurations-Aktivitäten zum (Re-)Allokieren *existierender* Ressourcen (z. B. Sortheap, Bufferpool, Sperrliste, Package Cache etc.) bzw. (De-)Aktivieren von Mechanismen (z. B. *Automatic Statistics Collection*) durch Zugriff auf die Konfigurationswerte. Zu den Konfigurations-Schritten zählen beispielsweise folgende Anweisungen:

- Anweisungen zum lesenden/schreibenden Zugriff auf die Umgebungsvariablen bzw. die Datenbank- sowie Datenbankmanager-Konfiguration;

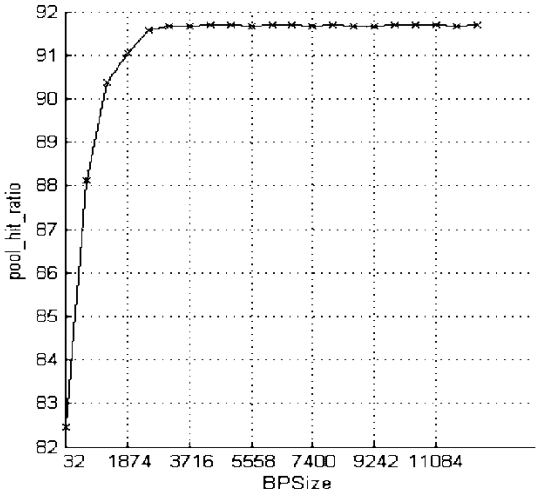


Abbildung 4.4: Exemplarischer Zusammenhang von Bufferpool-Größe und BPHR, abnehmender Grenznutzen [RW07]

- Aufruf des *DB2 Configuration Advisor*, um die vom System gelieferten Konfigurationsvorschläge beispielsweise dem Administrator zu präsentieren oder gar automatisch anzuwenden;

- Operatoren zum komfortablen absoluten bzw. relativen Ändern der Konfigurationswerte. Bei relativen Änderungen der Konfigurationswerte sind dabei einerseits die naheliegenden prozentualen Anpassungen (beispielsweise Erhöhung des Konfigurationswerts um 10%) denkbar. Andererseits ist insbesondere bei Speicherallokationen das *Gesetz des abnehmenden Grenznutzens* (*Sättigungsgesetz*) zu berücksichtigen: Während eine Erhöhung der Größe eines Speicherbereichs anfangs deutliche Performance-Verbesserungen mit sich bringen kann, tritt ab einem bestimmten Zeitpunkt eine Sättigung ein, so dass weitere Speichererhöhungen keinen oder nur einen marginal kleinen positiven Effekt haben (vgl. ***Abbildung 4.4***). Im Extremfall können Ressourcen-Überallokationen sogar für eine schlechtere Performance sorgen. Aus diesem Grund ist die Bereitstellung von Operationen zum Ändern der Konfigurationswerte mit einer dynamischen Schrittweitenanpassung sinnvoll: Die erste Ausführung einer solchen Operation in einem Tuning-Plan würde beispielsweise eine Erhöhung der Bufferpool-Größe um 1.500 Seiten implizieren. Hat die Ausführung dieses Tuning-Plans noch nicht den gewünschten Effekt gebracht und der Event tritt wiederholt ein, so wird bei der zweiten Ausführung dieses Tuning-Plans die gleiche Operation für eine Erhöhung der Bufferpool-Größe nur noch um 800 Seiten sorgen. Eine wiederholte Ausführung dieser Operation würde die Bufferpool-Größe um 200 Seiten erhöhen etc.

Aktivitäten zur Entscheidungsunterstützung zum Initiieren der Sammlung bzw. Interpretation/Mining von Daten zur Einbindung der Entscheidungsunterstützungsfunktionalitäten in die Tuning-Pläne und gegebenenfalls der Verwendung ihrer Ergebnisse in Kontrollstrukturen. Zu den klassischen Beispielen dieser Aktivitätsart zählen An-

weisungen zur Einbindung der Trendanalyse bzw. der Workload-Erkennung in die Tuning-Pläne.

Maintenance-Aktivitäten 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`.

Aktivitäten zur Interaktion mit Nutzern zum Informieren der Administratoren über die vorgenommenen Änderungen oder aufgetretenen Probleme bzw. zum Einbeziehen der Administratoren in die Tuning-Abläufe, die nicht automatisch durchgeführt werden können (beispielsweise bei Semantik-verändernden Operationen wie dem Ändern des *DB2 Isolation Level*). Die typischen Beispiele für diese Art der Tuning-Aktivitäten bilden:

- Anweisungen zum Bereitstellen von voraggregierten, problembeschreibenden Daten für den DBA;
- Anweisungen zum Auffordern des DBAs zur Bestätigung von Tuning-Aktionen.

Aktivitäten zur Interaktion mit Nutzeranwendungen zum Aufrufen von Tools und Skripten, deren Semantik dem ATE unbekannt ist. Durch solche Anweisungen soll der Zugriff beispielsweise auf DB2-externe Tools, Skripte und Services sowie DB2-interne Aufrufe von *Stored Procedures* bzw. *User-Defined Functions* ermöglicht werden.

Aktivitäten 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-Rekonfigurationsaktivitäten sind sogenannte *Meta-Aktivitäten* 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. Die Beispiele für ATE-Rekonfigurationsaktivitäten umfassen:

- Anpassung von PE-Schwellwerten bzw. ACT-Korrelationsregeln (vgl. *Abschnitt 5.2*);
- (De-)Aktivieren von Tuning-Plänen und entsprechenden Events;
- Beeinflussen der Planungslogik;
- Anpassen von Ressourcen-Restriktionen (vgl. *Abschnitt 6.2.2*).

Die Zwischenschicht mit den semantisch angereicherten Aktivitäten baut auf den in entsprechende Schnittstellen zusammengefassten Basisaktivitäten auf (***Abbildung 4.5***). Während einige semantisch reiche Aktivitäten direkt auf die einzelnen zugrunde liegenden Basisaktivitäten abgebildet werden können, setzen sich andere aus Abfolgen mehrerer Basisaktivitäten zusammen. Die Abbildung zwischen den beiden Aktivitätsschichten ist somit in einigen Fällen nicht trivial. In ***Tabelle 4.1*** wird daher die Zuordnung zwischen den Schnittstellen und den Kategorien der Semantik-angereicherten Aktivitäten zusammenfassend dargestellt [RWRA08].

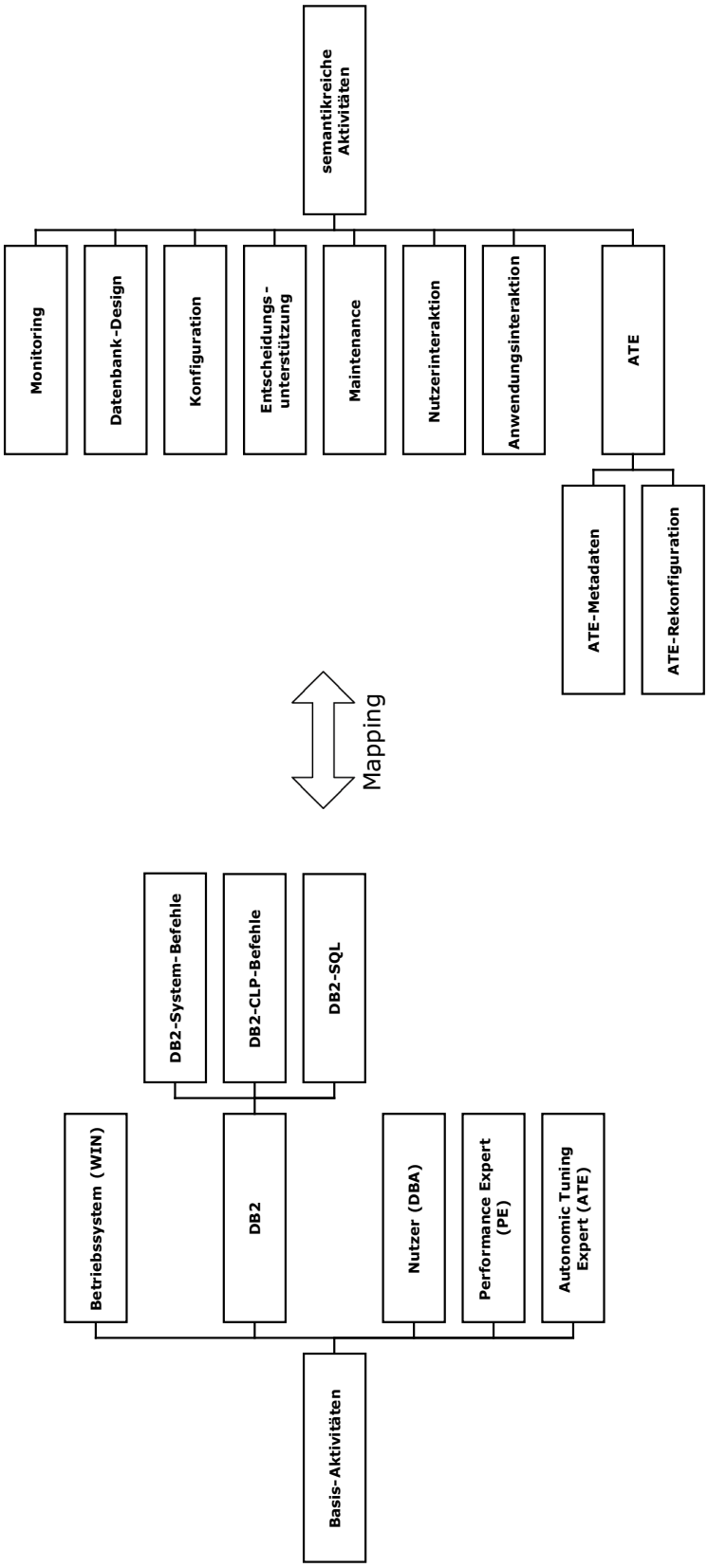


Abbildung 4.5: Kategorisierung von Tuning-Aktivitäten

		Schnittstellen				
		WIN	**DB2**	**DBA**	**PE**	**ATE**
Kategorien	**Monitoring**	+	+		+	
	Datenbank-Design		+			
	Konfiguration	+	+			
	Entscheidungsunterstützung	+	+		+	+
	Maintenance		+			
	Nutzer-Interaktion			+		
	Tool-Interaktion	+	+		+	
	ATE-Metadaten-Zugriff				+	+
	ATE-Rekonfiguration				+	+

Tabelle 4.1: Zuordnung von Anwendungsgebieten der semantikreichen Aktivitäten zu Komponenten-Schnittstellen

4.4.1.3 Benutzerdefinierte Aktivitäten

Zwar lässt sich der Großteil typischer Datenbank-Tuning-Aufgaben auf Abfolgen generischer Basisaktivitäten zurückführen, die Menge der darauf aufbauenden, vordefinierten semantikreichen Aktivitäten kann jedoch unter Umständen zur Umsetzung bestimmter Aufgaben nicht ausreichend sein. Einerseits kann in solchen Fällen auf die entsprechenden Basisaktivitäten zurückgegriffen und damit können die gewünschten Funktionalitäten umgesetzt werden. Andererseits ist es aus Gründen der höheren Wiederverwendbarkeit sinnvoll, eine Kapselung des diese Funktionalitäten umsetzenden Codes in *benutzerdefinierte Aktivitäten* zu ermöglichen.

In Abhängigkeit vom Grad der Wiederverwendbarkeit lassen sich die benutzerdefinierten Tuning-Aktivitäten in zwei Kategorien einordnen: *Lokale (Tuning-Plan-interne) Aktivitäten* sind für eine mehrfache Verwendung innerhalb eines bestimmten Tuning-Plans konzipiert. Die *globalen (Tuning-Plan-übergreifenden) Aktivitäten* lassen sich dagegen in verschiedenen Tuning-Plänen einsetzen. An dieser Stelle ist zu erwähnen, dass es sich bei den in *Abschnitten 4.4.1.1* und *4.4.1.2* vorgestellten Basis- sowie semantikreichen Aktivitäten um globale Aktivitäten handelt, die für die Benutzung in beliebigen Tuning-Plänen konzipiert sind.

Während die Semantik vordefinierter Aktivitäten eindeutig festgelegt ist, ist bei den benutzerdefinierten Tuning-Aktivitäten die Erfassung ihrer Semantik unverzichtbar, um beispielsweise die Vergleichbarkeit einzelner Aktivitäten untereinander zu ermöglichen. Zur Beschreibung der Semantik kann dabei beispielsweise auf die Verwendung von Taxonomien oder Ontologien zurückgegriffen werden. So lässt sich damit beispielsweise ein kontrolliertes, Datenbank-Tuning-spezifisches Vokabular erstellen. Mit einzelnen Begriffen dieses Vokabulars lassen sich anschließend die benutzerdefinierten Aktivitäten durch ihre Autoren verknüpfen. Die Semantik der Aktivitäten kann nun durch die Auswertung der mit den Aktivitäten verknüpften Begriffe ermittelt werden.

Eine ähnliche Vorgehensweise bietet sich auch zur Beschreibung der Semantik ganzer Tuning-Pläne an und wird in *Abschnitt 7.1.6* ausführlich diskutiert. Insbesondere findet sich dort eine kurze Gegenüberstellung von Taxonomien, Folksonomien und Ontologien. Auf weitere Ausführungen bezüglich der Semantikerfassung der benutzerdefinierten Tuning-Aktivitäten mit Hilfe von Taxonomien bzw. Ontologien wurde im Rahmen dieser Arbeit jedoch verzichtet. An dieser Stelle sei vollständigkeitshalber auf Ontologie-basierte Forschungsansätze zur Erfassung der Semantik von *Web Services* (*Semantic Web Services*) [MSZ01] bzw. die daraus abgeleiteten Ansätze zur Beschreibung der Semantik von Geschäftsprozessen (*Semantic Business Process Management*) [HLD+05] verwiesen.

4.4.2 Variablen und Kontrollstrukturen

Wie bereits aus der Beschreibung der semantikreichen Aktivitäten (*Abschnitt 4.4.1.2*) ersichtlich, können Tuning-Aktivitäten Daten sammeln bzw. auswerten. Um ein Zwischenspeichern und einen Austausch dieser Daten zwischen einzelnen Aktivitäten zu ermöglichen, sind spezielle Datencontainer notwendig. Während für einzelne Werte einfache Variablen ausreichen, müssen für das Zwischenspeichern größerer Datenmengen spezielle Datenstrukturen bereitgestellt werden.

Für die Umsetzung solcher Datenstrukturen sind zwei Vorgehensweisen vorstellbar. Zum einen kann die Tuning-Plan-Modellierungssprache um Programmiersprachen-ähnliche Datenstrukturen wie Arrays, Vektoren oder ähnliches erweitert werden. Der Vorteil dieser Variante liegt unter anderem in einem unkomplizierten Datenzugriff und somit der Möglichkeit einer effizienten Einbindung der Variablen in die Steuerung des Kontrollflusses. Andererseits benötigt diese Umsetzungsvariante entsprechende Konzepte und Technologien zur Sicherstellung der Datenpersistierung.

Alternativ dazu lassen sich die benötigten Datenstrukturen mit Hilfe von Datenbankobjekten wie Tabellen bzw. temporären Tabellen umsetzen. Zwar sind bei dieser Variante entsprechende Schnittstellen für den Datenzugriff bereitzustellen, der Vorteil ist jedoch ganz klar die durch die DBMS-Mechanismen gesicherte Datenpersistenz. Der Datenzugriff lässt sich außerdem durch die Bereitstellung einer Zwischenschicht, die deskriptive Anfragen, wie beispielsweise *„Gib mir alle Daten zum Bufferpool* `IBMDEFAULTBP` *der Datenbank* `TERRA`*, die von den Tuning-Plänen X und Y in den letzten 4 Tagen gesammelt wurden“* in entsprechende SQL-Anweisungen umwandelt und die angefragten Daten bereitstellt, deutlich verbessern. Diese Zwischenschicht könnte weiterhin auch für die Sicherstellung der Zugriffsrechte verwendet werden, damit nur ausgewählte Tuning-Pläne auf die durch andere Tuning-Pläne gesammelten Daten zugreifen können.

Des Weiteren werden Kontrollstrukturen benötigt, um einzelne Tuning-Schritte in Abhängigkeit von den Datenauswertungsergebnissen auszuführen bzw. zu überspringen. Die Basiskonstrukte sind dabei Sequenz, Verzweigung (bedingt sowie parallel) und Wiederholung. Abgesehen von der parallelen Verzweigung, die insbesondere im Kontext von Workflow-Management-Systemen Einsatz findet, wurden die oben genannten Basiskonstrukte bereits in [DH72] identifiziert. Mit Hilfe von sogenannten Blockkonstrukten lassen sich einzelne Aktivitäten zu Sub-Abläufen zusammenfassen. Zwar sind solche Blockkonstrukte sehr hilfreich beim hierarchischen Strukturieren von Arbeitsabläufen, jedoch für die Umsetzung von Tuning-Plänen nicht unbedingt notwendig.

Die Mächtigkeit einiger Basiskonstrukte kann durch ihre Parametrisierung erhöht werden. So lässt sich beispielsweise das Sequenzkonstrukt mit einer Bedingung versehen, die für einen Übergang von einer Tuning-Aktivität zur nächsten erfüllt sein muss. Eine solche Bedingung könnte einerseits eine Zeitangabe beinhalten, die zwischen den beiden Aktivitäten verstreichen muss. Andererseits könnte man sich vorstellen, dass die Bedingung das Auftreten eines bestimmten Events bzw. das Vorliegen bestimmter für die nachfolgende Aktivität benötigter Daten sicherstellt.

4.5 Modellierung von Tuning-Plänen

Nachdem in *Abschnitt 4.4.1* Datenbank-Tuning-spezifische Aktivitäten vorgestellt wurden, die sich mit Hilfe von entsprechenden Kontrollstrukturen wie Sequenzen, Verzweigungen oder Schleifen zu komplexen Tuning-Abläufen verknüpfen lassen, stellt sich nun die Frage nach einem geeigneten Modellierungswerkzeug zur Erstellung von Tuning-Plänen. Dabei sind Aspekte wie Benutzerfreundlichkeit, leichte Erlernbarkeit, Mächtigkeit und geringer Ressourcenverbrauch zur Modellierungs- sowie Ausführungszeit zu berücksichtigen.

Insbesondere zur Erhöhung der Verständlichkeit und Anschaulichkeit der zu konzipierenden Sprache zur Modellierung von Tuning-Plänen ist die Wahl eines visuellen Ansatzes naheliegend. Einerseits können dafür ein die gestellten Anforderungen erfüllendes Modellierungswerkzeug sowie eine entsprechende Ausführungsumgebung entwickelt werden. Andererseits ist aber auch der Einsatz eines beliebigen Workflow-Management-Systems sowohl zur Modellierung als auch zur Ausführung von Tuning-Plänen möglich.

Die Entwicklung eines Prototyps ist mit Hilfe eines Workflow-Management-Systems relativ schnell möglich, da der Großteil der Modellierungsfunktionen bereits vorliegt. Gegebenenfalls müssen lediglich wenige Anpassungen (wie beispielsweise die Umsetzung und Bereitstellung entsprechender Tuning-Aktivitäten) vorgenommen werden. Auch die Modellierungsmächtigkeit eines WfMS ist sehr hoch. Bei der Tuning-Plan-Modellierung werden jedoch viele der bereitgestellten Funktionalitäten gar nicht gebraucht und erhöhen unnötig die Komplexität der Modellierung. Die Ausführung von Tuning-Plänen, die mit Hilfe eines WfMS umgesetzt wurden, ist durch den Aufruf einer entsprechenden *Execution Engine* relativ einfach. Durch einen hohen Ressourcenverbrauch eines herkömmlichen Workflow-Management-Systems kann jedoch unter Umständen das zu tunende System beeinträchtigt werden. Ein weiterer wichtiger Nachteil ist, dass die *Workflow Execution Engine* kaum erweiterbar ist, so dass die Ausführungslogik im Workflow enthalten sein muss.

Ein eigens für die Modellierung bzw. Ausführung von Tuning-Plänen entwickeltes System kann andererseits optimal auf die Bedürfnisse der Nutzer zugeschnitten werden und bietet eine hohe Flexibilität bei der Ausführung von Tuning-Plänen. Der Implementierungsaufwand ist jedoch sowohl für das Modellierungs- als auch für das Ausführungssystem als sehr hoch einzustufen, weshalb darauf im Rahmen dieser Arbeit weitgehend verzichtet wurde.

Während für den in diesem Abschnitt vorgestellten Modellierungsansatz die Ausführungslogik eines WfMS vollkommen ausreichend ist, erfüllt diese bei der Betrachtung des im *Kapitel 6* vorgestellten Ansatzes einer intelligenten Hypothesenbaum-basierten Tuning-Plan-

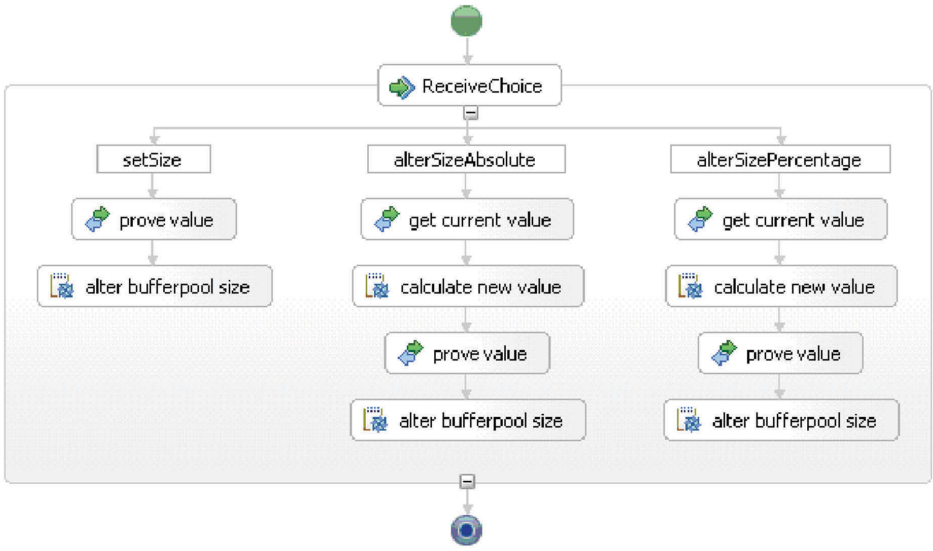


Abbildung 4.6: BPEL-Prozess *AlterBufferpoolSize* [Are08]

Ausführung nicht mehr die gestellten Anforderungen, da dort unter anderem Backtracking-Mechanismen eingesetzt werden und diese sich mit einem herkömmlichen WfMS nur schwer umsetzen lassen. Aus diesem Grund wurde in [Are09] ein entsprechendes Modellierungs- und Ausführungswerkzeug entwickelt (vgl. *Abschnitt 6.3*).

4.5.1 Tuning-Plan-Modellierung mit *IBM WebSphere Integration Developer*

Aufgrund seiner hohen Verbreitung in Unternehmensarchitekturen und aus dem Projekt-Kontext der IBM Kooperation heraus fiel die Wahl bei der Suche nach einem geeigneten Workflow-Management-System zunächst auf *IBM WebSphere Process Server (WPS)*[17]. Dabei handelt es sich um ein Workflow-Management-System, welches die Modellierung und Ausführung von BPEL-Prozessen ermöglicht. Er baut auf dem J2EE-Server *IBM WebSphere Application Server (WAS)* auf und ergänzt diesen um weitere Systemkomponenten wie einen *Service-oriented Architecture (SOA) Core*, bestehend aus der *Service Component Architecture (SCA)*, *Business Objects (BO)* und der *Common Event Infrastructure (CEI)*. Die Entwicklungsumgebung von SOA-basierten Geschäftsprozessen für den WPS ist der *IBM WebSphere Integration Developer (WID)*.

In [Are08] wurde untersucht, inwiefern sich WPS mit seinen entsprechenden Komponenten für die Modellierung und Ausführung von Tuning-Plänen eignet. Bei der Umsetzung einiger Tuning-Aktivitäten kamen dabei BPEL-Erweiterungen wie *BPEL4People* zur Einbeziehung von menschlichen Akteuren in den Workflow, *BPEL4J* zur Einbindung von Java-Code in den Workflow und *II4BPEL* zur Vereinfachung des Datenbankzugriffs aus den Workflows heraus zum Einsatz.

In ***Abbildung 4.6*** wird die BPEL-Umsetzung der Tuning-Aktivität zur Änderung der Bufferpool-Größe exemplarisch veranschaulicht. Die vorliegende Tuning-Aktivität zur Än-

[17] http://www-01.ibm.com/software/integration/wps

derung der Bufferpool-Größe unterscheidet dabei zwischen folgenden drei Arten der Größenänderung:

- `setSize` zum direkten Setzen der Bufferpool-Größe auf einen vorgegebenen Wert
- `alterSizeAbsolute` zu einer Änderung der aktuellen Bufferpool-Größe um eine vorgegebene Anzahl an Seiten
- `alterSizePercentage` zu einer Änderung der aktuellen Bufferpool-Größe um einen vorgegebenen Prozentsatz

Bei einem direkten Setzen der Bufferpool-Größe wird zunächst im Schritt `prove value` überprüft, ob der übergebene Wert die vorgegebene Oberschranke für die Größenbelegung des entsprechenden Bufferpools übersteigt oder nicht. Übersteigt der neue Wert den maximal möglichen Wert, so wird die neue Bufferpool-Größe auf diesen Maximalwert gesetzt. Bei diesem Schritt handelt es sich also um eine einfache Ressourcenbeschränkung, die eine Überallokation der entsprechenden Ressource und somit eine Performance-Abnahme vermeiden soll.

Im Fall einer absoluten Änderung der Bufferpool-Größe wird zunächst die aktuelle Bufferpool-Größe mittels `get current value` ausgelesen. Anschließend wird der neue Wert bestimmt (`calculate new value`) und hinsichtlich der vorgegebenen Obergrenze für die Größe des betrachteten Bufferpools überprüft (`prove value`). Anschließend wird die entsprechende Änderung vorgenommen (`alter bufferpool size`). Bei einer prozentualen Änderung der Bufferpool-Größe ist die Vorgehensweise analog. Für die Beschreibung weiterer mit *IBM WebSphere Process Server* umgesetzter Tuning-Aktivitäten und Tuning-Pläne sei hier auf [Are08] verwiesen.

Obwohl anfangs erwartet wurde, dass die durch *IBM WebSphere Integration Developer* ermöglichte graphische Modellierung von Workflows nach der Bereitstellung einer Menge von Datenbank-Tuning-spezifischen Aktivitäten die Entwicklung von Tuning-Plänen deutlich vereinfacht, haben die in [Are08] durchgeführten Untersuchungen ergeben, dass die hohe Komplexität des verwendeten Workflow-Systems nicht zu vernachlässigen ist. Die Tuning-Plan-Entwicklung mit den bereitgestellten Mitteln hat sich für einen ungeübten Anwender als äußerst aufwendig und fehleranfällig herausgestellt. Dem Großteil der Datenbankadministratoren können jedoch keine Kenntnisse eines komplexen Workflow-Management-Systems abverlangt werden. Nicht zuletzt ist auch der, wie erwartet, durch seine hohe Mächtigkeit bedingte verhältnismäßig hohe Ressourcenverbrauch des *IBM WPS* und der zugehörigen Komponenten zur Laufzeit zu erwähnen, welcher unter Umständen das zu tunende System beeinträchtigen kann.

4.5.2 Tuning-Plan-Modellierung mit *IBM WebSphere sMash*

IBM WebSphere sMash (WsM) – ein im Verlauf unserer Forschungsarbeiten identifiziertes Software Development Framework zur Erstellung von dynamischen Webinhalten – beinhaltet ein leichtgewichtiges System mit einem integrierten webbasierten Editor zur Erstellung von auf Internet-Anwendungen zugeschnittenen Workflows. Sowohl das System selbst als auch der zugehörige Editor zeichnen sich dadurch aus, dass sie einen geringen Ressourcenverbrauch haben und sich mit geringem Aufwand um benutzerdefinierte Aktivitäten erweitern lassen. Daher eignet sich WsM hervorragend zur Modellierung und Ausführung von

Tuning-Plänen [Kar07] und dient als Basis für die Tuning-Plan-Modellierung und Ereignisgesteuerte -Ausführung im Rahmen des *Autonomic Tuning Expert* (*Abschnitt 5.2*).

Das Community-Inkubationsprojekt von *IBM WebSphere sMash* namens *Project Zero*[18] ermöglicht es einer breiten Entwickler-Community, die Entwicklung des Frameworks durch ihre Beiträge (Feedback, Erfahrungen, Ideen, Code-Fragmente etc.) voranzutreiben. In den nicht-kommerziellen Project-Zero-Versionen des Frameworks sind somit typischerweise die neuesten Features enthalten, die in *WebSphere sMash* noch nicht aufgenommen wurden. Für den in *Abschnitt 5.2* beschriebenen Prototyp wurde aus diesem Grund eine Version des Project-Zero-Projekts verwendet. In der vorliegenden Arbeit verwenden wir jedoch die Begriffe *Project Zero* und *WebSphere sMash* aufgrund ihrer geringen Unterschiede als Synonyme.

4.5.2.1 Global Context

WsM wurde dafür konzipiert, eine simple und schnelle Entwicklung von Internet-Anwendungen und dabei eine hohe Wiederverwendbarkeit einzelner Anwendungskomponenten zu ermöglichen. Es basiert auf dem Paradigma der Event-getriebenen Programmierung: Die Kommunikation zwischen den einzelnen Systemmodulen erfolgt mittels entsprechender Events. Um einen Datenaustausch zwischen den Modulen sowie eine Langzeitspeicherung bestimmter Daten zu ermöglichen, wurde das Konzept namens *Global Context* eingeführt. Der *Global Context* ist in diverse Bereiche unterteilt. Dabei wird zwischen persistenten (*Config Zone*, *Request Zone*, *Event Zone*, *Tmp Zone* sowie *Connection Zone*) und nicht-persistenten Bereichen (*User Zone*, *App Zone* sowie *Storage Zone*) unterschieden [IBM11]. Die einzelnen Bereiche haben dabei je nach ihrem Verwendungszweck eine bestimmte Lebensdauer und Sichtbarkeit. Reichen die vordefinierten Bereiche des *Global Context* nicht aus, so können weitere anwendungsspezifische Bereiche erzeugt werden. Für jeden der Bereiche kann dabei sowohl die Sichtbarkeit als auch die Lebensdauer der darin gespeicherten Objekte angegeben werden.

4.5.2.2 Assemble Flow

Ein sehr wichtiger Bestandteil von WsM ist das *Assemble Flow*. Dabei handelt es sich um ein sehr leichtgewichtiges System zur Erzeugung von Internet-Anwendungen (*Flows*). Die Spezialisierung des WsM auf Entwicklung von Internet-Anwendungen macht sich insbesondere durch den Satz bereits vordefinierter Aktivitäten bemerkbar. Besonders interessant ist jedoch die Tatsache, dass WsM so konzipiert ist, dass benutzerdefinierte Aktivitäten (beispielsweise mit Java) implementiert und mit geringem Aufwand in das System integriert werden können. Die vordefinierten bzw. benutzerdefinierten Aktivitäten lassen sich anschließend, wie in jedem anderen WfMS, durch die entsprechenden Kontrollstrukturen miteinander verknüpfen. Hauptsächlich wird dabei der Kontrollfluss durch die Definition entsprechender Kanten zwischen den Aktivitätsknoten festgelegt. Der Datenfluss entspricht im Wesentlichen dem Kontrollfluss. Jede Aktivität hat einen Zugriff auf die Daten ihrer Vorgängeraktivität(en). Um den Aufwand bei einer sich über mehrere Aktivitäten erstreckenden Weitergabe von Daten zu reduzieren, haben die Aktivitäten weiterhin

[18] http://www.ProjectZero.org

Listing 4.1: Aufbau eines WsM-Tuning-Plans

```
<process name="TuningPlanName">
  ...
  <Aktivitaet1 name="TSName1" [Attribut1="Attributwert1" Attribut2="Attributwert2" ...]>
    <control source="VorgängerTSName1" [transitionCondition="Übergangsbedingung1"]/>
    <control source="VorgängerTSName2" [transitionCondition="Übergangsbedingung2"]/>
    ...
    <input name="Eingabeparameter1" value="Parameterwert1"/>
    <input name="Eingabeparameter2" value="Parameterwert2"/>
    ...
  </Aktivitaet1>
  ...
</process>
```

einen Zugriff auch auf die Daten aller ihrer indirekten Vorgänger. Dabei kann jede Aktivität genau ein serialisierbares Datenobjekt in den Datenfluss übergeben. Sollen mehrere Datenobjekte an andere Aktivitäten übergeben werden, so kann auf die Verwendung des *Global Context* zurückgegriffen werden. Neben den Kanten zwischen den einzelnen Aktivitäten, die den Kontrollfluss sowie den Datenfluss festlegen und mit Kontrollfluss-spezifischen Kantenbedingungen versehen werden können, gibt es weitere Kontrollflusselemente. Dazu zählen beispielsweise schachtelbare, Schleifen-umsetzende Aktivitäten `while` bzw. `for-each`.

Wie bereits erwähnt, kann die Modellierung von WsM-Anwendungen mit Hilfe eines entsprechenden Web-basierten Editors vorgenommen werden (vgl. ***Abbildung 4.7***). Die Speicherung der erzeugten Anwendungen erfolgt im WsM-internen XML-Format. Zur Laufzeit werden die auszuführenden Flow-Dateien durch die Ausführungskomponente interpretiert.

Jeder im WsM-Format abgespeicherte Tuning-Plan hat folgenden Aufbau (***Listing 4.1***): Das XML-Wurzelelement heißt `process` und hat als Attribut einen entsprechenden Tuning-Plan-Namen. Die Kindknoten des Elements `process` sind die einzelnen Aktivitäten des Flows. Dabei ist wichtig anzumerken, dass es sich bei Flow-Dateien um ungeordnete XML-Dokumente handelt. Das bedeutet, dass die Reihenfolge der Element-Knoten implementierungsabhängig sein und sich bei einem Laden-Speichern-Zyklus verändern kann (vgl. *Abschnitt 7.1.2.8*). Aus diesem Grund ist insbesondere die Reihenfolge der Aktivitäten in der Flow-Datei nicht von Bedeutung und hat keinen Einfluss auf die Abarbeitungsreihenfolge, die sich aus dem entsprechenden Kontrollfluss des Flows ergibt.

Neben einem eindeutigen Namen kann eine Aktivitätsinstanz Parameter haben (`Attribut`$_i$). Diese werden hauptsächlich für eine Parametrisierung von Aktivitäten verwendet und können ausschließlich Werte primitiver Datentypen annehmen.

Die den Kontrollfluss repräsentierenden Kanten zwischen den einzelnen Aktivitätsinstanzen (Tuning-Schritten) werden im Flow-Format als XML-Elemente mit dem Namen `control` repräsentiert. Jedes `control`-Element hat dabei eine Quelle (`source`), die den Vorgänger-Tuning-Schritt durch die Angabe seines Namens eindeutig identifiziert. Dabei kann weiterhin eine Übergangsbedingung (`transitionCondition`) angegeben werden. Die Ausführungskomponente wertet zur Laufzeit den Bedingungsausdruck aus. Der aktuelle Tuning-Schritt wird dabei nur im Fall einer positiven Auswer-

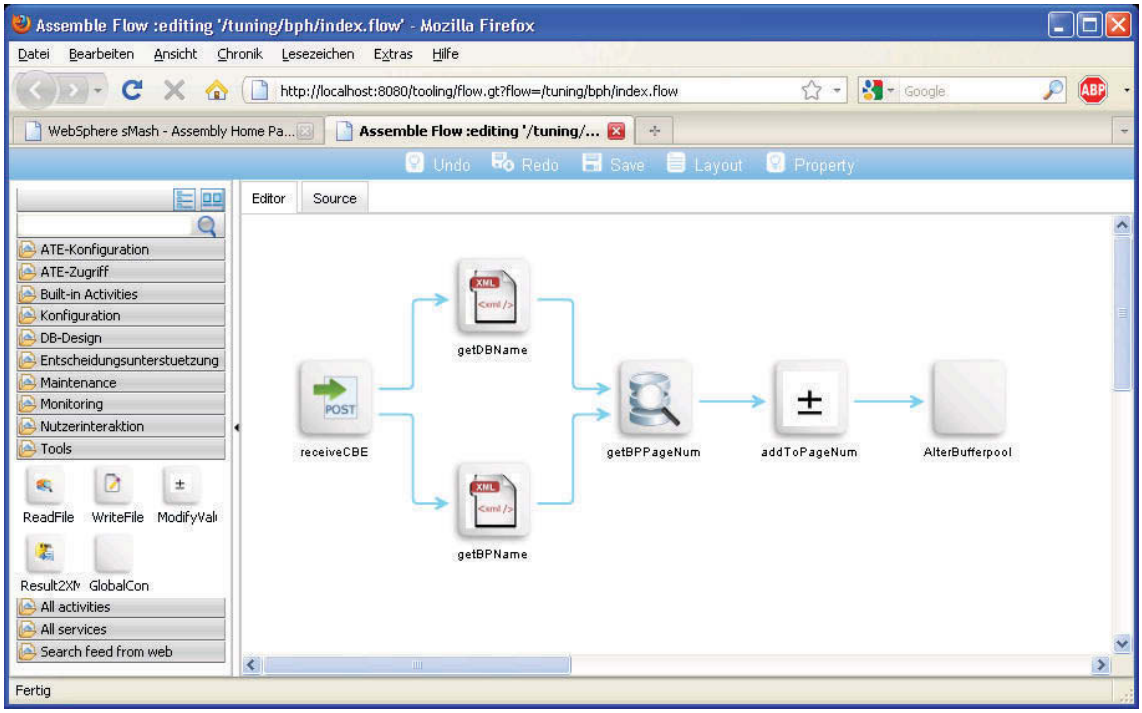


Abbildung 4.7: Tuning-Plan-Modellierung am Beispiel von „*Bufferpool-Tuning*“

tung (`transitionCondition`==true) dieses Ausdrucks ausgeführt. Hat ein Tuning-Schritt mehrere `control`-Elemente, so kann er erst dann ausgeführt werden, wenn alle seine Vorgänger beendet wurden und die gegebenenfalls angegebenen Übergangsbedingungen als positiv ausgewertet wurden.

Während der Kontrollfluss durch die `control`-Verbindungen definiert wird, kann mit Hilfe von `input`-Elementen der Datenfluss definiert werden. Für jeden Eingabewert wird ein entsprechendes XML-Element `input` definiert. Das Element wird mit einem Namen versehen und es wird angegeben, woher der Wert dieses Elements zu beziehen ist (***Listing 4.2***).Soll der Wert beispielsweise von einer Vorgängeraktivitätsinstanz namens *Determine_BPHR* übergeben werden, so wird dies entsprechend im `value`-Attribut des Elements `input` folgendermaßen verzeichnet: `value="${Determine_BPHR}"`. Dadurch erfolgt der Zugriff auf den für jeden Tuning-Schritt implizit erzeugten Ausgabeparameter (hier Ausgabeparameter des Tuning-Schritts *Determine_BPHR*). Hat eine Aktivität ein `input`-Element, dessen Wert direkt aus einer anderen Aktivität übernommen wird, so kann auf die Angabe des entsprechenden `control`-Elements verzichtet werden (*control shortcut*) [IBM11]. Die Anwendung dieses *control shortcut* findet beispielsweise in *Abbildung 4.7* statt.

Die meisten Aktivitätsinstanzen besitzen sowohl `input`- als auch `control`-Elemente, wobei auf eine explizite Modellierung der `control`-Elemente oft verzichtet werden kann. Der erste Tuning-Schritt eines Flows hat jedoch keine Vorgänger, so dass für ihn entsprechend kein `control`-Element (und typischerweise keine `input`-Elemente) definiert sind.

Listing 4.2: WsM-interne XML-Repräsentation des Tuning-Plans „*Bufferpool-Tuning*“

```
<process name="simplified_Bufferpool_Tuning">
  <receivePOST name="receiveCBE"/>
  <GetCBEElement name="getDBName" Element="db_name">
    <input name="CBEEventFile" value="${receiveCBE}" content-type="text/xml"/>
  </GetCBEElement>
  <GetCBEElement name="getBPName" Element="bp_name">
    <input name="CBEEventFile" value="${receiveCBE}" content-type="text/xml"/>
  </GetCBEElement>
  <QueryCatalog name="getBPPageNum" Where="bpname='${getBPName}'"
    Select="npages" From="syscat.bufferpools">
    <input name="db_name" value="${getDBName}" content-type="text/plain"/>
  </QueryCatalog>
  <ModifyValue name="addToPageNum" modification="+1000">
    <input name="inValue" value="${getBPPageNum[0].get(&amp;quot;npages&amp;quot;)}"
      content-type="text/plain"/>
  </ModifyValue>
  <AlterBufferpool name="AlterBufferpool" bpName="${getBPName}"
    immediateORdeferred="immediate" size="${addToPageNum}">
    <input name="db_name" value="${getDBName}" content-type="text/plain"/>
  </AlterBufferpool>
</process>
```

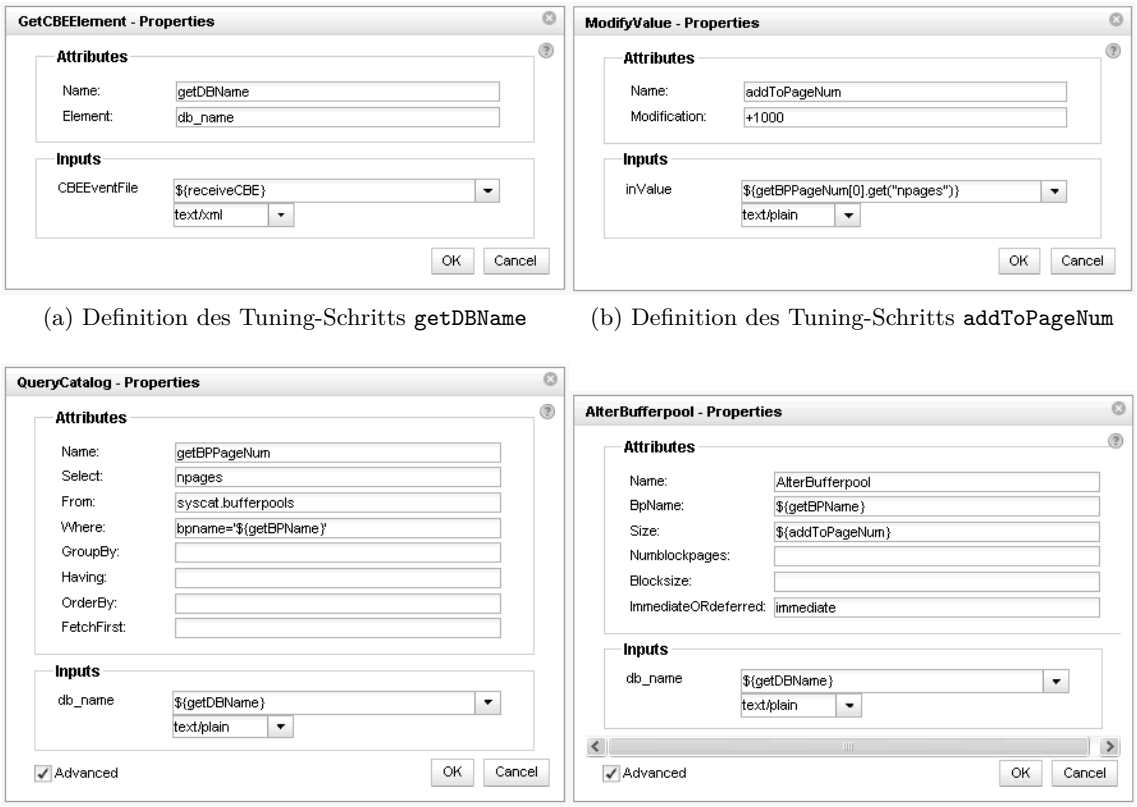
4.5.2.3 WsM-Modellierung am Beispiel vom Bufferpool-Tuning

Anhand eines vereinfachten Tuning-Plans zum Optimieren der *Bufferpool Hitratio* soll nachfolgend die Modellierungsmächtigkeit des auf WsM aufbauenden und auf Datenbank-Tuning zugeschnittenen Modellierungswerkzeugs veranschaulicht werden.

Der entsprechende Tuning-Plan zum Bufferpool-Tuning ist in *Abbildung 4.7* und seine WsM-interne XML-Repräsentation in *Listing 4.2* dargestellt. Der Tuning-Plan, angestoßen beim Fallen der *Bufferpool Hitratio* unter eine bestimmte Grenze, bestimmt die aktuelle Bufferpool-Größe durch den Zugriff auf den Datenbankkatalog und erhöht diese anschließend um eine bestimmte Anzahl an Seiten. Bei der Erstellung des Tuning-Plans wurde auf die entsprechenden benutzerdefinierten Aktivitäten zurückgegriffen. Eine Auswahl dieser Aktivitäten, geordnet nach Kategorien (vgl. *Abschnitt 4.4.1*), ist im linken Teil der Abbildung zu erkennen.

Der erste Tuning-Schritt des Tuning-Plans `receiveCBE` ermöglicht den Zugriff auf die im übergebenen *Common Base Event* enthaltenen Metadaten zum Kontext des aufgetretenen Problems. Mit Hilfe der beiden Instanzen der Aktivität `GetCBEElement` namens `getDBName` sowie `getBPName` werden Informationen über die Problemdatenbank sowie der betroffene Bufferpool extrahiert. In ***Abbildung 4.8a*** erkennt man beispielsweise, dass das Element namens `db_name` aus der XML-Element `receiveCBE` ausgelesen wird.

Der Tuning-Schritt `getBPPageNum` – eine Instanziierung der Aktivität `QueryCatalog` – ermittelt durch den Zugriff auf die Systemkatalogsicht `SYSCAT.BUFFERPOOLS` die aktuelle Größe des Bufferpools, dessen Name im Schritt `getBPName` aus dem übergebenen Event ausgelesen wurde (***Abbildung 4.8c***). An dieser Stelle wurde zu Demonstrationszwecken absichtlich auf die Verwendung einer generischen Aktivität zurückgegriffen, die das Absetzen beliebiger SQL-Anweisungen zum Zugriff auf die Systemkataloginformationen er-



(a) Definition des Tuning-Schritts `getDBName` (b) Definition des Tuning-Schritts `addToPageNum`

(c) Definition des Tuning-Schritts `getBPPageNum` (d) Definition des Tuning-Schritts `AlterBufferpool`

Abbildung 4.8: Ausgewählte Tuning-Schritte des WsM-Tuning-Plans „*Bufferpool-Tuning*“

möglicht. Der Einsatz einer speziell zum Auslesen von Bufferpool-spezifischen Informationen wie beispielsweise Bufferpool-Größe, Seitengröße bzw. Blockgröße [IBM06g] aus der Systemkatalogsicht `SYSCAT.BUFFERPOOLS` entwickelten Aktivität hätte insbesondere zur Laufzeit den Vorteil, dass auf eine Auswertung der SQL-Anweisung zur Ableitung der Semantik verzichtet werden könnte.

Der Tuning-Schritt `addToPageNum` – eine Instanziierung von der Aktivität `ModifyValue` – erhöht die ausgelesene Bufferpool-Größe um 1.000 Seiten (***Abbildung 4.8b***) und schließlich sorgt der Schritt `AlterBufferpool` dafür, dass die Größe des Bufferpools auf den neuen Wert gesetzt wird (***Abbildung 4.8d***).

4.6 Zusammenfassung

Nachdem im *Kapitel 3* einige Konzepte und Werkzeuge zur Beschreibung von (Datenbank-)Problemen untersucht und vorgestellt wurden, wurde im vorliegenden Kapitel ein Vorschlag für eine Datenbank-Tuning-spezifische Sprache, die eine formalisierte Erfassung von typischen Tuning-Vorgehensweisen der Datenbankadministratoren ermöglicht, unterbreitet. Der Zusammenhang zwischen den Probleme repräsentierenden Ereignissen und

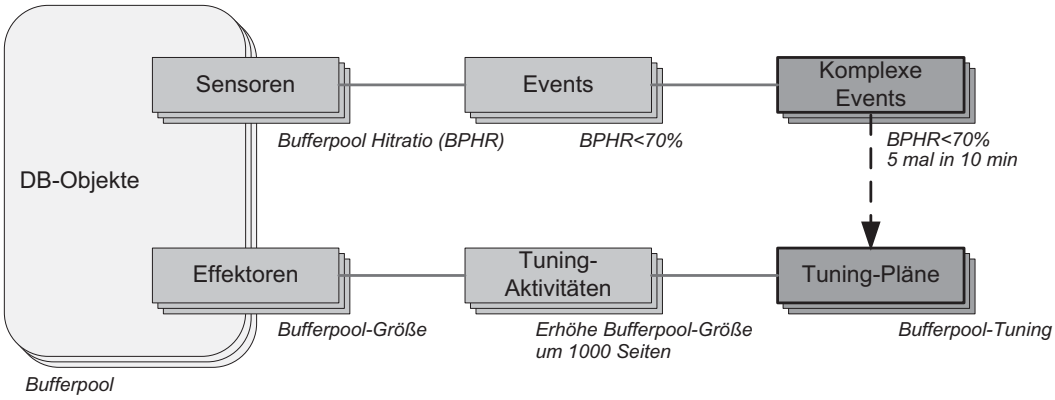


Abbildung 4.9: Zusammenhang zwischen Sensoren, Events, Effektoren und Tuning-Plänen

den hier eingeführten Tuning-Plänen wird in ***Abbildung 4.9*** anhand eines Beispiels verdeutlicht.

Wie bereits in *Kapitel 2* erläutert, besitzt jedes Objekt eine Sensor- und eine Effektor-Schnittstelle. Mit Hilfe der Sensoren einzelner Objekte lassen sich Ereignisse definieren, die zur Laufzeit durch eine entsprechende Überwachungskomponente erkannt werden. Zur Minimierung der Ereignisflut und insbesondere aufgrund der angestrebten Integration von typischerweise in komplexen IT-Infrastrukturen bereits vorhandenen, sich über mehrere Schichten der IT-Infrastruktur erstreckenden Überwachungswerkzeugen hat sich ein zweistufiger Ansatz als effektiv erwiesen. Im ersten Schritt erfolgt dabei die Definition von atomaren Ereignissen, die die problematischen Situationen repräsentieren. Im zweiten Schritt erfolgen anschließend eine Konsolidierung dieser Ereignisse aus sämtlichen aktiven Überwachungswerkzeugen und eine darauf durchgeführte Erkennung von vorab definierten Ereignismustern, wodurch sich die aufgetretenen Probleme genau eingrenzen lassen. Solchen Ereignismustern (komplexen Events) lassen sich nun Tuning-Pläne zuordnen, die beim Erkennen der Muster automatisch angestoßen werden. Tuning-Pläne bestehen aus einzelnen, mit Hilfe von Kontrollstrukturen miteinander verknüpften Tuning-Aktivitäten. Die Tuning-Aktivitäten bieten eine semantisch angereicherte Sicht auf die Effektor-Schnittstellen einzelner Objekte an und ermöglichen somit das Anlegen, Löschen bzw. Konfigurieren dieser Objekte.

Zur Veranschaulichung dieses Zusammenhangs wurde in ***Abbildung 4.9*** ein Bufferpool-Tuning-spezifisches Beispiel gewählt. Auf dem Sensor `Bufferpool Hitratio (BPHR)` des Datenbankobjekts *Bufferpool* wird im vorliegenden Beispiel ein atomarer Event `BPHR<`70% definiert. Dieser atomare Event wird in einem darauf aufbauenden komplexen Event verarbeitet, welcher generiert wird, sobald die BPHR fünf mal innerhalb von 10 Minuten unter 70% fällt. Diesem komplexen Ereignis ist der entsprechende Tuning-Plan *Bufferpool-Tuning* zugeordnet, der beim Auftreten des Events angestoßen wird[19]. Ein Bestandteil dieses Tuning-Plans ist der Tuning-Schritt zum Erhöhen der Bufferpool-Größe um 1.000 Seiten, welcher eine Instanziierung einer entsprechenden Tuning-Aktivität darstellt. Diese

[19] Auf die Ausführung von Tuning-Plänen beim Auftreten von entsprechenden Events können weitere Faktoren wie beispielsweise die von Administratoren vorgegebenen Richtlinien, gesammelten Statistiken oder weitere Metadaten einen Einfluss haben (vgl. *Kapitel 6*).

Tuning-Aktivität setzt die Veränderung der Bufferpool-Größe mittels der vom Datenbankobjekt *Bufferpool* bereitgestellten Effektorschnittstelle `Bufferpool-Größe` um.

Aufgrund ihrer Einfachheit, hohen Benutzerfreundlichkeit und dennoch ausreichenden Mächtigkeit und nicht zuletzt aufgrund ihres modularen Aufbau ist die eingeführte Tuning-Plan-Sprache nicht nur hervorragend zur Modellierung und automatischen Ausführung von Datenbank-Administrations- bzw. Datenbank-Tuning-Vorgehensweisen geeignet, sie ermöglicht zudem eine Austauschbarkeit der erstellten Tuning-Pläne unter den Administratoren. Die entsprechenden Konzepte zur Unterstützung eines effizienten Austauschs von Tuning-Plänen in einer Community werden in *Kapitel 7* vorgestellt.

Die Tuning-Pläne beschränken sich dabei nicht nur auf die automatische Auflösung von Problemen. Ihre Anwendungsbereiche sind sehr vielseitig und werden nachfolgend exemplarisch aufgeführt:

Reaktive Problembehebung. Bei einem erkannten Problem kann mittels eines entsprechenden Tuning-Plans eine automatische Behebung des Problems umgesetzt werden. Bei Bedarf kann dabei vor Ausführung bestimmter Aktionen die Bestätigung eines Administrators eingeholt werden.

Unterstützung der Administratoren. Ist eine automatische Problemlösung nicht möglich oder nicht erwünscht, so lassen sich mittels Tuning-Plänen die Ursachen für das Problem identifizieren und die Verantwortlichen darüber informieren. Dabei können sowohl weitere, das Problem beschreibende Informationen als auch Lösungsvorschläge an den Verantwortlichen übermittelt werden.

Qualitätssicherung. Durch eine periodische Ausführung von Tuning-Plänen lassen sich bestimmte Richtlinien, wie beispielsweise die Verwendung von bestimmten Namenskonventionen bei den Datenbankobjekten oder ähnliches, umgesetzt werden[20].

Umsetzung von Problemvermeidungsstrategien. Eine proaktive Problemvermeidung kann beispielsweise durch eine periodische Abarbeitung von Checklisten, die typische Fehler in einem System enthalten, erreicht werden. Ein Beispiel hierfür bildet die Liste mit den 10 weit verbreiteten Fehlern, die beim Konfigurieren bzw. Benutzen eines *Oracle*-Datenbank-Management-Systems gemacht werden („*Top Ten Mistakes Found in Oracle Systems*“ [Ora08]).

Workload-abhängige Systemkonfiguration. Durch eine Einbeziehung der Information über die aktuell anliegende Workload kann mit den entsprechenden Tuning-Plänen ein Workload-abhängiges Umkonfigurieren des Systems umgesetzt werden, was zu einer Steigerung des Durchsatzes und Vorbeugung von potenziellen Problemen beiträgt.

Wissensweitergabe. Auch ohne eine entsprechende Ausführungskomponente lassen sich Tuning-Pläne zur Kapselung und Weitergabe des Wissens über das Datenbank- bzw. IT-System-Tuning einsetzen.

Zur Modellierung von Tuning-Plänen ist die Wahl eines visuellen Ansatzes sinnvoll und erhöht nicht nur die Benutzerfreundlichkeit, sondern reduziert auch die Fehleranfälligkeit

[20] Hierbei wird deutlich, dass sich die Anwendungsgebiete der Tuning-Pläne nicht auf das eigentliche Tuning von datenbankbasierten Software-Stacks beschränken.

bei der Tuning-Plan-Entwicklung. Bei der Umsetzung einer Auswahl von Tuning-Plänen mit Hilfe eines herkömmlichen Workflow-Systems sind einerseits die hohe Komplexität eines solchen Systems und andererseits der damit verbundene hohe Ressourcenverbrauch zur Laufzeit negativ ins Gewicht gefallen, so dass die Entwicklung eines speziell auf das Datenbank-Tuning zugeschnittenen, leichtgewichtigen Systems zur Modellierung und Ausführung von Tuning-Plänen als unabdingbar erscheint. Bei der Entwicklung dieses Werkzeugs wurde weitgehend auf die Verwendung und Anpassung des Frameworks zur Erstellung von dynamischen Internetanwendungen namens *IBM WebSphere sMash (WsM)* zurückgegriffen, welches unter anderem einen webbasierten Editor zur Erstellung von auf Internet-Anwendungen zugeschnittenen Workflows bereitstellt. Der Einsatz eines solchen Modellierungswerkzeugs könnte beispielsweise auch eine Trennung zwischen der konzeptuellen und der technischen Tuning-Plan-Modellierung ermöglichen, ähnlich wie das durch die Abstraktionsstufen bei der GLIF-Modellierung umgesetzt wird (*Abschnitt 4.1.2*). Administratoren könnten damit die Tuning-Pläne konzeptuell entwickeln, ohne dabei die Datenquellen genau zu spezifizieren, Entscheidungskriterien zur Steuerung des Kontrollflusses anzugeben etc. Im nächsten Schritt könnte eine andere Person die technische Umsetzung dieser Tuning-Pläne übernehmen.

Die Mächtigkeit der durch WsM bereitgestellten Kontrollstrukturen ist für die Erstellung von Tuning-Plänen angemessen. Obwohl sich mit Hilfe dieser Kontrollstrukturen eigene Feedback-Kontrollschleifen innerhalb von Tuning-Plänen umsetzen lassen (ein Tuning-Plan könnte sich dem Tuning eines bestimmten Datenbankobjekts widmen und das Objekt kontinuierlich überwachen und optimieren), ist bei unserem Ansatz davon grundsätzlich abzuraten, da dieses Vorgehen einen erhöhten Koordinationsaufwand und eine höhere Fehleranfälligkeit verursachen würde. Die Tuning-Pläne sollten iterativ, durch die Events initiiert, angestoßen werden und idealerweise dem allseits bekannten Tuning-Prinzip folgend jeweils nur eine Änderung am System zu einem Zeitpunkt vornehmen, um die Wahrscheinlichkeit des Auftretens von Seiteneffekten zu reduzieren.

Da es jedoch naheliegenderweise oft nicht ausreichend ist, auf aufgetretene Probleme „blind“ mit entsprechenden Tuning-Plänen zu reagieren [RWRA08], lassen sich die von Administratoren vorgegebenen Richtlinien sowie die im laufenden Betrieb anfallenden Metadaten, wie beispielsweise Statistiken über Tuning-Plan-Ausführungen und deren Wirkung im Hinblick auf Systemzustandsverbesserungen bzw. -verschlechterungen, zur Unterstützung einer intelligenten Auswahl von Tuning-Plänen verwenden.

Nicht nur die Auswahl, sondern auch Ausführung von Tuning-Plänen lässt sich durch die Auswertung und Berücksichtigung von Tuning-Richtlinien bzw. Metadaten beeinflussen und optimieren. So weicht beispielsweise der in *Abschnitt 6.3* vorgestellte Ansatz einer intelligenten Hypothesenbaum-basierten Tuning-Plan-Ausführung von dem oben genannten Prinzip ab und ermöglicht die Erstellung und Ausführung von komplexen Tuning-Plänen, die sich auf das Tuning bestimmter Problembereiche konzentrieren und nach dem Anstoßen durch einen entsprechenden Event mit Hilfe von Backtracking-Mechanismen verschiedene Lösungswege ausprobieren, um das erkannte Problem zu lösen.

Kapitel 5

Probleminitiierte und -vorbeugende Ausführung von Tuning-Plänen

IT-Systeme mit nach [IBM06a] umgesetzten autonomen Fähigkeiten verwalten ihre Ressourcen mit Hilfe so genannter *Autonomic Manager*. Typischerweise konzentriert sich ein *Autonomic Manager* dabei auf eine konkrete Problemklasse und implementiert dazu eine MAPE-Schleife. Hierbei handelt es sich um ein bekanntes Konzept aus der Kontrolltheorie. Eine rückgekoppelte Schleife durchläuft kontinuierlich vier aufeinander folgende Phasen (vgl. *Abschnitt 2.2.4*):

- Sammeln von Informationen über den Zustand der kontrollierten Ressource (*Monitor*),
- Analyse der Zustandsänderungen / Problemanalyse (*Analyze*),
- Planung von Aktionen zur Optimierung des Ressourcen-Zustandes (*Plan*) sowie
- deren Ausführung (*Execute*).

Die einzelnen MAPE-Phasen tauschen von außen bereitgestelltes oder generiertes, akkumuliertes Wissen und Daten aus, um die Funktionalität des MAPE-Regelkreises zu gewährleisten. Dabei handelt es sich unter anderem um Wissen über Systemanforderungen, Topologie-Informationen sowie über die Abhängigkeiten zwischen einzelnen Komponenten (*Abschnitt 5.1*).

Autonomic Tuning Expert – die in *Abschnitt 5.2* vorgestellte Architektur zum Tunen von datenbankbasierten Infrastukturen – ist im Rahmen eines Kooperationsprojekts der *Friedrich-Schiller-Universität Jena* mit der *IBM Deutschland Research & Development GmbH* entstanden. Sie basiert primär auf dem MAPE-Paradigma und ergänzt diese um einige weitere Komponenten wie den *Workload Classifier*. Zur Evaluation der Funktionsweise des *Autonomic Tuning Expert* wurde ein Satz von Tuning-Plänen (*Abschnitt 5.3*) entwickelt und in das System integriert. Anschließend wurde das System einer wechselnden Workload ausgesetzt. Die Effektivität des ATE-Tuning konnte durch die Überwachung und Auswertung von Performance-relevanten Metriken evaluiert werden. Eine kurze Vorstellung und Bewertung der Evaluationsergebnisse findet sich in *Abschnitt 5.4*. In *Abschnitt 5.5* werden anschließend die in diesem Kapitel diskutierten Ansätze zusammengefasst.

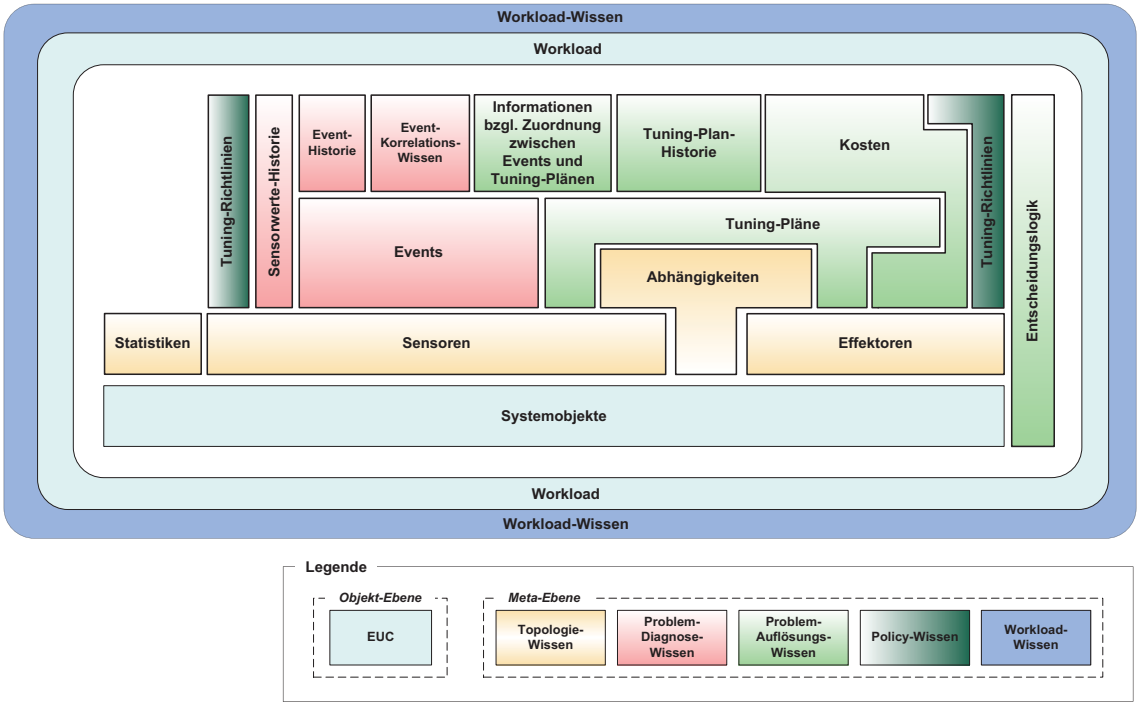


Abbildung 5.1: Wissensmodell zur Unterstützung des best-practices-orientierten Datenbank-Tuning (in Anlehnung an [RW07])

5.1 Unterstützung des Tuning-Prozesses durch das Tuning-Wissen

Zur Unterstützung der Funktionsweise einzelner MAPE-Phasen müssen entsprechende Informationen bereitgestellt sowie eine Möglichkeit geschaffen werden, in den einzelnen Phasen gesammeltes oder generiertes Wissen an andere Komponenten weiterzugeben. Das nachfolgend beschriebene und in ***Abbildung 5.1*** veranschaulichte Wissensmodell adressiert diese Anforderungen und umfasst sämtliche Informationen, die zum autonomen Datenbank-Tuning notwendig sind. Die einzelnen Schichten des Wissensmodells bauen, wie in *Abbildung 5.1* angedeutet, aufeinander auf. Während generisches und wiederverwendbares Wissen in unteren Schichten des Wissensmodells angesiedelt ist, steigt mit jeder nächst höheren Schicht die Spezialisierung und sinkt die Wiederverwendbarkeit der Wissensschichten. Auf gewisse Aspekte der Wiederverwendbarkeit wird in *Kapitel 7* eingegangen. Die nachfolgenden Ausführungen basieren größtenteils auf [RW07, WR07a, WR09].

5.1.1 Überwachtes System

Die Grundlage für das Tuning und somit auch für das nachfolgend beschriebene Wissensmodell zur Unterstützung des Tuning bietet das überwachte und zu tunende System, welches hier als *Environment under Control (EUC)* bezeichnet wird. Das EUC umfasst

somit einerseits sämtliche Objekte des zu tunenden Systems sowie die am System anliegende Workload, da diese den Tuning-Prozess stark beeinflusst. Alle weiteren Schichten des Wissensmodells bauen auf der EUC-Schicht auf und konzentrieren sich auf einzelne Aspekte des überwachten Systems.

Wie der *Abbildung 5.1* zu entnehmen ist, spielt die Workload eine entscheidende Rolle im Tuning-Prozess und somit auch im hier vorgestellten Wissensmodell. Eine sich verändernde Workload ist einer der Hauptgründe für ein entsprechendes, fortlaufendes System-Tuning. Die Workload beeinflusst nicht nur das Verhalten des überwachten Systems entscheidend, sondern hat auch einen tiefgreifenden Einfluss auf die Instanziierung des Wissensmodells durch das Beeinflussen sämtlicher Modellschichten wie beispielsweise des *Topologie-*, *Problem-Diagnose-*, *Problem-Auflösungs-* bzw. *Policy-Wissens*.

5.1.2 Workload-Wissen

Das *Workload-Wissen* umfasst Informationen über Workload-Typen wie beispielsweise OLTP bzw. OLAP ihre Auswirkungen auf das überwachte System sowie Workload-Erkennungsregeln. Für die einzelnen Workload-Typen können unterschiedliche Performance-Ziele, Anforderungen an die Ressourcen, Abhängigkeiten zwischen den Systemobjekten, Tuning-Pläne, Events bzw. die zu überwachenden Sensoren definiert werden. Da sämtliche Metadaten pro Workload-Typ erfasst werden müssen, empfiehlt sich, die Anzahl der zu erkennenden Workload-Typen gering zu halten (vgl. *Abschnitt 5.2.3*).

5.1.3 Topologie-Wissen

Das *Topologie-Wissen* umfasst statische und dynamische Informationen über die Hierarchie der Systemkomponenten, ihren Status, ihre Konfiguration, ihr Verhalten sowie ihre Abhängigkeiten untereinander. Es setzt sich aus den entsprechenden Schichten zusammen, die die Statistiken, die Sensoren, die Effektoren sowie die Abhängigkeiten unter den einzelnen Objekten beschreiben. Das Topologie-Wissen bieten somit die Grundlage für die Umsetzung sämtlicher autonomer Problem-Diagnose- bzw. -Auflösungs-Abläufe.

5.1.3.1 Statistiken

Statistische Informationen über den Inhalt der DBMS-Objekte wie beispielsweise Tabellen, Sichten, Indizes oder materialisierte Sichten können unter anderem zur Unterstützung der Problem-Diagnose und -Auflösung verwendet werden. Zu solchen Informationen gehört beispielsweise die Anzahl der Zeilen einer Tabelle, die durchschnittliche Zeilenlänge, der Speicherplatzverbrauch einer Tabelle bzw. eines Index, die Werteverteilung etc. Heutige DBMS bieten diese Daten in entsprechenden Katalogtabellen bzw. -sichten an.

5.1.3.2 Sensoren

Grundsätzlich bietet jede Ressource eine Auswahl an charakterisierenden Sensorelementen an, die den aktuellen Zustand, Ressourcenverbrauch, Performance und ähnliches beschreiben. Sammelt man diese Daten über einen längeren Zeitraum, so lassen sich daraus die Werteverläufe und Tendenzen ablesen. Es kann zwischen zwei Arten von Sensorelementen unterschieden werden: *Statische Sensorelemente* wie beispielsweise die physische Speichergröße einer Datenbank sind fix während der Lebensdauer einer Ressource. Zwar lassen sich diese unter Umständen modifizieren, jedoch sind sie typischerweise stabil und ändern sich nie oder selten. *Dynamische Sensorelemente* dagegen unterliegen ständigen Veränderungen. Für eine Datenbank sind das beispielsweise die Sensoren zur Repräsentation des Gesundheitszustands der Datenbank, die Anzahl der verbundenen Agenten etc.

Datenbank-Management-Systeme bieten typischerweise Mechanismen und Werkzeuge zur Sammlung, Speicherung und Analyse von Sensorwerten an. Im Fall von DB2 handelt es sich bei solchen Werkzeugen beispielsweise um *IBM DB2 Performance Expert* (*Abschnitt 3.3.2.2*) bzw. DB2-interne Mechanismen wie *DB2 System Monitor* (*Abschnitt 3.3.2.1*).

Die Erfassung von Sensordaten ist von großer Bedeutung für die Problem-Diagnose und die Trend-Erkennung. Aufgrund des potentiell hohen Ressourcenverbrauchs bei der Sammlung dieser Daten ist ein Kompromiss zu finden und von einer zu häufigen Datensammlung abzusehen.

5.1.3.3 Effektoren

Neben den Sensoren bieten die meisten Ressourcen jeweils einen Satz von Effektoren an. Mit ihrer Hilfe lässt sich der Zustand bzw. das Verhalten einzelner Ressourcen verändern. Die DBMS bieten diverse Einstellmöglichkeiten. Eine Auswahl der wichtigsten Einstellmöglichkeiten in *IBM DB2* findet sich in *Abschnitt 4.3*. Dazu gehören beispielsweise SQL-Anweisungen, Befehlzeilenprozessorbefehle, Systembefehle bzw. Betriebssystembefehle.

5.1.3.4 Abhängigkeiten

Gründe für eine schlechte Performance können sehr vielfältig und oft nur schwer erkennbar sein. Ihre Identifikation wird zusätzlich durch Abhängigkeiten zwischen den Systemkomponenten erschwert. So kann es zu Engpässen aufgrund von einer falschen Konfiguration einzelner Ressourcen oder aber auch duch das Zusammenspiel mehrerer Ressourcen und ihrer Konfigurationen kommen. Die Abhängigkeiten zwischen den Ressourcen sind dabei sehr komplex und kaum manuell quantifizierbar.

Eine Änderung eines der Effektoren eines Systemobjekts kann unter Umständen einen unvorhersehbaren Einfluss auf andere Systemobjekte ausüben und gar neue Probleme verursachen, anstatt das vorliegende Problem zu lösen. Andererseits kann es vorkommen, dass Änderungen bestimmter Effektoren erst in Kombination mit Änderungen anderer Effektoren wirksam werden.

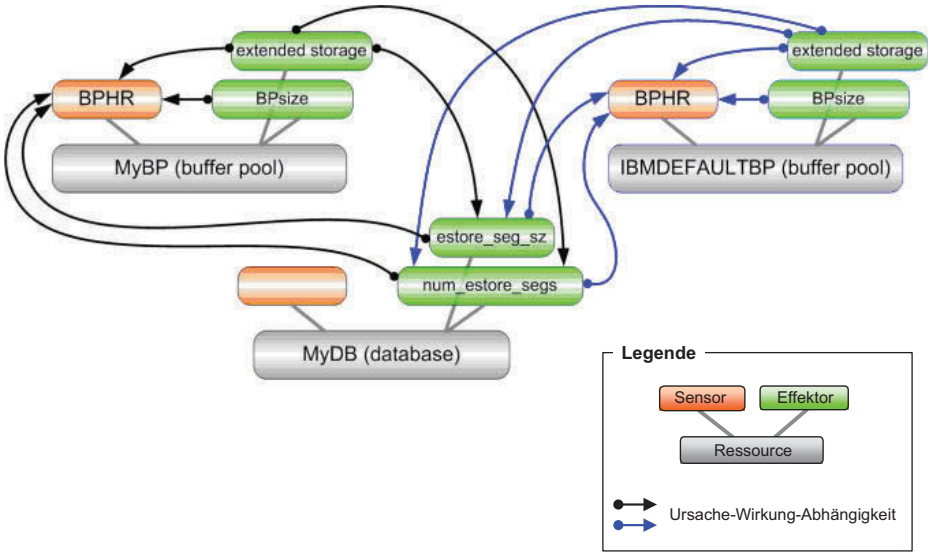


Abbildung 5.2: Beispiel für Ursache-Wirkung-Abhängigkeiten [WR09]

Zur Unterstützung der Planungsphase ist die Erstellung eines Ressourcenabhängigkeitsmodells unabdingbar. Das in [Gan06] ursprünglich eingeführte und in darauf aufbauenden Arbeiten [Exn07, RW07, WR09] weiterentwickelte Modell erfasst die Abhängigkeiten zwischen den Sensoren und Effektoren eines Systems und quantifiziert diese unter Berücksichtigung der anliegenden Workload. Dabei werden drei Arten von Abhängigkeiten zwischen den Ressourcen sowie ihren Sensoren und Effektoren unterschieden:

Hierarchien der Systemobjekte. Die meisten Datenbankobjekte lassen sich hierarchisch anordnen. Eine DB2-Instanz besteht aus mehreren Datenbanken, jede Datenbank umfasst mehrere Tablespaces, die wiederum zum Gruppieren von Tabellen verwendet werden. Das Wissen über die hierarchische Einordnung von Ressourcen findet unter anderem bei ihrer Identifikation Anwendung. So hat beispielsweise jede DB2-Datenbank einen Bufferpool namens *IBMDEFAULTBP1*. Dieser kann somit erst durch die Angabe des vollständigen Hierarchiepfades eindeutig identifiziert werden (z. B. *instanzname.datenbankname.IBMDEFAULTBP1*).

Situationsbezogene Informationen. Zu den einzelnen Sensorwerten können weiterhin Informationen über die Nebenbedingungen und Umstände, die zu den jeweiligen Sensorwerten geführt haben, verwaltet werden. Zu solchen Informationen zählen beispielsweise: welche Anwendungen und auf welchen Tablespaces zu bestimmten Zeitpunkten aktiv waren und welche Operationen sie dabei ausgeführt haben, welche Datenbanktabellen lesend bzw. schreibend zugegriffen wurden oder welche Konfigurationseinstellungen zu diesem Zeitpunkt aktiv waren. Die Erfassung solcher Informationen kann mittels multidimensionaler Strukturen erfolgen [Wie05]. Eine regelmäßige Sammlung und Speicherung dieser Informationen kann jedoch das überwachte System stark beeinträchtigen und einen hohen Speicherverbrauch verursachen.

Ursache-Wirkung-Abhängigkeiten. Diese Abhängigkeitsart beschreibt die Ursache-Wirkung-Zusammenhänge zwischen den einzelnen Ressourcen. Dabei wird nicht nur die

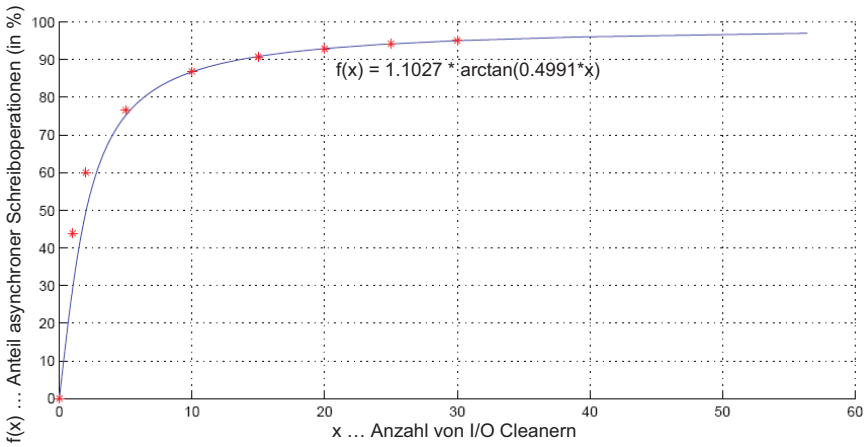


Abbildung 5.3: Performance-Funktion für die Abhängigkeit zwischen der Anzahl von I/O-Cleanern und dem Anteil asynchroner Schreiboperationen [RW07]

Existenz der Abhängigkeiten zwischen Effektoren und Sensoren bzw. Effektoren und Effektoren, sondern auch ihr Ausmaß erfasst.

Effektor-Sensor-Abhängigkeiten. Effektor-Sensor-Abhängigkeiten beschreiben das Ausmaß der Sensorwertveränderungen, die durch Änderungen der Werte anderer Sensoren verursacht wurden. Dabei erfolgt eine Berücksichtigung der Nebenbedingungen wie die Charakteristika der Systemumgebung bzw. der anliegenden Workload. Die Erfassung dieser Abhängigkeiten ermöglicht unter anderem das Beantworten von Fragen wie beispielsweise folgende[21]:

- Welchen Wert wird der Sensor s annehmen, wenn der Effektor e_1 auf den Wert v_1 und der Effektor e_2 auf den Wert v_2 gesetzt werden?
- Welche Effektoren müssen geändert werden und zu welchen Werten, um den Sensor s auf den Wert v zu bringen?

In ***Abbildung 5.2*** findet sich ein Beispiel für eine Effektor-Sensor-Abhängigkeit zwischen dem Effektor `BPSize` zur Veränderung der Bufferpool-Größe und dem Sensor `BPHR` zum Auslesen der Bufferpool Hitratio. Zur Bestimmung des Ausmaßes der Abhängigkeiten bietet sich beispielsweise die nicht-lineare Regression an, die eine Approximierung von Performance-Funktionen ermöglicht [Exn07]. Eine exemplarische Veranschaulichung einer mittels der nicht-linearen Regression ermittelten Performance-Funktion zur Beschreibung der Abhängigkeit zwischen dem Parameter zur Festlegung der Anzahl von Agenten (*I/O Cleaners*) zum Herausschreiben geänderter Bufferpool-Seiten auf die Platte (`num_iocleaners`) und dem berechneten Sensor zur Angabe des Anteils der asynchron durchgeführten Schreiboperationen gegenüber den synchron durchgeführten Schreiboperationen (`percentage_of_async_writes`) findet sich in ***Abbildung 5.3***. Für weitere Ausführungen zum Verfahren sowie den identifizierten Performance-Funktionen sei auf [RW07] sowie [Exn07] verwiesen.

[21] Es ist jedoch zu berücksichtigen, dass eine quantitative Erfassung solcher Abhängigkeiten bzw. die Beantwortung solcher Fragen bei komplexen Systemen nicht immer möglich ist.

Effektor-Effektor-Abhängigkeiten. Effektor-Effektor-Abhängigkeiten erfassen die Zusammenhänge, die unter Berücksichtigung bestimmter Nebenbedingungen zwischen den einzelnen Effektoren bestehen. Zwei mögliche Typen der Effektor-Effektor-Abhängigkeiten sind dabei im Folgenden angegeben:

- Die Werte des Effektors e_1 hängen von den Werten des Effektors e_2 ab. Ein Beispiel hierfür bieten der Parameter `max_querydegree`, welcher den maximalen Grad des partitionsinternen Parallelismus angibt, der für SQL-Anweisungen verwendet wird, sowie der Parameter `intra_parallel`, der angibt, dass der Datenbankmanager partitionsinternen Parallelismus verwenden kann [IBM06g]. Durch den Parameter `max_querydegree` führt jede SQL-Anweisung nicht mehr als die in diesem Parameter festgelegte Anzahl paralleler Operationen innerhalb einer Partition aus. Damit dieser Parameter jedoch berücksichtigt wird, muss zuvor der Konfigurationsparameter `intra_parallel` auf den Wert „yes“ gesetzt werden.
- Zur Erreichung der maximalen Wirkung der Parameterwertänderungen sind einige Parameter oft in Kombination zu ändern. So sollten beispielsweise die Parameter `maxlocks` und `locklist` vorzugsweise gemeinsam geändert werden.

In *Abbildung 5.2* werden unter anderem die Abhängigkeiten zwischen den Konfigurationsparametern zur Angabe der Segmentgröße (`estore_seg_sz`) sowie der Segmentanzahl (`num_estore_segs`) für erweiterten Speicher veranschaulicht, die nur dann einen Einfluss auf die Bufferpool Hitratio haben, wenn die Verwendung des erweiterten Speichers mittels des Effektors `extended storage` aktiviert wurde.

5.1.4 Problem-Diagnose-Wissen

Das Problem-Diagnose-Wissen adressiert bewährte Vorgehensweisen („*best-practices*“) der Administratoren beim Erkennen und Interpretieren von Problemen sowie beim Identifizieren ihrer Ursachen durch die Auswertung von aktuellen Sensorwerten bzw. der Sensorwerte-Historie. Weiterhin finden bei der Problem-Diagnose Informationen bezüglich der Ereigniskorrelationen sowie die Historie der aufgetretenen Events Anwendung.

Zur Ermöglichung einer Trennung der Überwachungs- und der Problem-Auflösungslogik enthält das Problem-Diagnose-Wissen keine Informationen zur Problemauflösung. Diese sind stattdessen im entsprechenden Problem-Auflösungs-Teil des Wissensmodells enthalten.

5.1.5 Problem-Auflösungs-Wissen

Beim Auftreten bestimmter Events werden entsprechende, diesen Events zugeordnete Tuning-Pläne angestoßen. Dies sind bewährte Tuning-Maßnahmen, die sich aus einzelnen Tuning-spezifischen Basisaktivitäten zusammensetzen. Mit Hilfe von Tuning-Plänen lassen sich also Reaktionen auf erkannte Problemsituationen umsetzen bzw. bestimmte Problemsituationen vorbeugen (*Abschnitt 4.4*). Tuning-Pläne können dabei Sensorwerte auslesen, die vorliegenden Informationen über die Abhängigkeiten zwischen den Sensoren

Schichten des Wissensmodells		MAPE-Phasen			
		M	**A**	**P**	**E**
	Topologie-Wissen				
	Statistiken			+	
	Sensoren	+			
	Effektoren				+
	Abhängigkeiten		+	+	+
	Problem-Diagnose-Wissen				
	Events		+	+	
	Event-Korrelations-Wissen		+		
	Event-Historie		+	+	
	Sensor-Werte-Historie	+	+	+	
	Problem-Auflösungs-Wissen				
	Tuning-Pläne			+	+
	Tuning-Plan-Historie		+	+	+
	Zuordnung zwischen Events und Tuning-Plänen			+	+
	Kosten			+	+
	Entscheidungslogik			+	+
	Policy-Wissen	+	+	+	+
	Workload-Wissen	+	+	+	+

Tabelle 5.1: Abbildung der Schichten des Wissensmodells auf die MAPE-Phasen

bzw. zwischen Sensoren und Effektoren auswerten und diese Daten in die Entscheidungslogik zur Behebung von Problemen durch Veränderungen der Sensorwerte integrieren.

Bei der Ausführung von Tuning-Plänen erfolgt unter anderem auch die Berücksichtigung der bereits in der Vergangenheit ausgeführten Tuning-Pläne sowie der *Effektor-* bzw. *Tuning-Plan-Kosten*. Hierbei handelt es sich um die gelernten bzw. geschätzten Kosten der Änderung einzelner Effektoren. Die Kosten können dabei auf die Anzahl von zur Umsetzung entsprechender Änderungen notwendigen Operationen, auf die entsprechenden I/O-Operationen bzw. die Auswirkungen der Änderungsoperationen auf das überwachte System, wie beispielsweise ein Neustart des Systems zur Aktivierung der Änderungen, zurückgeführt werden. Durch ein Aufsummieren der Kosten einzelner Tuning-Aktivitäten eines Tuning-Plans lassen sich die Gesamtkosten dieses Tuning-Plans ermitteln und beispielsweise bei einer kostenbasierten Tuning-Plan-Auswahl berücksichtigen. So lässt sich beispielsweise beim Vorliegen mehrerer Tuning-Pläne zur Lösung eines bestimmten Problems der Tuning-Plan mit den geringsten Kosten auswählen. Bei einem schwerwiegenden Problem und/oder hoher Dringlichkeit kann es jedoch unter Umständen notwendig sein, den Tuning-Plan mit höchster Effektivität ungeachtet der Kosten auszuwählen.

5.1.6 Policy-Wissen

Der Planungsprozess im Rahmen des autonomen Tuning lässt sich mit Hilfe von *Tuning-Richtlinien* (engl. *Policies*) beeinflussen. So kann mit Hilfe von entsprechenden Sprachkon-

strukten angegeben werden, welche Ressourcen primär überwacht werden sollten und ob bzw. welche Änderungen am System vorgenommen werden dürfen (*Abschnitt 6.2*). Das Policy-Wissen umfasst somit beispielsweise die von den Administratoren vorgegebenen Tuning-Ziele, zulässige Wertebereiche einzelner Sensoren bzw. Effektoren sowie die zur Veränderung einzelner Effektoren akzeptablen Kosten (vgl. *Abschnitt 5.1.5*).

5.1.7 Einfluss des Tuning-Wissens auf einzelne MAPE-Phasen

Das soeben vorgestellte Wissensmodell wurde zur Unterstützung eines MAPE-Schleifen-basierten Datenbank-Tuning-Ansatzes entwickelt. ***Tabelle 5.1*** ordnet die Schichten des Wissensmodells den einzelnen MAPE-Phasen zu, in denen sie verstärkt Anwendung finden bzw. generiert werden.

So erfolgt beispielsweise das Erzeugen der Informationen bezüglich der Historie der Sensorwerte in der Monitoring-Phase, Historie der generierten Events in der *Analyze*-Phase und die Historie der ausgeführten Tuning-Pläne in der *Execute*-Phase. Dieses Wissen findet anschließend unter anderem bei der Planung Anwendung und hilft dabei, Tuning-Pläne zu identifizieren, die den besten Nutzen bei der Problem-Auflösung versprechen. Im Gegensatz zu anderen Schichten des Wissensmodells finden sowohl das Policy- als auch das Workload-Wissen Anwendung in allen MAPE-Phasen und haben daher einen großen Einfluss auf das MAPE-Schleifen-basierte Datenbank-Tuning.

5.2 Architektur des *Autonomic Tuning Expert*

Im vorliegenden Abschnitt erfolgt eine kurze Vorstellung des im Rahmen des „*IBM Center for Advanced Studies (CAS)*“-Projekts entwickelten Architekturvorschlags zum autonomen, best-practices-orientierten Tunen von datenbankbasierten Anwendungen. Den nachfolgenden Ausführungen liegt zu einem großen Teil [RWRA10] zugrunde.

Bei dem klassischen MAPE-Ansatz [IBM06a] erfolgt die Betrachtung der Problembereiche lokal durch die *Autonomic Manager*. Zur bereichsübergreifenden Problem-Erkennung und -Auflösung ist die Einführung von übergeordneten, in einer Hierarchie angeordneten *Autonomic Manager* möglich. Im ATE-Framework, unserem Architekturvorschlag zum autonomen Datenbank-Tuning [WRRA08, RWRA08, WRRA09], weichen wir jedoch von dieser Herangehensweise ab, da wir nach der „goldenen Regel“ des Tuning Änderungen am System ausschließlich sequentiell vornehmen möchten, um somit die Auswirkungen einzelner Aktionen nachvollziehen und gegebenenfalls rückgängig machen zu können. Dadurch reduziert sich der Aufwand zur Erstellungs- und Ausführungszeit, da der Nutzer die benötigte Planungslogik nicht selbst umsetzen muss und systemseitig keine Orchestrierung der einzelnen MAPE-Schleifen notwendig ist.

Die MAPE-Phasen werden im ATE-Framework durch einzelne Komponenten realisiert, die miteinander kooperieren und untereinander Informationen austauschen. Die Architektur integriert Standardprodukte wie *IBM DB2 Performance Expert (PE)*, *IBM WebSphere sMash (WsM)* und *IBM DB2 Intelligent Miner* und basiert auf weit verbreiteten Technologien wie *Generic Log Adapter (GLA)*, *Tivoli Active Correlation Technology (ACT)* und

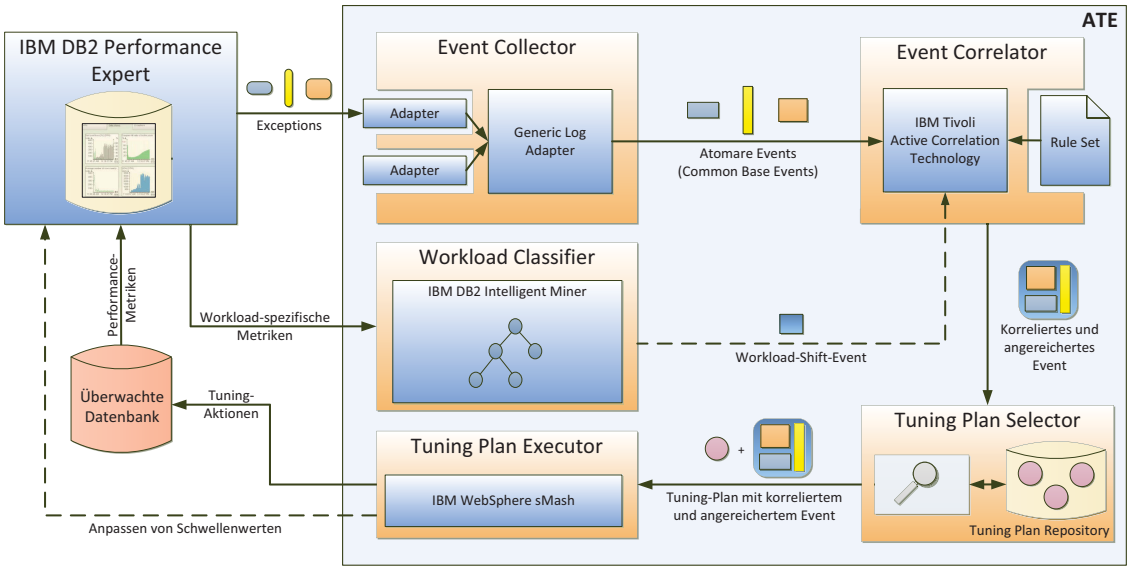


Abbildung 5.4: Architektur des *Autonomic Tuning Expert* [RWRA08]

Common Base Events (CBE). ***Abbildung 5.4*** veranschaulicht dabei das Zusammenspiel aller Komponenten des Frameworks, die im Folgenden näher beschrieben werden.

Die Remote-Fähigkeit des auf standardisierten Schnittstellen basierenden ATE-Frameworks erlaubt die Auslagerung einzelner ATE-Komponenten auf entfernte Systeme, wodurch der zusätzliche Overhead auf dem Produktivsystem minimiert werden kann. Derzeit erscheint eine solche Maßnahme jedoch lediglich für den rechenintensiven *Workload Classifier* (*Abschnitt 5.2.3*) sinnvoll.

5.2.1 Überwachung des zu tunenden Systems

Zur Überwachung des zu tunenden Systems können zahlreiche Überwachungswerkzeuge eingesetzt werden. Typischerweise finden sich in den meisten IT-Infrastrukturen bereits Überwachungskomponenten, die primär von den Systemadministratoren genutzt werden. Unsere Referenzarchitektur wurde daher so konzipiert, dass sie auf beliebigen, bereits vorhandenen Systemüberwachungsprodukten aufsetzen kann und dabei zugleich die Möglichkeit bereitstellt, mehrere Überwachungssysteme zu integrieren und ihre Daten konsolidiert auszuwerten. Somit können die für die Auswertung des Systemzustands notwendigen Daten aus verschiedenen Quellen (Überwachungssysteme wie beispielsweise *IBM Tivoli Monitoring*-Produkte, Log-Dateien oder ähnliches) stammen.

Da wir uns in unserem Ansatz primär auf das Tuning von DB2-basierten Systemen konzentrieren, wurde *IBM Performance Expert (PE)* – ein Werkzeug zur Überwachung von DB2-Datenbanken – als eine der Hauptquellen für Systemüberwachungsdaten in die Gesamtarchitektur aufgenommen. PE bietet neben der Möglichkeit, aktuelle Performance-Metriken anzuzeigen bzw. Trendanalysen auf Basis von in der Vergangenheit gesammelten Performance-Daten durchzuführen (*Performance Warehouse*) auch die Möglichkeit zur Notifikation über Schwellenwert-basierte Ausnahmesituationen. Daher wird PE nicht nur

zur regelmäßigen Erzeugung und Speicherung von Performance-Snapshots herangezogen, sondern ebenfalls zur Definition und Überwachung von Exceptions genutzt. Hierbei wird die Überschreitung von vorher definierten Schwellenwerten, 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, mittels *User Exits* die Informationen über aufgetretene Exceptions für beliebige Konsumenten maschinenlesbar in XML-Form zu externalisieren. Aufgrund seiner Funktionalitäten zur Generierung von Schwellenwert-basierten Ausnahmesituationen ist der PE neben der Monitoring-Phase des MAPE-Zyklus auch der Analyse-Phase zuzuordnen.

Im *Autonomic Tuning Expert* wurde die Möglichkeit der Einbindung verschiedener Quellen der Überwachungsdaten durch die Verwendung des *Generic Log Adapters (GLA)* bei der Umsetzung des *Event Collectors* – der Überwachungskomponente des ATE-Frameworks – bereitgestellt. Der GLA sammelt und verarbeitet kontinuierlich Monitordaten, die durch Nutzung verschiedener Adapter aus beliebigen Quellen (unter anderem Tabellen bzw. Dateien) stammen können. Die aus den verschiedenen Quellen gelieferten und oft in proprietären Formaten vorliegenden Ereignisdaten werden mit Hilfe der entsprechenden Adapter in das standardisierte, XML-basierte „Common Base Event (CBE)“-Format [IBM04a] transformiert und an die für die Korrelation einzelner Ereignisse zuständige Systemkomponente namens *Event Correlator* weitergeleitet. Das CBE-Format ist ein weit verbreitetes Format zum Austausch von Ereignisinformationen zwischen Systemen bzw. Systemkomponenten, welche die Problemidentifikation ermöglichen bzw. autonome Funktionalitäten bereitstellen. Daher werden im CBE-Format entsprechende Elemente zur Beschreibung von Situationen, ihren Ursachen bzw. zur Erfassung weiterer Informationen bereitgestellt.

Es ist zu betonen, dass durch die Entwicklung und Integration entsprechender Adapter die Infrastruktur um weitere, typischerweise in zu tunenden Systemumgebungen bereits vorhandene, Systemüberwachungsprodukte erweitert werden kann. So lassen sich beispielsweise das *DB2 Diagnostic Log (db2diag.log)* oder aber auch diverse Tivoli-Produkte in die ATE-Infrastruktur integrieren.

5.2.2 Analyse von Problemen

Die Analyse-Komponente (*Event Correlator*) bietet Mechanismen zur Problem-initiierten oder auch periodischen Korrelation von CBEs und ermöglicht dadurch die Modellierung komplexer Problemsituationen. Der *Event Correlator* macht Gebrauch von der *Tivoli Active Correlation Technology* zur Korrelation und Verarbeitung von Ereignissen aus unterschiedlichen Quellen (vgl. *Abschnitt 3.4.3*). Die vom *Event Collector* bereitgestellten, eingehenden CBEs werden in einer Queue abgelegt und mit einem vom Benutzer definierten, Workload-abhängigen Satz von Regeln abgeglichen. Diese Regeln definieren, welche Folgen bzw. Muster von atomaren Ereignissen ein komplexes Ereignis darstellen. Dabei ist es unter anderem möglich, Ereignisse zu filtern oder zu zählen. Somit kann der Umgang mit der in einer komplexen Systemumgebung auftretenden Flut von Wertebereichsüberschreitungen vereinfacht werden, indem irrelevante Ereignisse beispielsweise ignoriert werden. Es können also ausschließlich Muster erkannt werden, die in dem aktiven Regelsatz definiert wurden. Dies entspricht dem typischen Vorgehen eines Administrators, der primär beim Auftreten ihn bekannter Problemsituationen diese durch seine Aktionen zu

lösen versucht. Des Weiteren ermöglicht die Korrelation einzelner Ereignisse die Erfassung des Kontexts des zu erkennenden Problems (z. B. andere existente Ereignisse bzw. überschrittene Schwellenwerte). Durch Korrelation atomarer Ereignisse lassen sich also Problemsituationen präziser modellieren.

Mit Hilfe von Korrelationsmustern wie *Collection Pattern* bzw. *Duplicate Pattern* kann das ATE-Framework die Überreaktionen auf bestimmte, mehrfach erkannte Problemsituationen vermeiden. Damit kann das ATE-Framework, ähnlich einem DBA, auf die erste aufzulösende Problemsituation mit entsprechenden Aktionen reagieren und für einen bestimmten Zeitraum (Einpegelungszeit) die nachfolgenden Problemsituationen des gleichen Typs ignorieren. Dadurch lassen sich nach der „goldenen Regel“ des Tuning Änderungen am System sequentiell vornehmen und somit die Auswirkungen einzelner Aktionen nachvollziehen und gegebenenfalls rückgängig machen (siehe oben).

Durch das *Threshold Pattern* kann schließlich das Anstoßen von jeweiligen Aktionen bei einzelnen, seltenen Problemsituationen („Ausreißern“) vermieden werden. Auch ein Administrator ignoriert typischerweise solche Ausreißer, solange sie nicht mehrfach auftreten und widmet sich erst dann ihrer Auflösung.

Wird ein Muster erkannt, so wird eine entsprechende Regelantwort ausgelöst. In unserer Architektur enthält diese Regelantwort das erkannte, diese Regelantwort auslösende Ereignis. Neben den im CBE bereits enthaltenen, vom PE bereitgestellten Informationen, wie beispielsweise Name der Problemdatenbank bzw. Namen der durch das Ereignis betroffenen Datenbankobjekte, wird der CBE um weitere Metainformationen, wie insbesondere die mittels der Workload-Erkennungskomponente (*Abschnitt 5.2.3*) erkannte, am System aktuell anliegende Workload, angereichert. Solche mit Metadaten angereicherten CBEs werden dann an den *Tuning Plan Selector* – die ATE-Komponente für die Umsetzung der Plan-Phase – weitergereicht.

5.2.3 Workload-Erkennung

Der in *Abbildung 5.4* dargestellte *Workload Classifier* lässt sich nicht eindeutig einer MAPE-Phase zuordnen. Seine Aufgabe ist, die aktuell anliegende System-Workload zu erkennen um dadurch Workload-spezifische Problem-Erkennung sowie -Auflösung zur Laufzeit zu ermöglichen und die System-Konfiguration somit stets an den aktuellen Workload-Typ anzupassen. Die Workload-Erkennung sollte parallel zum laufenden Betrieb des jeweiligen Systems möglich sein und in ihren Berechnungen die aktuellen, geschäftskritischen Anfragen nicht merklich behindern. Mögliche Änderungen des Workload-Typs (*Workload-Shifts*) sollten dabei zeitnah und verlässlich erkannt werden, um die betroffenen ATE-Komponenten über die Notwendigkeit einer weiteren Behandlung benachrichtigen zu können. Für ein autonomes System bietet das die Möglichkeit, frühzeitig auf die Veränderungen zu reagieren und so das System optimal auf die wechselnden Anforderungen einzustellen. Im ATE-Framework lassen sich die Workload-Shifts beispielsweise zur dynamischen Anpassung der vom *Exception Processing* des PE verwendeten Schwellenwerte, zur Veränderung der Korrelationsmuster des *Event Correlators* oder zum Steuern der Ausführungslogik einzelner Tuning-Pläne verwenden.

Das in den ATE integrierte, generische Klassifikationsframework bietet folgende Funktionalitäten:

- Definition von klassifikationsrelevanten Workload-Charakteristika und deren Zuordnung zu den zu erstellenden Klassifikationsmodellen,
- Daten-Sammlung und Vorverarbeitung für die Klassifikationsmodelle,
- Erlernen von Klassifikationsmodellen sowie
- Anwenden von Klassifikationsmodellen zur Identifikation von unbekannten Workloads.

Folgende Abschnitte bieten einen Einblick in die Funktionsweisen einzelner Bereiche des Klassifikationsframeworks und basieren hauptsächlich auf [RWRA10].

5.2.3.1 Auswahl klassifikationsrelevanter Metriken und Datenvorverarbeitung

In unserem Ansatz sollen durch den PE 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-Tuning der Aspekt der System- und vor allem Tuning-Unabhängigkeit der Metriken eine große Rolle. [Eln04] beschäftigt sich mit der Identifikation von DB2-Metriken, die von der eigentlichen Hardware-Konfiguration des Systems nicht abhängen (*System-unabhängige Metriken*). Eine Optimierung des Systems im laufenden Betrieb macht jedoch die *Tuning-Unabhängigkeit* der zur Bildung des Klassifikationsmodells verwendeten Metriken unabdingbar. Während beispielsweise die Anzahl der in einer bestimmten Zeit abgesetzten, 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-Anweisungen bezüglich aller abgesetzten Anweisungen lediglich von der Workload ab und kann vom Tuning des Systems nicht beeinflusst werden. Für weiterführende Erläuterungen zur System- bzw. Tuning-Unabhängigkeit sei an dieser Stelle auf [Göb08] verwiesen.

Zur Workload-Klassifikation kann man sich nun auf Tuning-unabhängige Metriken beschränken, auf die sich Änderungen der Systemeinstellungen und das Tuning an sich, also beispielsweise die Änderungen von Datenbank-Konfigurationsparametern oder Anpassungen der Bufferpool-Größe, nicht bzw. nur unwesentlich auswirken. Sollen jedoch Tuning-abhängige Metriken bei der Workload-Klassifikation berücksichtigt werden, so ist ein Vorverarbeitungsschritt unabdingbar. Dazu eliminieren wir den Einfluss von Performance-Tuning oder bestimmten Systemkonfigurationen beispielsweise durch die Division der betrachteten Metrik durch die Anzahl der erfolgreich ausgeführten SQL-Anweisungen (Datenbank-Durchsatz). Die auf solchen normalisierten, System- und Tuning-unabhängigen Metriken basierenden Klassifikationsmodelle können somit auf einem beliebigen, im laufenden Betrieb optimierten Zielsystem zur Klassifikation der Workload eingesetzt werden.

Das Auslesen und Speichern der Werte der DB2-Metriken durch den PE erfolgt im ATE-Framework minütlich, um zeitnah auf Veränderungen der Workload reagieren zu können.

In DB2 stellen Elemente der Typen *Zähler (Counter)* und *Zeit (Time)* seit Monitoring-Start aufsummierte Werte dar und vermischen somit Werte des aktuellen Snapshot-Intervalls mit vorherigen Intervallen. 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 Metriken innerhalb des letzten Zeitintervalls, das heißt 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 (gegebenenfalls anderen Typs) abgeleitet sein können (vgl. *Abschnitt 3.3.2.1*). ATE bestimmt für jeden neuen, vom PE ermittelten und gespeicherten Snapshot die Differenz zum vorherigen und speichert diese in einer entsprechenden Tabelle ab. Dabei erfolgt eine zusätzliche Normierung auf eine konstante Intervalllänge, 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, aber auch zur Visualisierung der Tuning-Ergebnisse verwendet werden.

5.2.3.2 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 5.2.3.1* beschrieben, von Fehlern bereinigt, normiert und anschließend mit der Bezeichnung des jeweiligen Workload-Typs annotiert. Nachdem dies für alle gewünschten Workload-Typen durchgeführt wurde, analysiert ein vom *IBM DB2 Intelligent Miner Modeling* [IBM06c] zur Verfügung gestelltes Entscheidungsbaum-basiertes Klassifikationsverfahren diese Trainingsdatensätze und erstellt ein Profil für ihre Merkmale. Anschließend kann das erstellte Workload-Modell dazu genutzt werden, um unbekannte Workloads zu erkennen.

5.2.3.3 Workload-Erkennung mittels Klassifikation

Die Klassifikation ähnelt der Modell-Erzeugungsphase. Analog zur Modell-Erstellung (*Abschnitt 5.2.3.2*) wird dieselbe Auswahl an Metriken gesammelt und entsprechend vorverarbeitet. Die Datenbank wird nun jedoch mit einer unbekannten Workload belastet. Die Aufgabe der Klassifikation ist es, die Daten mit Hilfe des vorgegebenen Workload-Modells einer bekannten Workload-Klasse zuzuordnen. Dazu werden die vorverarbeiteten Metriken einer *Stored Procedure* des *Intelligent Miner* als Input übergeben. Durch die Prozedur wird eine Sicht erstellt. Diese enthält die Eingabetabelle mit den Metriken sowie zwei neue Spalten, die sowohl das Ergebnis der Klassifikation jedes Tupels als auch die Güte[22] der Klassifikation enthalten.

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 ermitteln kann. Dieses nahe liegende Glätten beseitigt

[22] Als Güte der Workload-Klassifikation verstehen wir das Verhältnis von richtig klassifizierten zu falsch klassifizierten Tupeln.

fehlerhafte Klassifikationen und erhöht die Genauigkeit. Eine gesteigerte Genauigkeit erkauft man sich jedoch mit einer höheren Dauer bis zum Erkennen der aktuell anliegenden Workload [Göb08].

5.2.3.4 Nutzung des Workload-Wissens im ATE

Wird eine Änderung des Workload-Typs erkannt, wird ein Workload-Shift-Event generiert und über den *Event Correlator* in den MAPE-Kreislauf injiziert (*Abbildung 5.4*). Das Wissen über die aktuelle Workload verbreitet sich somit an alle Komponenten und stellt sicher, dass diese nun präziser auf neue Umstände reagieren können. So werden beispielsweise folgende Workload-abhängige Anpassungen des ATE-Verhaltens ermöglicht [RAWR08]:

Workload-abhängige Initialkonfiguration des Systems. In Reaktion auf Workload-Shift-Events lassen sich beliebige Tuning-Pläne anstoßen. So ist es beispielsweise denkbar, die Konfiguration der Datenbank oder des DBMS einmalig für die erkannte Workload proaktiv zu optimieren.

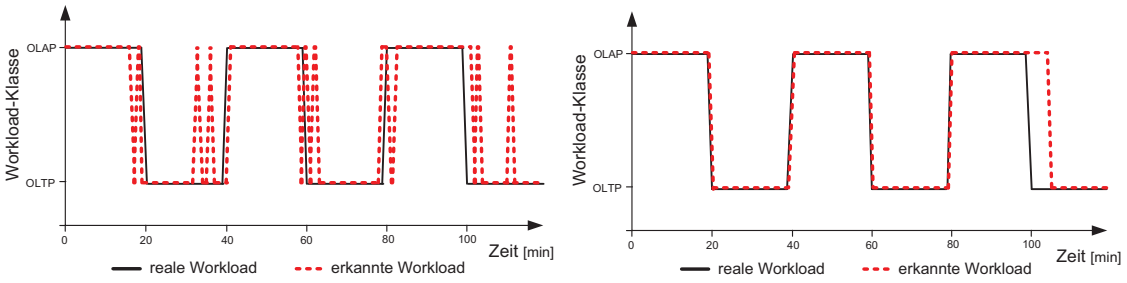
Modifikation der ATE-Konfiguration. Im Rahmen von *Meta-Tuning-Plänen* können Veränderungen an der Konfiguration des ATE vorgenommen werden. So lassen sich beispielsweise die Schwellenwerte des PE und damit die atomaren Ereignisse anpassen (vgl. *Abschnitt 3.3.2.2*). Weiterhin können in Abhängigkeit von der anliegenden Workload auch die Korrelationsmuster des *Event Correlators* verändert werden.

Steuerung der Tuning-Plan-Auswahl. Workload-Information kann auch zur Steuerung der Tuning-Plan-Auswahl durch den *Tuning Plan Selector* eingesetzt werden. So lassen sich beim gleichen Event in Abhängigkeit von der anliegenden Workload verschiedene Tuning-Pläne zur Ausführung auswählen.

Steuerung der Ausführungslogik einzelner Tuning-Pläne. Die Information über die aktuelle Workload kann von Tuning-Plänen dazu genutzt werden, um ihre Ausführung durch die Parametrisierung mit der Workload als Input-Parameter zu verändern (*Abschnitt 5.3*).

5.2.3.5 Übersicht und Bewertung erstellter Workload-Modelle

Für das Training der Workload-Modelle und das anschließende Testen wurden im Rahmen unserer Forschungsarbeiten TPC-C- bzw. TPC-H-angelehnte Workloads [TPC07, TPC08] mit einem eigens entwickelten Java-basierten Workload-Generator erzeugt. 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.



(a) Qualitätstest ohne Glättung (b) Qualitätstest mit Glättung

Abbildung 5.5: Qualitätsuntersuchung des Workload-Modells [Göb08]

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]. Zudem ist es dem Entscheidungsbaum-Algorithmus möglich, mehr als zwei Workload-Klassen vorauszusagen, so dass man weitere Workload-Modelle, die gegebenenfalls neben den reinen OLTP- und OLAP-Klassen auch Mischformen unterscheiden, ohne größeren Aufwand in das System einbringen kann.

Für die Erstellung des Workload-Modells wurden dem *Intelligent Miner Modeling* etwa 1.500 Trainingsdatensätze zur Verfügung gestellt. Die Trainingsdaten stammen zu annähernd gleichen Teilen aus zwei Workload-Klassen und bestehen aus minütlich erstellten Metrikwerten des PE.

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 (***Abbildung 5.5***). Durch die beim ersten Lauf häufigen Ausreißer (*Abbildung 5.5a*) beträgt die Güte des Modells nur 88%. Durch eine Glättung (*Abbildung 5.5b*), die wir durch 5-minütiges Sammeln von Klassifikationsergebnissen und eine anschließende Mehrheitsentscheidung bezüglich der Workload-Klasse des gesamten 5-minütigen Intervalls erzielten, konnten wir die Güte des Modells auf 96% erhöhen. Für weiterführende Details bezüglich der erstellten Workload-Modelle bzw. bezüglich weiterer Qualitätsuntersuchungen sei auf [Göb08] verwiesen.

Generell lässt sich sagen, dass sich die Glättung der Ergebnisse bei allen ermittelten Classifiern 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 unentbehrlich.

Auch in Tests mit parallel durchgeführtem Performance-Tuning durch den ATE hat sich der erstellte Classifier als sehr robust und zuverlässig erwiesen – mit nur einer geringen zeitlichen Verzögerung wurde die anliegende Workload richtig erkannt [Göb08].

5.2.4 Auswahl von Tuning-Plänen zur Laufzeit

Die Planungskomponente des ATE namens *Tuning Plan Selector* hat die Auswahl, Priorisierung und Verknüpfung vordefinierter, auf die Lösung eines bestimmten Problems bzw. einer Problemklasse zugeschnittener Tuning-Praktiken (Tuning-Pläne) zur Aufgabe. Hierfür werden im Vorfeld Tuning-Pläne und deren Zuordnungen zu den durch komplexe Ereignisse repräsentierten Problemsituationen durch DBA definiert und im *Tuning Plan Repository* gespeichert. Zur Laufzeit werden anhand von erkannten Problemen entsprechende Tuning-Pläne aus dem Repository selektiert und an den *Tuning Plan Executor* zur Ausführung weitergeleitet.

Bei der Auswahl von Tuning-Plänen spielen dabei unter anderem insbesondere folgende Aspekte eine wichtige Rolle:

Semantik der Tuning-Pläne. Von großer Bedeutung bei der Auswahl von Tuning-Plänen sind Informationen bezüglich der Tuning-Plan-Semantik. Zu solchen Semantik erfassenden Informationen gehören beispielsweise Annotationen einzelner Tuning-Schritte bzw. ganzer Tuning-Pläne durch entsprechende Taxonomie- bzw. Ontologie-Elemente (vgl. *Abschnitt 4.4.1.3* sowie *Abschnitt 7.1.6*).

Vorgaben der Administratoren. Mit Hilfe der in *Abschnitt 6.2* vorgestellten Tuning-Richtlinien (Policies) lassen sich durch die Administratoren Vorgaben bezüglich der Tuning-Ziele bzw. der beim Tuning zu berücksichtigenden Nebenbedingungen vornehmen. Zur Laufzeit werden diese Administratorvorgaben durch ATE ausgewertet und von den entsprechenden Komponenten berücksichtigt, wodurch unter anderem die Auswahl der Tuning-Pläne sowie die Ausführung einzelner Tuning-Schritte beeinflusst werden.

Kontext der erkannten Problemsituation. Das gleiche durch Events repräsentierte Problem kann beispielsweise in Abhängigkeit vom aktuell anliegenden Workload-Typ mit unterschiedlichen Tuning-Plänen adressiert werden.

Wissen über Ressourcenabhängigkeiten. Auch verschiedene Aspekte des in *Abschnitt 5.1* vorgestellten Wissensmodells können bei der Auswahl von Tuning-Plänen eingebunden werden. So lässt sich beispielsweise das Wissen über die Abhängigkeiten zwischen einzelnen Sensoren und Effektoren in der Planungsphase anwenden und basierend darauf lassen sich Tuning-Pläne identifizieren, die keine oder minimale Seiteneffekte oder Oszillationen verursachen.

Vorliegende Metadaten. Auch diverse Metadaten haben einen entscheidenden Einfluss auf die Tuning-Plan-Auswahl. So kann beispielsweise anhand der Ausführungshistorie sowie der Effektivitätsmetriken einzelner Tuning-Pläne festgestellt werden, welche Tuning-Pläne im vorliegenden Problemfall bessere Tuning-Ergebnisse versprechen. Weiterführende Informationen zu den möglichen bei der Planung zu berücksichtigenden Metadaten finden sich in *Abschnitt 6.1*. Neben den das Verhalten von Tuning-Plänen beschreibenden Ausführungsmetadaten können auch Bewertungen einzelner Tuning-Pläne durch die Community-Mitglieder bei der Tuning-Plan-Auswahl berücksichtigt werden (vgl. *Abschnitt 7.2.2*).

Nicht nur der eigentliche Tuning-Plan, sondern auch der diesen Tuning-Plan ausgelöste CBE wird an den *Tuning Plan Executor* übergeben, um sicherzustellen, dass sämtliche

Problemsituation beschreibenden Informationen für die Tuning-Plan-Ausführung verfügbar sind und bei Bedarf ausgewertet werden können.

Im Rahmen der vorliegenden Arbeit erfolgt auch die Vorstellung eines alternativen Planungsansatzes, bei welchem der Nutzer den entsprechenden Planungsablauf mit Hilfe von sogenannten *Hypothesenbäumen* modellieren kann. Beim Auftreten eines zu lösenden Problems wird statt eines Tuning-Plans der entsprechende Hypothesenbaum identifiziert und schrittweise abgearbeitet. Die Reihenfolge der Traversierung des Hypothesenbaums wird dabei basierend auf den vorliegenden Metadaten bzw. gesammelten Informationen dynamisch angepasst. Eine ausführliche Beschreibung dieses Ansatzes ist dem *Abschnitt 6.3* zu entnehmen.

5.2.5 Ausführung von Tuning-Plänen

Die Ausführungskomponente des ATE namens *Tuning Plan Executor* (*Abbildung 5.4*) 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-Anwendungsserver integrierten, leichtgewichtigen Workflow-Engine. Die einzelnen Schritte eines Tuning-Plans entsprechen dabei Workflow-Aktivitäten. Hierbei kommen sowohl vordefinierte, semantikreiche Aktivitäten als auch benutzerdefinierte, Datenbank-Tuning-spezifische Aktivitäten zum Einsatz (*Abschnitt 4.4.1*). 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 Definition der Schranken kann dabei entweder absolut oder relativ beispielsweise in Abhängigkeit vom verfügbaren Arbeitsspeicher erfolgen.

Das durch den *Tuning Plan Selector* übergebene CBE, welches den Kontext des mittels Tuning-Plans zu lösenden Problems erfasst, kann ausgewertet und beispielsweise in die Steuerung des Kontrollflusses des Tuning-Plans einbezogen werden. Damit wird eine kontextabhängige Anpassung der Tuning-Pläne zur Laufzeit ermöglicht.

Die Ausführung der Tuning-Pläne verursacht eine Zustandsänderung der überwachten Datenbank. Diese Zustandsänderung wird durch die Überwachungskomponente beobachtet und der Kreislauf beginnt erneut.

5.2.6 Kontext-abhängige Rekonfiguration des Tuning-Systems

Wie bereits in *Abschnitt 4.4.1.1* erwähnt, lassen sich mit Hilfe von *Meta-Tuning-Plänen* ereignisgesteuerte Änderungen an der ATE-Konfiguration vornehmen. In unserem Prototyp nutzen wir diese Funktionalität insbesondere für eine Workload-abhängige Anpassung der Schwellenwerte des *Performance Expert*. Für eine an die anliegende Workload angepasste, genauere Definition der atomaren Ereignisse erfolgt dabei bei jeder Veränderung des Typs der anliegenden Workload eine Rekonfiguration der Ausnahmeverarbeitung des PE (vgl. *Tabelle 5.2*, Spalte 2 und 3). Während beispielsweise die Bufferpool Hitratio (PE-Metrik `bufferpool hit ratio`) von weniger als 90% bei einer OLTP-Workload in der Regel als problematisch angesehen wird und entsprechende Reaktionen notwendig sind, kann die

PE-Metrik	PE-Schwellenwert für OLTP	PE-Schwellenwert für OLAP	Atomares Ereignis	Komplexes Ereignis	Tuning-Plan
`bufferpool hit ratio`	$< 90\%$	$< 70\%$	`BPHR`	`BPHR`	*Bufferpool-Tuning*
`post threshold hash joins`	> 0	> 4	`PTHJ`	`PTHJ` *or* `PTS`	*Sheapthres-Tuning*
`post threshold sorts`	> 0	> 4	`PTS`		
`overflowed sorts`	> 2	> 10	`OS`	`OS` *or* `HJO` *or* `HJSO`	*Sortheap-Tuning*
`hash join overflows`	> 0	> 4	`HJO`		
`hash join small overflows`	> 0	> 4	`HJSO`		
`avg number of rows read per selected row`	> 5	> 25	`RRRS`	`RRRS`	*Index-Design*

Tabelle 5.2: Zuordnungen von Metriken, Ereignissen und Tuning-Plänen

akzeptable Unterschranke für die Bufferpool Hitratio bei einer OLAP-Workload mit 70% deutlich niedriger liegen.

Neben den Veränderungen der PE-Ausnahmeverarbeitung sind grundsätzlich auch Modifikationen der Korrelationsmuster des *Event Correlators*, Umkonfiguration des *Workload Classifiers* bzw. Anpassungen des Logging-Verhaltens des ATE denkbar. Auf weitere Erklärungen und Beispiele wurde im Rahmen dieser Arbeit jedoch verzichtet.

5.3 Beispiele umgesetzter Tuning-Pläne

Nachfolgend wird eine exemplarische Auswahl der im Rahmen einer Machbarkeitsstudie umgesetzten Ereignisse und Tuning-Pläne beschrieben. Die Tuning-Pläne beschränken sich im Wesentlichen auf einfache Konfigurationsparameteranpassungen der elementaren Heaps und Caches von DB2. Der eigentliche Fokus unseres universellen, zum DB2-internen Self-Tuning-Memory-Manager-Mechanismus (STMM) [SGAL+06] komplementären Ansatzes liegt jedoch auf Black-Box-orientiertem Tuning von datenbankbasierten Software-Stacks. Der Zugriff auf die zu tunenden Komponenten erfolgt dabei ausschließlich über die nach außen sichtbaren Schnittstellen und erfordert keinen Einblick in die System-Interna.

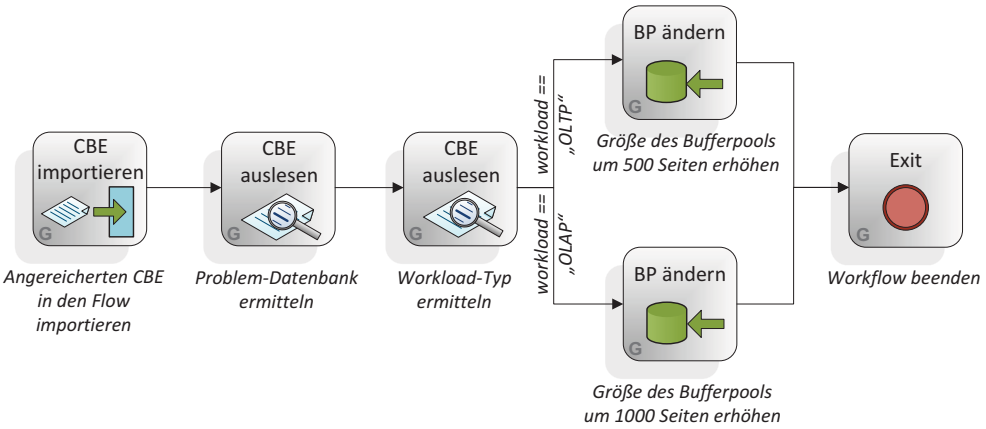


Abbildung 5.6: Graphische Darstellung des Tuning-Plans „*Bufferpool-Tuning*“

Tabelle 5.2 fasst die (vier) 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 Schwellenwerte 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 für jedes komplexe Ereignis Überreaktionen verhindert. Durch die Verwendung dieser Regel lassen sich bei Auftreten eines komplexen Ereignisses für die nachfolgende, vordefinierte Zeitperiode alle eintretenden Ereignisse gleichen Typs ignorieren.

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 (*Abschnitt 4.5.2*). Die Workflows enthalten in dieser konzeptuellen Darstellung zur Erleichterung des Verständnisses der WsM-Konzepte zusätzliche Informationen. Dabei stellen gerundete Rechtecke globale („*G*“) bzw. lokale („*L*“) Aktivitäten dar, die sich wiederum zusammensetzen lassen und dann mit einem „+“ gekennzeichnet sind. 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.

5.3.1 Bufferpool-Tuning

Der Bufferpool-Tuning-Plan (***Abbildung 5.6***) 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. Da es sich bei

den hier angegebenen Tuning-Plänen lediglich um einen Nachweis der Umsetzbarkeit typischer Datenbank-Tuning-Aufgaben mittels Tuning-Plänen handelt, beschränkt sich der Bufferpool-Tuning-Plan in seiner aktuellen Version ausschließlich auf das Tuning des standardmäßig verwendeten Bufferpools *IBMDEFAULTBP* durch die Erhöhung seiner Größe.

Ausgelöst wird der Tuning-Plan durch Auftreten eines entsprechenden Ereignisses, das eine Unterschreitung des Workload-abhängigen Schwellenwertes für die `bufferpool hit ratio` (BPHR) beschreibt (90% für OLTP und 70% für OLAP). Mittels einer speziellen ACT-Filterungsregel werden dabei Überreaktionen verhindert. Ziel dabei ist, ein „Einschwingen“ des Systems nach einer Konfigurationsänderung abzuwarten, bevor gegebenenfalls erneute Tuning-Maßnahmen eingeleitet werden. So lassen sich beispielsweise bei Auftreten eines Ereignisses für die nachfolgende, vordefinierte Zeitperiode alle eintretenden Ereignisse gleichen Typs ignorieren.

Durch die Aktivität `CBE importieren` erfolgt der Zugriff auf die den Problemkontext beschreibenden Metadaten, aus welchen dann 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 in Abhängigkeit von der aktuellen Workload um 500 Seiten für OLTP bzw. 1.000 Seiten für OLAP, was zu einer einhergehenden Verbesserung der BPHR führt. Die in den *Tuning Plan Executor* eingebaute Ressourcen-Beschränkungsfunktionalität verhindert dabei übermäßige Ressourcenallokationen durch Beachtung der vorgegebenen benutzerdefinierten Obergrenzen.

5.3.2 Sortheap- und Sheapthres-Tuning

Wie bereits in *Abschnitt 3.1.1* erläutert, führt DB2 Sortieroperationen durch, sobald in dem abgesetzten SQL-Statement ein explizites `order by`, `group by`, `union` oder `distinct` vorkommen. Aber auch bei OLAP- bzw. Aggregationsfunktionen, sowie bei der Reorganisation, der Index-Erzeugung bzw. bei *Hash Joins* 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 gemeinsamen Sortiervorgängen (*Shared Sorts*). Im Gegensatz dazu handelt es sich um private Sortiervorgänge (*Private Sorts*), wenn dieser Parallelismus inaktiv ist und jeder DB2-Agent zum Sortieren auf seinen privaten Speicherbereich (*Agent Private Memory*) zugreift.

Mittels des Datenbankkonfigurationsparameters `sortheap` wird festgelegt, wie viel Speicher im privaten Speicherbereis jedes DB2-Agenten (*Agent Private Memory*) bzw. im gemeinsamen Speicher (*Database Shared Memory*) für jede einzelne Sortierung zur Verfügung steht. 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*

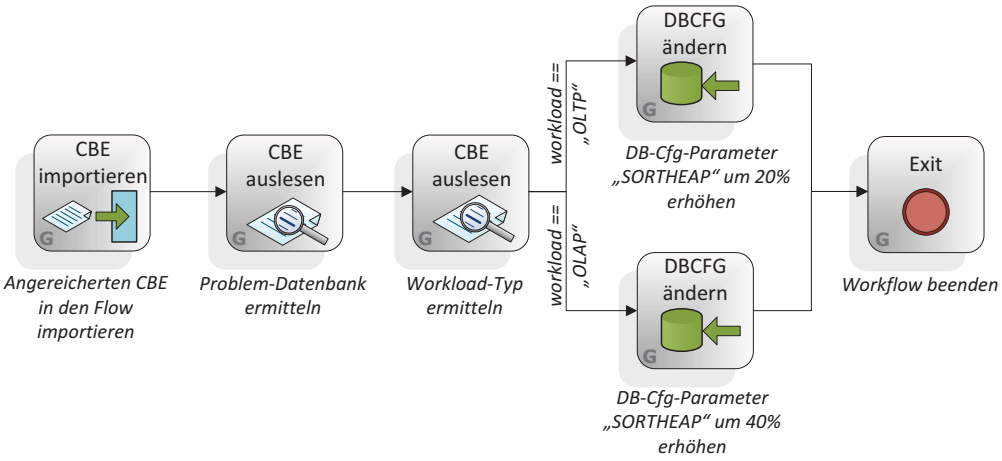


Abbildung 5.7: Graphische Darstellung des Tuning-Plans „*Sortheap-Tuning*“

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, gemeinsamen Sortiervorgängen legt der Konfigurationsparameter des Datenbankmanager (DBM)[23] `sheapthres` die obere Grenze für den gemeinsamen Speicherverbrauch fest, so dass keine weiteren Sortheap-Allokationen zulässig sind, wenn dies zu einer Überschreitung der Grenze führen würde. Für private Sortiervorgänge ist eine Überschreitung dennoch möglich. Dabei wird ein Speicherdrosselungsmechanismus aktiviert, so dass bei nachfolgenden Sortieroperationen weniger Speicher als angefordert allokiert wird. Durch eine kontinuierliche Halbierung des zu allokierenden Sortierspeichers 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 Speicherbegrenzung für Sortiervorgänge durch den Wert des DB-Konfigurationsparameters `sheapthres_shr` möglich. Dazu muss der Parameter `sheapthres` der Instanzebene auf den Wert 0 gesetzt werden.

Der in ***Abbildung 5.7*** dargestellte Sortheap-Tuning-Plan wird abhängig von der aktuellen Workload durch die atomaren Ereignisse `overflowed sorts` (OS), `hash join overflows` (HSO) oder `hash join small overflows` (HJSO) ausgelöst. Durch Erhöhen des für nachfolgende Sortierungen allokierbaren Speichers mittels des Konfigurationsparameters `sortheap` um 20% für OLTP bzw. 40% für OLAP sollen künftig derartige Überläufe vermieden werden.

Der Sheapthres-Tuning-Plan aus ***Abbildung 5.8*** wird jedes Mal dann ausgeführt, wenn die Schwellenwerte für die Metriken `post threshold hash joins` (PTHJ) bzw. `post threshold sorts` (PTS) verletzt werden. In Abhängigkeit von der aktuellen, durch den *Workload Classifier* bestimmten Workload wird ermittelt, wie stark der Wert des

[23] In Anlehnung an die DB2-Terminologie werden in dieser Arbeit die Begriffe *Instanz* und *Datenbankmanager* synonym verwendet.

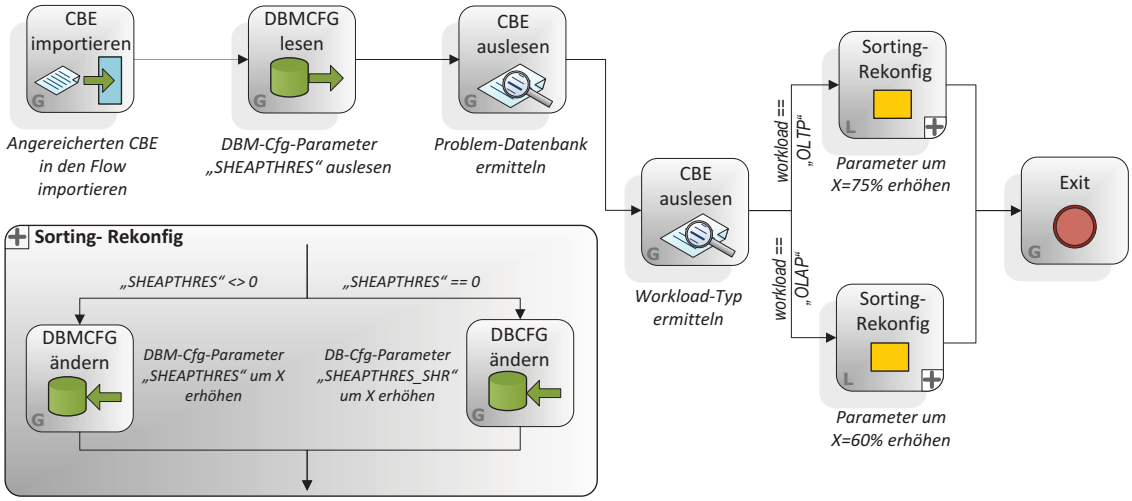


Abbildung 5.8: Graphische Darstellung des Tuning-Plans „*Sheapthres-Tuning*“

entsprechenden Konfigurationsparameters zu erhöhen ist. Die globale, zusammengesetzte Aktivität `Sorting-Rekonfig` führt die Parameteranpassung auf Datenbankebene (`sheapthres_shr`) bzw. Instanzebene (`sheapthres`) in Abhängigkeit vom aktuellen Wert des Konfigurationsparameters `sheapthres` durch.

5.3.3 Index-Design

Die Idee hinter diesem auf dem bereits in [WRRA08] vorgestellten Tuning-Plan basierenden und um zusätzliche Funktionalitäten erweiterten Index-Design-Tuning-Plan besteht darin, den *IBM DB2 Design Advisor* [IBM06f] ereignisgesteuert aufzurufen und ihn dabei mit Informationen über die aktuelle Workload zu versorgen (***Abbildung 5.9***). Die vom *DB2 Design Advisor* erstellten Vorschläge zum Anlegen bzw. Löschen von Indizes sowie zum Aktualisieren von Statistiken sollen anschließend an einen DBA zur Qualitätssicherung per E-Mail weitergeleitet werden. Nach Prüfung der erstellten Vorschläge kann der DBA diese automatisch ausführen lassen oder aber auch verwerfen. Der Tuning-Plan soll immer dann ausgeführt werden, wenn aufgrund fehlender Indizes bedeutend mehr Tupel vom DBMS gelesen werden mussten, als in der Anfrageergebnismenge enthalten sind. Ein Indikator hierfür ist eine Schwellenwertüberschreitung des auf den DB2-Metriken `rows_read` und `rows_selected` basierenden Verhältnisses `rows read / rows selected` (RRRS)[24].

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 Änderungsoperationen (*Insert*, *Update*, *Delete*) zu berücksichtigen, sollen zusätzlich die entsprechenden IUD-Statements zusammen mit ihren Häufigkeiten für den Aufruf des DB2 Design Advisors bestimmt werden.

[24]PE bezeichnet diese Metrik als `avg number of rows read per selected row`.

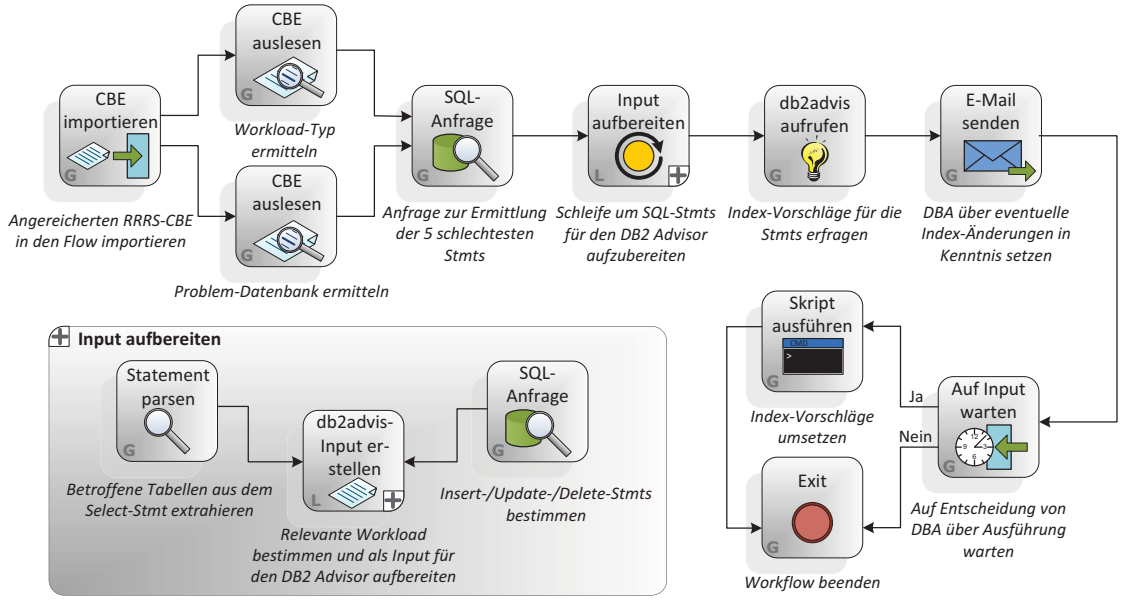


Abbildung 5.9: Graphische Darstellung des Tuning-Plans „*Index-Design*“

Die globale Aktivität `SQL-Anfrage` erlaubt es, SQL-Anfragen gegen beliebige Datenbanken abzusetzen. Sie wird hierbei verwendet, um historische Performancedaten über den *Dynamic Statement Cache* aus der PE-internen Datenbank abzufragen. Der vorher aus dem CBE ermittelte Datenbankname dient dabei der Einschränkung des Anfrageergebnisses 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 `db2-advis-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* in Abhängigkeit von der anliegenden (und erkannten) Workload gesteuert. Für eine OLAP-ähnliche Workload erscheint es sinnvoll, dem *DB2 Design Advisor* mehr Zeit für seine Berechnungen zur Verfügung zu stellen und gegebenenfalls sogar materialisierte Sichten zu berücksichtigen.

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 vorge-

geben 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-Zustandes. Andernfalls wird ein vom *DB2 Design Advisor* erstelltes Skript zur Umsetzung der Designentscheidungen ausgeführt (globale Aktivität `Skript ausführen`).

In der Zeitspanne zwischen dem Versenden der E-Mail und der Reaktion des DBA sind Veränderungen des Workload-Typs (*Workload-Shifts*) möglich, so dass vorgeschlagene Indizes gegebenenfalls keinen Einfluss auf die 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, das heißt der Anzahl der zu betrachtenden Select- und entsprechenden IUD-Statements, zu finden.

Grundsätzlich lässt sich der Index-Design-Tuning-Plan optimieren und weiterentwickeln. Dazu kann beispielsweise untersucht werden, inwieweit sich in den Tuning-Plan einige der in den letzten Jahren entwickelten Ansätze zur Index-Auswahl [SGS03, BC06] integrieren lassen.

5.4 Evaluationsergebnisse

Die in diesem Kapitel vorgestellte Architektur des *Autonomic Tuning Expert* wurde so konzipiert, dass eine Verteilung der einzelnen Architekturkomponenten auf mehrere Rechner möglich ist. Beim Aufbau unserer Evaluationsinfrastruktur haben wir davon profitiert und das gesamte System auf drei Rechner verteilt. Während sich die zu optimierende Datenbank auf einem Rechner befand, war der zweite Rechner für die Ausführung des MAPE-Kreislaufs des ATE verantwortlich. Über entsprechende Schnittstellen hat ATE den Zustand der entfernten, zu tunenden Datenbank überwacht und notwendige Zustandsänderungen vorgenommen.

Für die Erzeugung der Workload wurde ein von der IBM bereitgestellter *Workload Generator* verwendet. Dieser hat auf dem dritten Rechner eine Mischung aus OLTP- und OLAP-Workloads erzeugt und an den Datenbank-Server gesendet. Um bei der Workload-Erzeugung möglichst realistische, aber vor allem wiederholbare Workload-Szenarien für aufeinander folgende Tests mit vergleichbaren Ergebnissen zu erzeugen, wurde der *Workload Generator* auf die Erzeugung von an die Standard-Benchmarks TPC-C [TPC07] und TPC-H [TPC08] angelehnten Workloads eingestellt.

Für die Überwachung des Tuning und Auswertung von Tuning-Ergebnissen wurde ein spezielles Werkzeug namens *KPI Visualizer* entwickelt. Dieser ermöglicht sowohl das Auslesen von zahlreichen Performance-relevanten Indikatoren aus der Datenbank des *Performance Expert* als auch einen integrierten Zugriff auf die vom ATE geschriebenen Log-Einträge, die beispielsweise den Start bzw. Stopp des ATE, aber auch die Zeitpunkte der Event-Erkennung sowie der Tuning-Plan-Ausführung beinhalten.

Initial wurde eine Datenbank mit einer vom *DB2 Configuration Advisor* 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

Tuning-Plan	Einfluss des Tuning-Plans	(Initial-)Wert / Anzahl vor dem Tuning	Wert / Anzahl nach dem Tuning
Bufferpool-Tuning	Größe des Default-Bufferpools	17.354 Seiten	36.354 Seiten
Sortheap-Tuning	Größe von `sortheap`	754 Seiten	40.651 Seiten
Sheapthres-Tuning	Größe von `sheapthres`	2.917 Seiten	52.256 Seiten
Index-Design	OLTP-Indizes	9	21
	OLAP-Indizes	8	33

Tabelle 5.3: Auswahl umgesetzter Tuning-Pläne und deren Wirkung

zu können, haben wir lediglich die jeweils 9 OLTP- bzw. 8 OLAP-Primärschlüssel-Indizes anlegen lassen und auf weitere verzichtet.

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 wurde dafür Sorge getragen, dass Caches und Heaps geleert wurden. Die Testläufe wurden mit einer Auswahl von Tuning-Plänen, die sich über die Bereiche des Tuning von DB2-Heaps, -Caches und des Index-Designs erstrecken, durchgeführt. ***Tabelle 5.3*** stellt eine für die Evaluationsergebnisse 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 die Erhöhung von Konfigurationsparameterwerten[25]. Die in diesem Zusammenhang potentiell auftretenden Ressourcen-Überallokationen werden durch das Ressourcen-Restriktionskonzept des *Tuning Plan Executors* unter Beachtung nutzerdefinierter Obergrenzen zurückgewiesen (*Abschnitt 5.2.5*).

Die Effektivität des Tuning 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 normierte Anzahl von Sortierungen (`total sorts`); sowie das Verhältnis von gelesenen Tupeln zu Ergebnistupeln (`rows read / rows selected`) [IBM06h].

Jedes Diagramm der ***Abbildung 5.10*** veranschaulicht die Werteverläufe der entsprechenden Metrik sowohl *ohne* als auch *mit* Tuning. 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 lediglich bei ca. 25%, wodurch der Schwellenwert für das Auslösen des atomaren

[25] Erklärungen zu den jeweiligen Konfigurationsparametern finden sich in [IBM06f].

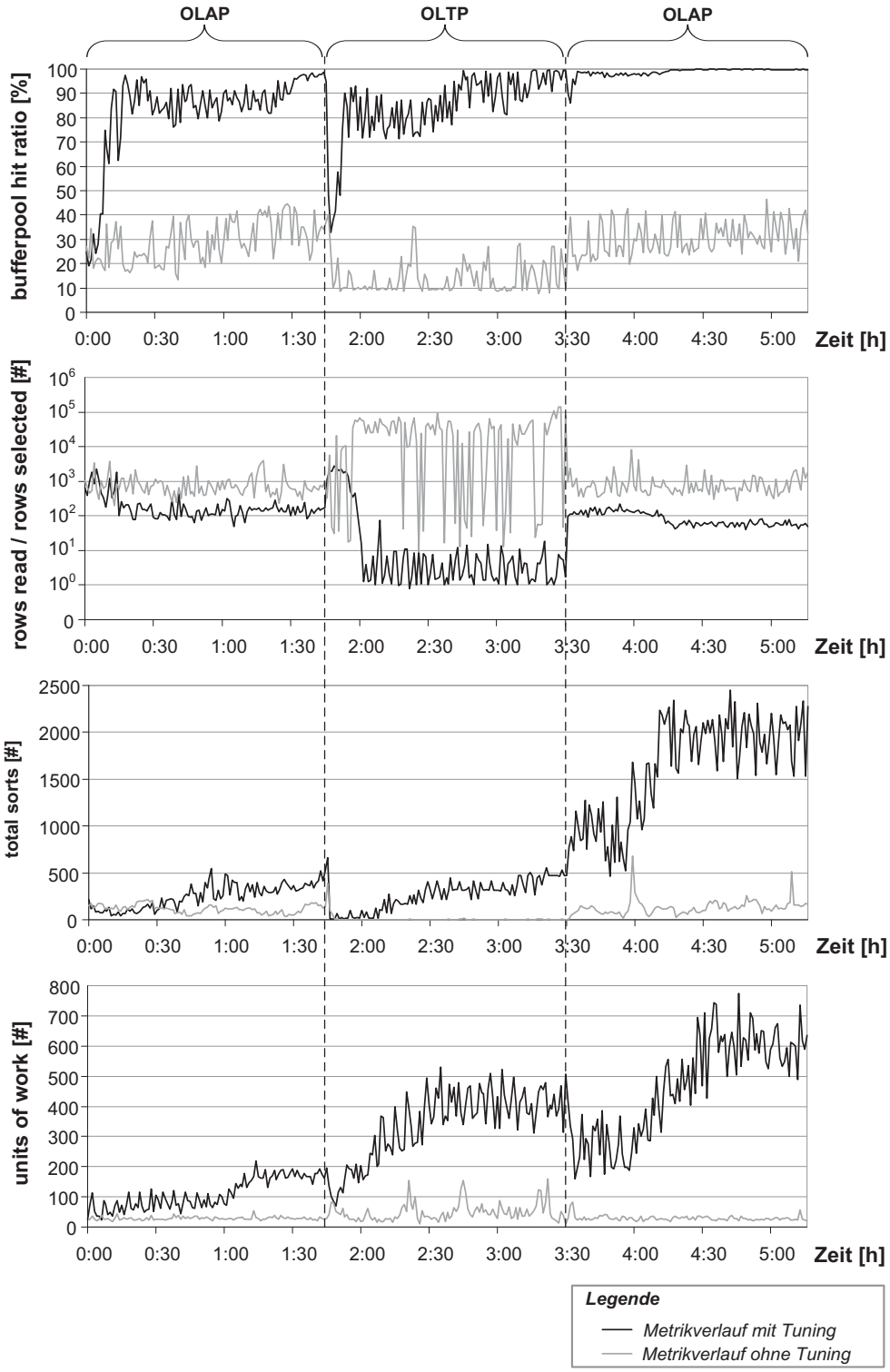


Abbildung 5.10: Ausgewählte Ergebnisse der Testläufe

BPHR-Ereignisses verletzt ist. Als Reaktion auf das Unterschreiten des Schwellenwertes wird der Bufferpool-Tuning-Plan angestoßen, welcher die Größe des Bufferpools um 1.000 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 Tuning steigt die Größe des Bufferpools von initial 17.354 Seiten auf 36.354 Seiten.

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 durch schlechte Indexausnutzung oder gar nicht vorhandene Indizes. Da anfänglich bewusst nur die Primärschlüssel-Indizes 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 Indizes 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 PE-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 Anzahl der Transaktionen repräsentierende Metrik `units of work` (UOW) unmittelbar nach Aktivieren des Tuning 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 (Systemdurchsatz). 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.

5.5 Zusammenfassung

Zur Unterstützung einer probleminitiierten Ausführung von Tuning-Plänen wurde im vorliegenden Kapitel ein generisches Wissensmodell vorgestellt, welches die Erfassung des für alle Phasen des Tuning-Kreislaufs notwendigen Wissens ermöglicht (*Abschnitt 5.1*). Neben den Informationen über die Ereignisse bzw. die Tuning-Pläne selbst, die eine Identifikation und Auflösung von Problemen ermöglichen, spielen dabei insbesondere die Topologie-Informationen eine entscheidende Rolle. Das Wissen über die Systemabhängigkeiten – ein wichtiger Bestandteil des Topologie-Wissens – ist für ein effektives System-Tuning unabdingbar. Insbesondere bei der Planung des Vorgehens zur Behebung erkannter Probleme kann von Informationen darüber, unter welchen Systemressourcen welche Abhängigkeiten bestehen, profitiert werden. Auch das Wissen über die durch einzelne Operationen verursachten Kosten findet bei der Planung Anwendung und ist keinesfalls zu vernachlässigen. Zur Auflösung einzelner Probleme kann es unter Umständen mehrere Wege geben. Die Herausforderung besteht dabei darin, den Weg (Tuning-Plan) zu identifizieren, der das Problem im akzeptablen Zeitrahmen mit akzeptablen Kosten bzw. Nebenwirkungen behebt. Die Information darüber, was für den Administrator akzeptabel ist, wird im Policy-Wissen erfasst und kann ebenfalls zur Laufzeit ausgewertet und entsprechend berücksichtigt werden. Im Rahmen der vorliegenden Arbeit wurden die Kosten der einzelnen Effektoren bzw. ganzer Tuning-Pläne jedoch nur in einem begrenzten Umfang betrachtet.

In *Abschnitt 5.2* wurde eine Architektur vorgestellt, die im Rahmen einer Kooperation mit IBM entstanden ist und ein autonomes, best-practices-orientiertes Tunen von datenbankbasierten Anwendungen ermöglicht. Dabei handelt es sich um eine generische Referenzarchitektur, die auf dem MAPE-Paradigma aufbaut. Zwar basiert die aktuelle Implementierung dieser Architektur auf IBM-Produkten, sowohl die der Architektur zugrunde liegenden Konzepte als auch die Architektur selbst sind jedoch Produkt-unabhängig und verallgemeinerbar. Darüber hinaus ist das gewählte Architekturdesign nicht auf das autonome Tunen von DBMS beschränkt.

Im Rahmen einer Machbarkeitsstudie wurde ein Satz von auf das Tuning eines DB2-Systems zugeschnittenen Tuning-Plänen (*Abschnitt 5.3*) entwickelt und zur Validierung der Funktionsweise des *Autonomic Tuning Expert* eingesetzt. Das System wurde anschließend verschiedenen Tests unterzogen. In *Abschnitt 5.4* wird das Ergebnis eines aus zwei Testläufen bestehenden Tests beschrieben. Dabei wurde ein entsprechend initialisiertes DB2-System einer wechselnden Workload (Abfolge von zwei Workload-Typen OLTP und OLAP) ausgesetzt. Beim ersten Testlauf ohne Tuning wurde lediglich eine Auswahl von Performance-relevanten Metriken erfasst. Im zweiten auf der gleichen Initial-Konfiguration basierenden Testlauf wurde ATE aktiviert und die Ergebnisse des Tuning durch die Erfassung der gleichen Performance-relevanten Metriken gemessen. Durch einen anschließenden Vergleich der Metrikverläufe beider Testläufe konnte die Effektivität des Tuning nachgewiesen werden.

Neben den in *Abschnitt 5.3* vorgestellten Tuning-Plänen wurden zahlreiche weitere Tuning-Pläne umgesetzt [WR07c]. Dazu zählen beispielsweise folgende:

- Tuning-Plan zur Reduktion von Sperreskalationen und Warte-Situationen aufgrund von lang gehaltenen Sperren bzw. Deadlocks.

- Tuning-Plan zur Aktualisierung von Statistiken, der sich entweder probleminitiiert oder bei Bedarf durch andere Tuning-Pläne anstoßen lässt.
- Tuning-Plan zur Bestimmung von bezüglich ihrer Ausführungszeit bzw. der Anzahl von Ausführungen teuersten SQL-Anweisungen. Die identifizierten Anweisungen lassen sich anschließend beispielsweise dem zuständigen Administrator per E-Mail zuschicken und unterstützen somit eine weiterführende Problem-Diagnose.
- Tuning-Pläne zum Vornehmen Workload-abhängiger Initialkonfigurationseinstellungen wie beispielsweise Grad des partitionsinternen Parallelismus, Anzahl von Agenten zum Füllen (*I/O Prefetchers*) bzw. Leeren (*I/O Cleaners*) des Bufferpools etc. Diese Tuning-Pläne lassen sich beispielsweise bei jedem Workload-Wechsel ausführen und bereiten somit das System auf den aktuellen Workload-Typ optimal vor.

Zur Erhöhung des Autonomie-Grades des ATE-Frameworks lassen sich einerseits die Verfahren zur Auswahl von Tuning-Plänen erweitern. Dabei können unter anderem mathematische Optimierungsalgorithmen Anwendung finden. Andererseits ist die Möglichkeit zur Angabe von Zielvorgaben für ein autonomes System unabdingbar. Im *Kapitel 6* werden daher ausgewählte Aspekte vorgestellt, die sowohl eine Verbesserung der Tuning-Plan-Auswahl als auch die Definition von Tuning-Richtlinien ermöglichen.

Kapitel 6

Steuerung des Tuning-Prozesses

Die Grundvoraussetzung für die Ausführung von Tuning-Plänen ist die Planung, sprich die Entscheidung, welche Tuning-Pläne in welcher Reihenfolge in Reaktion auf welche Ereignisse ausgeführt werden. Dabei ist es wichtig, nicht nur die Tuning-Pläne zu identifizieren, die die beste Wirkung bei der Auflösung von erkannten Problemen versprechen, sondern insbesondere auch weitere Nebenbedingungen zu berücksichtigen. Durch entsprechende Planungsvorgehensweisen lassen sich beispielsweise Oszillationen vermeiden bzw. die Historie der vergangenen Tuning-Plan-Ausführungen in die Tuning-Plan-Auswahl miteinbeziehen. Unter Umständen können die Tuning-Pläne auch während ihrer Ausführung an die geltenden Nebenbedingungen, beispielsweise durch eine Veränderung der Rekonfigurationsschrittweite, angepasst werden.

In diesem Kapitel werden Konzepte vorgestellt, die zu einer Steigerung der Planungsqualität des *Autonomic Tuning Expert* beitragen. Ein Ansatz erweitert dabei beispielsweise die in den bisherigen Betrachtungen im Rahmen der vorliegenden Arbeit übliche Planungsphase, die aus einer probleminitiierten Auswahl von Tuning-Plänen besteht, um eine darauf aufbauende Auswahlverfeinerungsphase. Dieser Auswahlverfeinerungsansatz basiert dabei primär auf der Auswertung und Berücksichtigung der die Tuning-Plan-Semantik, -Historie und -Effektivität erfassenden Metadaten (*Abschnitt 6.1*) sowie der durch einen Administrator festgelegten Tuning-Richtlinien (*Abschnitt 6.2*).

Mit Hilfe von Tuning-Richtlinien wird dabei nicht nur die Verfeinerung der Tuning-Plan-Auswahl umgesetzt, sondern unter anderem auch das Pruning ermöglicht, also die Deaktivierung von den Administratorvorgaben widersprechenden Tuning-Plänen und sie auslösenden Ereignissen bzw. die Berücksichtigung von durch die Administratoren vorgegebenen Informationen beispielsweise über das zu erwartende Nutzerverhalten.

Ein alternativer Ansatz wird in *Abschnitt 6.3* vorgestellt. Zwar spielen bei diesem Ansatz die Metadaten sowie die Tuning-Richtlinien ebenfalls eine Rolle, der Schwerpunkt liegt jedoch auf der Entwicklung von Konzepten, die die Administratoren bei der Definition der Planungsabläufe unterstützen. Die Möglichkeit zur Erfassung bewährter Tuning-Maßnahmen durch die Erstellung von Tuning-Plänen wird also nun um eine Möglichkeit zur Definition der Planungslogik erweitert. Der Administrator kann somit mit Hilfe von sogenannten Hypothesenbäumen das Tuning ganzer Problembereiche planen.

Eine Zusammenfassung der in diesem Kapitel vorgestellten Konzepte zur Unterstützung und Erweiterung der Planungsphase des MAPE-basierten autonomen Datenbank-Tuning findet sich in *Abschnitt 6.4*.

6.1 Metadaten zur Unterstützung der Tuning-Steuerung

Die im Tuning-Plan-Header enthaltenen Metadaten (vgl. *Abschnitt 4.4*) können unter anderem zur Laufzeit dazu genutzt werden,

- die für eine bestimmte Konstellation nicht zutreffenden bzw. nicht sinnvollen Tuning-Pläne herauszufiltern,
- den optimalen Tuning-Plan zur Lösung eines bestimmten Problems zu bestimmen oder
- bei der Ausführung von Tuning-Plänen beispielsweise Überallokationen von Ressourcen zu vermeiden.

Nachfolgend findet sich eine Auswahl der wichtigsten Metadaten. Auf die Darstellung von zur Umsetzung von Community-Funktionen (wie beispielsweise Bewertung von Tuning-Plänen) notwendigen Metadaten wurde dabei verzichtet. Eine zusammenfassende Darstellung der hier vorgestellten Metadatenarten findet sich in ***Abbildung 6.1***.

6.1.1 Verwaltungsmetadaten

Zu den klassischen Verwaltungsmetadaten der Tuning-Pläne zählen Angaben wie

- Name,
- Autoren,
- Erstellungsdatum sowie
- Änderungsdatum.

6.1.2 Deskriptive Metadaten

Deskriptive Metadaten beschreiben den Inhalt bzw. den Anwendungszweck der Tuning-Pläne. Sie können sowohl zur Suche von Tuning-Plänen als auch zur Steuerung der Tuning-Plan-Auswahl bzw. -Ausführung verwendet werden. Die deskriptiven Metadaten sind im Folgenden exemplarisch zusammengefasst:

Beschreibung. Die Beschreibung bzw. Dokumentation eines Tuning-Plans ist grundsätzlich ein Freitextfragment, welches die Funktionsweise des Tuning-Plans skizziert und primär für die Nutzer gedacht ist. Insbesondere soll die Tuning-Plan-Beschreibung nicht für eine maschinelle Auswertung optimiert werden, stattdessen soll auf das Konzept der Tuning-Absichten (siehe unten) zurückgegriffen werden.

Stichwörter. Mit Hilfe von Stichwörtern lässt sich der Inhalt des Tuning-Plans erfassen. Dabei sind Konventionen bezüglich der Wahl sowie Schreibweise der Stichwörter zu treffen. Gegebenenfalls können diverse Normalisierungs- bzw. Lematisierungsansätze berücksichtigt bzw. ein Thesaurus eingesetzt werden.

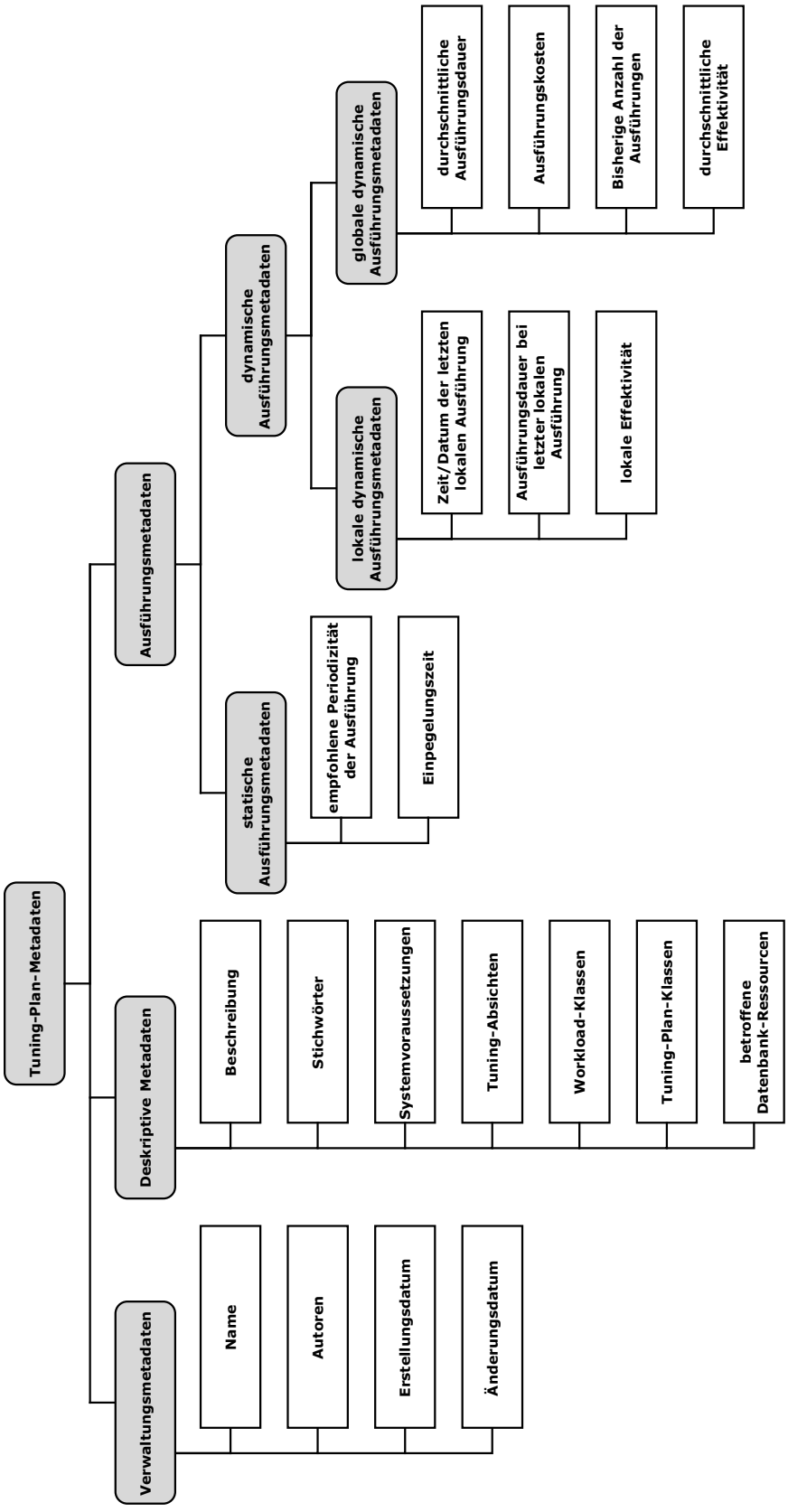


Abbildung 6.1: Ein Überblick über die Metadaten der Tuning-Pläne

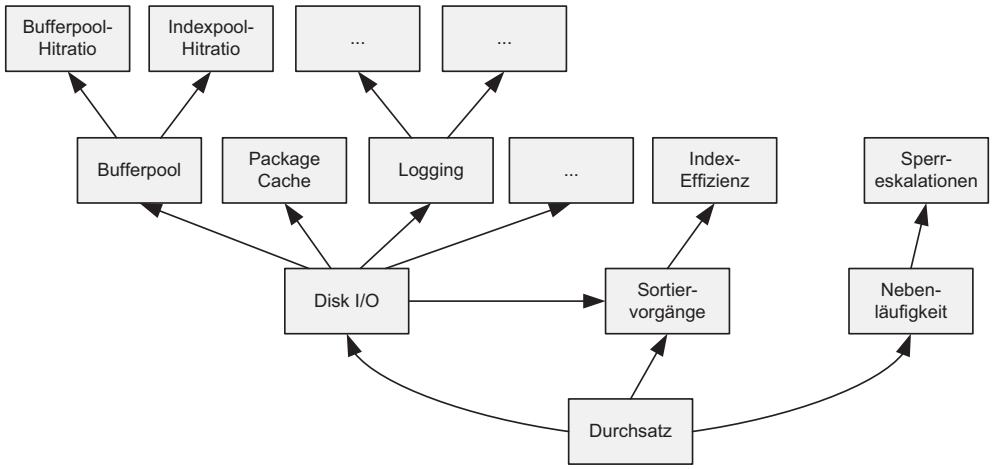


Abbildung 6.2: Ein Ausschnitt des Tuning-Absichten-Graphen

Systemvoraussetzungen. Tuning-Pläne sind in der Regel auf bestimmte Systeme und unter Umständen sogar Software-Versionen zugeschnitten. Systemvoraussetzungen wie beispielsweise bestimmte Hardware, Betriebssystem oder ein bestimmtes DBMS erlauben eine Spezifikation von systemspezifischen Nebenbedingungen, die für eine erfolgreiche Ausführung eines Tuning-Plans erfüllt sein müssen.

Tuning-Absichten. Mit Hilfe von Tuning-Absichten kann erfasst werden, welchen Zweck die Tuning-Pläne hauptsächlich erfüllen. Um eine Vergleichbarkeit der Tuning-Pläne zu gewährleisten, lassen sich Abhängigkeiten zwischen den einzelnen Tuning-Absichten bestimmen und in einem entsprechenden Modell erfassen. Dadurch lassen sich indirekt auch die Abhängigkeiten zwischen den einzelnen Tuning-Plänen abbilden. Das Modell zur Verwaltung von Abhängigkeiten zwischen den Tuning-Absichten lässt sich anhand des Tuning-Absichten-Graphen graphisch veranschaulichen. Dabei handelt es sich um einen gerichteten, azyklischen Graphen. Seine Knoten repräsentieren einzelne Tuning-Absichten und die gerichteten Kanten repräsentieren die Abhängigkeiten dazwischen. Eine gerichtete Kante von der Tuning-Absicht A zur Tuning-Absicht B gibt dabei an, dass das durch die Tuning-Absicht A repräsentierte Ziel durch die Erreichung des durch die Tuning-Absicht B erfassten Ziels unterstützt bzw. ermöglicht wird.

Wie in ***Abbildung 6.2*** zu erkennen ist, kann der Systemdurchsatz beispielsweise durch das Tuning der *Disk I/O*, der Sortiervorgänge bzw. der Nebenläufigkeit optimiert werden. Die *Disk I/O Performance* wird wiederum durch die Performance der Sortiervorgänge, der Bufferpools, des *Package Caches* bzw. des Loggings beeinflusst. Die Bufferpool-Performance lässt sich durch das Tuning der Bufferpool Hitratio bzw. der Indexpool-Hitratio optimieren. Die Sortiervorgänge können durch die Optimierung der Index-Effizienz und die Nebenläufigkeit durch die Reduktion der Sperreskalationen optimiert werden. Bei den soeben skizzierten Zusammenhängen handelt es sich lediglich um einen Ausschnitt eines typischen Tuning-Absichten-Graphen. Die in diesem Ausschnitt enthaltenen Abhängigkeiten wurden bereits größtenteils in *Abschnitt 3.1.2* erläutert.

Die Angabe von Tuning-Absichten zu den einzelnen Tuning-Plänen lässt sich implementierungstechnisch durch einen Verweis von Tuning-Plänen zu entsprechenden Elementen des Tuning-Absichten-Graphen umsetzen.

Das Tuning-Absichten-Modell setzt einerseits eine vereinfachte Semantikerfassung der Tuning-Pläne um. Es bietet eine Kategorisierung von Tuning-Plänen und kann zu einer navigierenden, semantischen Suche der Tuning-Pläne verwendet werden (*Abschnitt 7.1.6*). Andererseits werden durch dieses Modell sowohl das Pruning von Tuning-Plänen zur Laufzeit anhand von *Direction Policies* als auch eine Unterstützung der Tuning-Plan-Auswahl ermöglicht (*Abschnitt 6.2.4*).

Die Angabe von Tuning-Absichten zu den Tuning-Plänen hat durch die Autoren der Tuning-Pläne zu erfolgen. Aufgrund einer eingeschränkten Mächtigkeit der Tuning-Plan-Beschreibungssprache (vgl. *Abschnitt 4.4*) lassen sich jedoch Verfahren entwickeln, um in einzelnen Fällen die Tuning-Absichten aus den in Tuning-Plänen verwendeten Tuning-Schritten und ihren Metadaten abzuleiten.

Workload-Klassen. Durch die Workload-Klasse kann angegeben werden, für welchen Workload-Typ der Tuning-Plan entwickelt bzw. optimiert wurde. Zur Laufzeit werden nur Tuning-Pläne berücksichtigt, deren Workload-Klasse der aktuellen (erkannten) Workload entspricht.

Tuning-Plan-Klassen. Mit Hilfe von Tuning-Plan-Klassen können Tuning-Pläne nach beliebigen Kriterien gruppiert werden. Für einzelne Tuning-Plan-Klassen können anschließend beispielsweise Gültigkeitszeiträume bzw. Prioritäten mit Hilfe von entsprechenden Tuning-Richtlinien (*Abschnitt 6.2*) angegeben werden.

Datenbank-Ressourcen. Durch ein Mitführen von Informationen, welche Datenbank-Ressourcen bzw. -Objekte durch den Tuning-Plan erstellt, gelöscht oder modifiziert werden, wird die Möglichkeit geboten, bei Bedarf die Tuning-Pläne zu identifizieren, die bestimmten Nutzervorgaben widersprechen. Diese Art von Metadaten kann sowohl vom DBA angegeben als auch automatisch von der Tuning-Plan-Definition abgeleitet werden.

6.1.3 Ausführungsmetadaten

Ausführungsmetadaten dienen der Ausführungssteuerung und -beschreibung von Tuning-Plänen. Des Weiteren können die globalen, dynamischen Ausführungsmetadaten in die Suchanfragen der Nutzer auf dem Community-Server miteinbezogen werden (*Abschnitt 7.1.6*). Es lässt sich zwischen statischen und dynamischen Ausführungsmetadaten unterscheiden. Während die *statischen Ausführungsmetadaten* grundsätzlich von TP-Autoren anzugeben sind, basieren die *dynamischen Ausführungsmetadaten* auf Systemstatistiken. Eine kurze Zusammenstellung dieser beiden Arten der Ausführungsmetadaten findet sich in den nachfolgenden Abschnitten.

6.1.3.1 Statische Ausführungsmetadaten

Empfohlene Periodizität der Ausführung. Die empfohlene Periodizität der Ausführung gibt an, wie oft ein Tuning-Plan ausgeführt werden sollte. Indirekt lässt sich daraus

ableiten, ob ein Tuning-Plan nach einer längeren Nicht-Ausführung effizienter ist. Ein einfaches Beispiel dafür ist ein `Reorg`, welches nach einer längeren Zeit, in der es nicht genutzt wurde, effektiver ist.

Einpegelungszeit. Die Einpegelungszeit gibt an, nach welcher Zeit die Wirkung des Tuning-Plans sichtbar wird. Gründe dafür, dass der Effekt der Tuning-Plan-Ausführung nicht unmittelbar sichtbar wird, können beispielsweise Caches sein, die zuerst gefüllt oder bestimmte Thresholds, die überschritten werden müssen. Typischerweise wird dieser Wert vom Autor des Tuning-Plans angegeben. Es ist jedoch auch denkbar, den Wert in Abhängigkeit von bestimmten Statistiken über die Tuning-Plan-Ausführungen automatisch anzupassen.

6.1.3.2 Auf Statistiken basierende, dynamische Ausführungsmetadaten

Dynamische Ausführungsmetadaten spiegeln die Ausführungsaspekte einzelner Tuning-Pläne wider und können daher insbesondere bei der Planung (Tuning-Plan-Auswahl) Anwendung finden. Geht man von dem einfacheren Fall eines einzelnen Systems aus, welches durch einen Satz vorliegender Tuning-Pläne optimiert wird, so sind lediglich die nachfolgend dargestellten lokalen, dynamischen Ausführungsmetadaten zu betrachten. Eines der Ziele der vorliegenden Arbeit ist jedoch die Betrachtung einer Community von Datenbankadministratoren, die ihr Wissen in Form von Tuning-Plänen untereinander austauschen. Im Hinblick darauf ist die Betrachtung sogenannter globaler, dynamischer Ausführungsmetadaten unabdingbar. Diese bestimmen sich durch eine Aggregation vorliegender lokaler Daten und bieten eine nutzerübergreifende Sicht auf die Ausführungsaspekte der Tuning-Pläne.

Lokale dynamische Ausführungsmetadaten

Eine Auswahl lokaler dynamischer Ausführungsmetadaten findet sich nachfolgend:

Zeit/Datum der letzten lokalen Nutzung. Manche Tuning-Pläne mögen bessere Ergebnisse erzielen, wenn sie längere Zeit nicht genutzt wurden. In diesen Fällen stellt diese Größe eine Hilfe zur Entscheidungsfindung dar. Für solche Tuning-Pläne ist vom DBA die empfohlene Periodizität der Ausführung anzugeben.

Ausführungsdauer beim letzten lokalen Durchlauf. Die Ausführungsdauer beim letzten lokalen Durchlauf auf eigenem System gibt an, wie lange der letzte Durchlauf eines Tuning-Plans benötigte. Damit lässt sich nach kurzfristigen Kriterien ein Eindruck gewinnen, wie schnell ein Tuning-Plan ist. Die Tuning-Plan-Ausführung soll dabei nicht zu lange zurück liegen, da sich der Systemzustand (in Abhängigkeit von der anliegenden Workload) mit der Zeit verändern kann.

Lokale Effektivität. Zur Steigerung der Qualität der Planungsphase ist die Information bezüglich der Effektivität einzelner Tuning-Pläne von großer Bedeutung. Damit können beim Vorliegen mehrerer Tuning-Pläne zur Lösung des aufgetretenen Problems nicht nur die Tuning-Pläne mit geringsten Ausführungskosten oder der kürzesten Einpegelungszeit, sondern insbesondere mit höchster Effektivität selektiert werden.

Weiterführende Betrachtungen der Effektivität der Tuning-Pläne finden sich in *Abschnitt 6.1.4*.

Globale dynamische Ausführungsmetadaten

Da die auf den lokalen Nutzersystemen gesammelten Ausführungsmetadaten einzelner Tuning-Pläne aufgrund unterschiedlicher Systemkonfigurationen bzw. einiger nicht erfassbarer Nebenbedingungen stark voneinander abweichen können, ist die Bestimmung durchschnittlicher Werte sinnvoll. Diese aggregierten Werte können auf einem zentralen System (Community-Server, vgl. *Kapitel 7*) gesammelt und bei Bedarf zusammen mit den Tuning-Plänen auf die lokalen Systeme übertragen und beispielsweise bei der Planung berücksichtigt werden.

Durchschnittliche Ausführungsdauer. Die Verwaltung der durchschnittlichen Ausführungsdauer des Tuning-Plans ermöglicht eine Unterscheidung zwischen zeitintensiven und schnellen Tuning-Plänen. Dabei handelt es sich jedoch lediglich um die Dauer der Ausführung ohne Berücksichtigung des Nutzens der Tuning-Pläne.

Ausführungskosten. Die Ausführungskosten geben an, wie „teuer“ die Ausführung eines bestimmten Tuning-Plans ist. Die Ausführungskosten können beispielsweise mit I/O oder Anzahl der CPU-Takte gemessen werden. Es ist jedoch auch möglich, die Ausführungskosten mit Hilfe abstrakterer Operationsklassen zu kategorisieren. Dabei kann beispielsweise zwischen folgenden Operationsklassen unterschieden werden:

- **Kompensierbar / nicht-kompensierbar.** Ein fälschlicherweise oder „wirkungslos“ ausgeführter Tuning-Plan (bzw. ein einzelner Tuning-Schritt) kann unter Umständen wieder kompensiert werden. Dazu muss der Systemzustand vor der Ausführung dieses Tuning-Plans weitgehend wiederhergestellt werden. Wurden beispielsweise Indizes angelegt, so sind diese wieder zu löschen. Einige Operationen, wie eine durchgeführte Reorganisation (`Reorg`), lassen sich jedoch nicht kompensieren.
- **Eingriff in die Semantik / kein Eingriff in die Semantik.** Einige Tuning-Pläne greifen potenziell in die Anwendungssemantik ein, indem sie beispielsweise eine Änderung der Isolationsstufe vornehmen. In solchen Fällen ist jedoch Vorsicht geboten. Unter Umständen sind in solche Tuning-Pläne Mechanismen zu integrieren, um vor den Änderungen eine Bestätigung des Administrators einzuholen.

Die Ausführungskosten können initial vom DBA geschätzt und angegeben und anschließend, zur Laufzeit, vom System angepasst werden. Liegt ein Ressourcenmodell vor, welches die Kosten einzelner Ressourcenänderungen modelliert, so lassen sich durch die Analyse einzelner Tuning-Schritte ihre Kosten ermitteln. Diese können wiederum zu Gesamtkosten des Tuning-Plans aggregiert werden.

Bisherige Anzahl der Ausführungen. Durch die Erfassung der bisherigen Gesamtanzahl der Ausführungen wird eine Normalisierung anderer Metadaten ermöglicht. Für einen direkten Zugriff durch die Nutzer bzw. ATE-Komponenten ist dieses Metadatum jedoch kaum geeignet (vgl. *Abschnitt 7.1.6*).

Durchschnittliche Effektivität. Durch eine Aggregation der Angaben bezüglich der lokalen Effektivität des Tuning-Plans auf unterschiedlichen Nutzersystemen kann die globale Effektivität dieses Tuning-Plans abgeschätzt werden.

6.1.4 Effektivität und ihre Bewertungsmaßstäbe

Effektivität beschreibt die „Güte“ eines Tuning-Plans. Um eine Bewertung der Effizienz eines Tuning-Plans zu ermöglichen, muss zunächst festgelegt werden, welche Tuning-Plan-Ausführung als erfolgreich und welche als nicht erfolgreich anzusehen ist.

Der Erfolg einer Ausführung lässt sich durch die Auswertung eines beliebigen vordefinierten Kriteriums bestimmen (*Tuning-Plan-Qualitätsbewertung*). Geeignete Metriken zur Definition von Erfolgskriterien sind beispielsweise Performance-relevante Metriken wie die die Anzahl durchgeführter Transaktionen repräsentierende Metrik *Units of Work* (UOW), die *Bufferpool Hitratio* (BPHR), die Anzahl von Sortiervorgängen oder ähnliches. Zur Bewertung einzelner Tuning-Pläne lassen sich dabei Metriken einsetzen, auf Basis welcher die Tuning-Plan-Events definiert wurden. Wird die Effektivität der Tuning-Pläne nach verschiedenen Kriterien bewertet, so ist eine Tuning-Plan-übergreifende Vergleichbarkeit jedoch kaum gegeben. Es ist daher wünschenswert, einen allgemeingültigen Bewertungsmaßstab zu finden.

Die Tuning-Plan-Qualitätsbewertung kann absolut bzw. relativ erfolgen [Ehl07]. Dies soll am folgenden Beispiel veranschaulicht werden: Nehmen wir an, es treten x Sperreskalationen pro Minute auf. Wenn diese nach der Tuning-Plan-Ausführung und der Einpegelungszeit um $y\%$ gesunken sind, könnte dies als Erfolg gewertet werden. Dies wäre eine relative Bewertung der Tuning-Plan-Qualität. Eine absolute Tuning-Plan-Qualitätsbewertung hingegen wäre durch eine feste Schranke durchführbar, unter welche die Anzahl der Sperreskalationen pro Minute sinken muss.

Die Effektivität lässt sich nun beispielsweise durch *„Anzahl der erfolgreichen Ausführungen“/ „Anzahl der gesamten Ausführungen“* bestimmen[26]. Das Ergebnis dieser Berechnungsvorschrift liegt im Intervall $[0, 1]$. Es ist anzumerken, dass die Effektivität die Ausführungskosten eines Tuning-Plans nicht berücksichtigt. Eine Kombination aus beiden Metadaten erscheint daher sinnvoll.

6.1.5 Benutzerdefinierte Tuning-Plan-Bewertungskriterien

Zu einem erkannten Problem bestimmt ATE zur Laufzeit einen Tuning-Plan, der das Problem am effektivsten löst. Die unter Umständen komplexe Auswahl erfolgt dabei basierend auf den Tuning-Plan-Metadaten. Insbesondere bei einer benutzerdefinierten Umsetzung der Planungslogik (*Abschnitt 6.3*) kann es dabei unter Umständen vorkommen, dass die Menge der vordefinierten dynamischen Ausführungsdaten zur Ausführungssteuerung von Tuning-Plänen nicht ausreicht. In solchen Fällen kann es also sinnvoll sein, neue Bewertungskriterien einschließlich ihrer Berechnungsvorschrift für Tuning-Pläne zu definieren. Dabei muss jedoch berücksichtigt werden, dass eine Tuning-Plan-übergreifende

[26] An dieser Stelle ist zu erwähnen, dass für die Metadaten, die sich aus anderen Metadaten errechnen, neben den errechneten Werten auch die entsprechenden Berechnungsvorschriften zu verwalten sind.

Vergleichbarkeit nur dann gewährleistet werden kann, wenn die Effektivität der betrachteten Tuning-Pläne nach gleichen Kriterien bewertet wird.

Durch die Architektur vom ATE bedingt, werden nur die Tuning-Pläne bei der Auswahl betrachtet, die direkt oder indirekt zur Lösung des Problems beitragen und daher durch eine entsprechende Tuning-Absicht gekennzeichnet sind (vgl. *Abschnitte 6.2.4.1, 6.2.4.2*). Aus diesem Grund müssen diese Tuning-Pläne im Sinne ihrer Qualitätsbewertung untereinander vergleichbar sein. Sollen also neue Bewertungskriterien für einen oder mehrere Tuning-Pläne definiert werden, so ist darauf zu achten, dass diese Bewertungskriterien zugleich für alle anderen Tuning-Pläne, die der/den gleichen Tuning-Absicht(en) zugeordnet sind, definiert werden.

Um auszuschließen, dass die Vergleichbarkeit von Tuning-Plänen durch die Einführung nutzerdefinierter Tuning-Plan-Qualitätsbewertungskriterien beeinträchtigt wird, ist es ratsam, eine Grundvergleichbarkeit der Tuning-Pläne durch die Einführung von Pflicht-Ausführungsmetadaten zu gewährleisten. Dazu eignen sich insbesondere die vordefinierten dynamischen Ausführungsmetadaten (*Abschnitt 6.1.3.2*). Selbstdefinierte Ausführungsmetadaten sind als optional anzusehen. Sie können bei einer benutzerdefinierten Umsetzung der Planungslogik ergänzend zu den vordefinierten Ausführungsmetadaten berücksichtigt werden.

6.2 Tuning-Richtlinien

Policies (oder auch *Tuning-Richtlinien* bzw. *Richtlinien*) bezeichnen allgemein Regeln, die Abläufe beeinflussen. Je nach der Abstraktionsstufe wird oftmals zwischen *high-level* und *low-level* Policies unterschieden [MS93]. High-level Policies sind allgemeiner gefasste Richtlinien, die grobe, systemweite Ziele wie beispielsweise *„Datenverlust durch Feuer oder defekte Datenträger vermeiden“* bzw. *„Antwortzeit des Systems verbessern“* beschreiben. Bei den deutlich spezifischeren low-level Policies handelt es sich dagegen um konkrete Handlungsanweisungen, die im Wesentlichen angeben, *wie* die abstrakteren, durch die high-level Policies vorgegebenen Ziele zu erreichen sind (z. B. *„Führe jeden Sonntag 22:00 Uhr das Backup-Skript aus“*) [Rei08].

Um ihre Erfüllung zu ermöglichen, müssen grundsätzlich die mittels der high-level Policies festgelegten Ziele zunächst 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 [MS93, Rei08]. Im Kontext des autonomen Datenbank-Tuning haben wir eine Reihe von Policy-Sprachen untersucht (*Abschnitt 6.2.1*). Dabei handelt es sich ausschließlich um low-level Policies, die sich recht nah an der Seite der technischen Umsetzung befinden.

Für autonom agierende Systeme bieten Policies eine Schnittstelle für Nutzer, um Zielvorgaben zu spezifizieren und damit das Verhalten des Systems zu beeinflussen. Zur Integration und Berücksichtigung von Policies im *Autonomic Tuning Expert* wurde die ATE-Architektur um eine spezielle Planungskomponente namens *Tuning Coordinator* erweitert, wodurch der Autonomie-Grad des ATE gesteigert werden konnte.

Der *Tuning Coordinator* interagiert zur Laufzeit mit anderen ATE-Komponenten und ist unter anderem verantwortlich für:

- **Pruning von Tuning-Plänen.** Dabei handelt es sich um die Aktivierung bzw. Deaktivierung von Tuning-Plänen und entsprechenden Events bei der Initialisierung sowie zur Laufzeit, um den Suchraum des *Tuning Plan Selectors* einzuschränken.

- **Auswahl von Tuning-Plänen.** Der *Tuning Coordinator* ermöglicht die Identifikation von Tuning-Plänen, die zur Auflösung des erkannten Problems unter Berücksichtigung der durch den Administrator mittels Tuning-Richtlinien vorgegebenen Nebenbedingungen beitragen.

- **Eingriff in die Tuning-Plan-Ausführung.** Während ihrer Ausführung können die Tuning-Pläne unter Berücksichtigung der Vorgaben des Administrators automatisch modifiziert werden.

Die Steuerung des *Tuning Coordinators* erfolgt dabei durch die vom DBA vorzugebenden Policies. Nachfolgend erfolgt eine kurze Vorstellung des Policy-Modells sowie einer entsprechenden darauf aufbauenden Policy-Beschreibungssprache.

6.2.1 Evaluation existierender Policy-Sprachen und -Modelle

In [Rei08] wurde eine Reihe von bekannten Policy-Beschreibungssprachen sowie entsprechenden -Modellen untersucht und bezüglich ihrer Anwendbarekt im Kontext des ATE bewertet. Zu den untersuchten Policy-Sprachen gehören *Ponder* [DDLS01], *WS-Agreement* [ACD+07], *Web Service Level Agreement* [LKD+03], *CIM Simplified Policy Language* [DMT07], *eXtensible Access Control Markup Language* [OAS05], *DB2 Maintenance Policies* [IBM08a] und *Policy Core Information Model* [MESW01].

In den untersuchten Sprachen wird der Begriff Policy sehr weit gefasst. Zu den wichtigsten Policy-Arten zählen dabei nach [KW04] folgende:

Authorization Policies. Richtlinien zur Umsetzung der Zugriffskontrolle auf Services bzw. Ressourcen.

Goal Policies. Ausdrücke zur Beschreibung des Zielzustands des Systems bzw. der Systemkomponenten (*was* erreicht werden soll).

Action Policies. Ausdrücke der Form *„Wenn Bedingung, Dann Aktion“*, die im System hinterlegt werden. Bei Erfüllung vordefinierter Bedingungen werden dann zur Laufzeit die entsprechenden Aktionen ausgeführt. Im Gegensatz zu den *Goal Policies* spezifizieren die *Action Policies* somit, *wie* der Wunschzustand erreicht werden soll.

Zwar lassen einige der untersuchten Sprachen sehr große Spielräume für eigene Erweiterungen zu, jedoch sind sie auf bestimmte Anwendungsbereiche viel zu sehr spezialisiert und können daher im Kontext des autonomen Datenbank-Tuning nicht eingesetzt werden. Aus diesem Grund haben wir uns für die Konzeption und Entwicklung einer eigenen, auf den Einsatz im Kontext des *Autonomic Tuning Expert* zugeschnittenen Sprache zur Beschreibung von Tuning-Richtlinien (*Abschnitt 6.2.3*) sowie eines entsprechenden, zugrunde liegenden Modells entschieden (*Abschnitt 6.2.2*).

6.2.2 Ein Modell zur Definition und Anwendung von Tuning-Richtlinien

Wie bereits erwähnt, ist eine Sprache, die den Administrator in die Lage versetzt, einem autonom agierenden Tuning-System seine Präferenzen bezüglich des zu erfolgenden Tuning zugänglich zu machen, von einer sehr großen Bedeutung. In [Rei08] wurde ein Anforderungskatalog für die zu entwickelnde Sprache zur Definition von Tuning-Richtlinien aufgestellt. In Anlehnung an [Rab09] erfolgt nachfolgend eine kurze Zusammenfassung der dabei identifizierten Anforderungen:

- Definition von Zeitfenstern für die Gültigkeit einzelner Richtlinien bzw. für den Einsatz bestimmter Tuning-Pläne (Tuning-Plan-Klassen, *Abschnitt 6.1.2*). Zu solchen Zeitfenstern zählen unter anderem Wartungszeitfenster bzw. Zeitfenster für bestimmte Anwendungen wie die Gehaltsabrechnungsanwendung.
- Bereitstellung von Zusatzinformationen über das zu tunende System bzw. seine Systemumgebung für den *Autonomic Manager*.
- Definition von Ressourcenrestriktionen, wie beispielsweise Angabe oberer bzw. unterer Schranken für die zulässige Allokation einzelner Ressourcen oder Angabe von Verboten bezüglich der Erstellung, Löschung bzw. Rekonfiguration bestimmter Datenbankobjekte.
- Priorisierung von Datenbankobjekten und Anwendungen.
- Definition und Priorisierung von Tuning-Zielen.

Zur Abdeckung der oben zusammengefassten Anforderungen wurden vier Arten von Tuning-Richtlinien entwickelt. Dabei handelt es sich um *Direction Policies*, *Information Policies*, *Priority Policies* und *Constraint Policies*, die nachfolgend erklärt werden.

Direction Policies. Mit Hilfe von *Direction Policies* wird eine unscharfe Vorgabe von Tuning-Zielen ermöglicht, die das System erreichen sollte. Im Gegensatz zu den *Goal Policies*, die für ihre Umsetzung einen tiefen Einblick in die Systeminterna und mathematische Modelle zur Erfassung von Abhängigkeiten unter den Sensoren und Effektoren voraussetzen, was unserem universellen, black-box-orientierten Ansatz widerspricht, spezifiziert diese Art von Policies keinen Zielzustand des Systems, sondern gibt vielmehr die Richtung für das Tuning vor. Beispiele für solche Vorgaben der Tuning-Richtung sind *„Nebenläufigkeit steigern“*, *„Bufferpool-Performance steigern“*, *„Sortiervorgänge optimieren“*, *„Durchsatz erhöhen“* bzw. *„Antwortzeit reduzieren“*.

Information Policies. Zur Bereitstellung von Zusatzinformationen über das zu tunende System selbst sowie seine Umgebung, die vom ATE ausgewertet und sowohl beim reaktiven als auch proaktiven Tuning berücksichtigt werden können, wurden *Information Policies* entwickelt. Zu solchen, vom Administrator bereitgestellten Informationen zählen beispielsweise Informationen über die bekannten Workload-Veränderungen (z. B. *„analytische Datenverarbeitung mit ca. 30-50 Nutzern jeden ersten Montag des Monats“*), Anzahl von parallelen Nutzern zu bestimmten Zeiten, Zugriffsprofile von bestimmten Anwendungsgruppen etc. Zwar können solche Informationen in der Regel automatisch erlernt werden, dies ist jedoch mit entsprechenden Verzögerungen

verbunden[27]. Das Vorhandensein dieses Wissens im Voraus ermöglicht außerdem ein proaktives Tuning. So kann das System beispielsweise zu gewünschten Zeiten proaktiv für den OLAP- bzw. OLTP-Betrieb optimiert werden.

Priority Policies. *Priority Policies* dienen dazu, einzelne Tuning-Objekte mit Prioritätsangaben zu versehen. Da Administratoren typischerweise in der Lage sind, die im Sinne des Unternehmens wichtigsten Ressourcen (wie bestimmte Bufferpools, Tablespaces oder ähnliches) bzw. Anwendungen (z. B. Fließbandsteuerungsanwendung) zu identifizieren, lassen sich diese Informationen mit Hilfe der *Priority Policies* dem ATE zur Verfügung stellen und können somit im Tuning-Prozess beispielsweise bei der Speicherverteilung oder ähnlichem entsprechend berücksichtigt werden.

Constraint Policies. Um einzelne Ressourcen mit Beschränkungen bezüglich ihrer Allokation versehen zu können bzw. die Erzeugung, Löschung oder Veränderung von bestimmten Datenbankobjekten wie beispielsweise Indizes zu verbieten, wurden *Constraint Policies* eingeführt. Damit lassen sich beispielsweise Über- bzw. Unterallokationen von Ressourcen verhindern. Zu typischen Beispielen für diese Art der Tuning-Richtlinien zählen *„Sortheap-Größe > 2.000 Seiten“*, *„Größe des Bufferpools IBMDEFAULTBP1 zwischen 10.000 Seiten und 100.000 Seiten“*, *„Verbot der Löschung jeglicher Indizes von der Tabelle Steuerung“*.

Des Weiteren bieten alle der oben erwähnten Policy-Arten einen Mechanismus zur Definition von Zeitfenstern an, in denen sie gültig sind. Außerhalb dieser Zeiträume sind die durch die Policies definierten Nebenbedingungen ungültig und werden daher vom ATE ignoriert. Bei der Definition der Zeitfenster lässt sich zwischen *globalen* und *lokalen Zeitfenstern* unterscheiden. Während ein lokales Zeitfenster für eine bestimmte Richtlinie definiert wird und nicht wiederverwendbar ist, kann auf global definierte und mittels eines eindeutigen Namens identifizierte Zeitfenster von mehreren Tuning-Richtlinien zugegriffen werden.

Eingangs wurde der Unterschied zwischen high-level und low-level Policies beschrieben. Bei den hier eingeführten Policy-Arten handelt es sich offenbar um eine Mischform: Zwar kann damit durch die Definition der Tuning-Richtung in einem gewissen Umfang angegeben werden, was erreicht werden soll, jedoch definiert man dadurch keinen Zielzustand des Systems, der dann im Rahmen des Tuning erreicht wird. Somit sind diese Policies abstrakter als typische low-level Policies, aber nicht abstrakt genug für high-level Policies.

6.2.3 Vorschlag einer XML-basierten Sprache zur Definition von Tuning-Richtlinien

In diesem Abschnitt wird in Anlehnung an [Rab09] ein kurzer Einblick in die Syntax der in [Rei08] entwickelten XML-basierten Sprache zur Definition von Tuning-Richtlinien gegeben. Auf eine vollständige Darstellung der Sprachsyntax wurde dabei verzichtet. Für weiterführende Informationen dazu sei auf [Rei08] verwiesen.

Die beim Tuning durch den ATE zu berücksichtigenden Tuning-Richtlinien werden zunächst in einer entsprechenden XML-Datei definiert, die anschließend von dem *Tuning*

[27] Bei der Erkennung von Workload-Shifts durch den ATE kommt es beispielsweise aktuell zu Verzögerungen von ca. $3 - 5$ Minuten.

Coordinator eingelesen wird. Sollen zur Laufzeit des ATE Änderungen an den festgelegten Tuning-Richtlinien vorgenommen bzw. neue Tuning-Richtlinien definiert werden, so ist in der aktuellen Implementierung die entsprechende XML-Datei anzupassen und vom *Tuning Coordinator* erneut einlesen zu lassen.

In ***Listing 6.1*** findet sich ein Beispiel einer solchen XML-Datei. Der grobe Aufbau dieser Datei ist dabei wie folgt: Nach einem entsprechenden XML-Header mit Namensraum- sowie Schemadefinitionen finden sich XML-Elemente zur Definition globaler Zeitfenster (`TimePeriods`), der *Direction Policies* (`Directions`), der *Information Policies* (`Knowledge`), der *Priority Policies* (`Priorities`) sowie der *Constraint Policies* (`Constraints`).

Listing 6.1: Definition von Tuning-Richtlinien – ein Beispiel (in Anlehnung an [Rab09])

```
1  <?xml version="1.0" encoding="UTF-8"?>
2
3  <Policies
4    xmlns="http://www.minet.uni-jena.de/dbis/ate"
5    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
6    xsi:schemaLocation="http://www.minet.uni-jena.de/dbis/
7                        ate-policy.xsd">
8
9    <!-- Definition globaler Zeitfenster  -->
10   <TimePeriods>
11     <TimePeriod name="workingHours">
12       <DayOfWeek>Mon Tue Wed Thu Fri Sat</DayOfWeek>
13       <TimeOfDay>T080000/T180000</TimeOfDay>
14     </TimePeriod>
15     <TimePeriod name="maintenanceHours">
16       <DayOfWeek>Mon Tue Wed Thu Fri Sun</DayOfWeek>
17       <TimeOfDay>T220000/T040000</TimeOfDay>
18     </TimePeriod>
19   </TimePeriods>
20
21   <!-- Priorisierte Tuning-Ziele -->
22   <Directions>
23     <DefaultQualifier>
24       <Instance>NODE0001</Instance>
25       <Database>DATA</Database>
26     </DefaultQualifier>
27     <Direction name="concurrency" priority="1">
28       <Validity timePeriodName="workingHours"/>
29       <Qualifier>
30         <Instance>NODE0002</Instance>
31         <Database>SHOP</Database>
32       </Qualifier>
33     </Direction>
34     <Direction name="bufferpool" priority="3">
35       <ValidityDef>
36         <DayOfWeek>Mon Tue Wed Thu Fri</DayOfWeek>
37         <TimeOfDay>T080000/T120000</TimeOfDay>
38       </ValidityDef>
39     </Direction>
40   </Directions>
41
42   <!-- Information ueber die Umgebung des Systems -->
43   <Knowledge>
44     <Information xsi:type="WorkloadInformation">
```

```
45        <Validity timePeriodName="workingHours"/>
46        <WorkloadClass>OLTP</WorkloadClass>
47      </Information>
48      <Information xsi:type="WorkloadInformation">
49        <Validity timePeriodName="workingHours"/>
50        <NumParallelUsers>100</NumParallelUsers>
51      </Information>
52      <Information xsi:type="TimeSlot">
53        <Validity timePeriodName="maintenanceHours"/>
54        <TuningPlanClass>maintenance</TuningPlanClass>
55      </Information>
56    </Knowledge>
57
58    <!-- Prioritaeten fuer Datenbank-Ressourcen
59          und Anwendungen -->
60    <Priorities>
61      <Priority value="1">
62        <Validity timePeriodName="workingHours"/>
63        <Qualifier xsi:type="TableSpaceQualifier">
64          <Instance>NODE0001</Instance>
65          <Database>DATA</Database>
66          <TableSpace>TBS2</TableSpace>
67        </Qualifier>
68      </Priority>
69    </Priorities>
70
71    <!-- Ressourcen-Beschraenkungen -->
72    <Constraints>
73      <Constraint xsi:type="SizingConstraint">
74        <Validity timePeriodName="workingHours"/>
75        <BufferpoolQualifier>
76          <Instance>NODE0001</Instance>
77          <Database>DATA</Database>
78          <Bufferpool>BUFFPOOL2</Bufferpool>
79        </BufferpoolQualifier>
80        <Expression predicate="GreaterEqual" value="10000"/>
81      </Constraint>
82      <Constraint xsi:type="ExecutionConstraint" mode="create">
83      <!-- keine Indizes fuer die Tabelle myOrders
84            der Datenbank DATA erstellbar -->
85        <IndexTypeQualifier>
86          <Instance>NODE0001</Instance>
87          <Database>DATA</Database>
88          <Schema></Schema>
89          <!-- ein leeres Schema steht fuer alle Schemas -->
90          <Table>myOrders</Table>
91        </IndexTypeQualifier>
92      </Constraint>
93    </Constraints>
94
95  </Policies>
```

Definition globaler bzw. lokaler Zeitfenster. Globale Zeitfenster werden unterhalb des XML-Elements `TimePeriods` geschachtelt definiert (Zeilen 10 – 19) und können anschließend innerhalb der Policy-Datei mehrfach verwendet werden. Dazu wird in die entsprechende Policy das XML-Element `Validity` aufgenommen, welches sich auf den Namen eines globalen Zeitfensters bezieht (Zeile 28). Zur Angabe eines

lokalen, in anderen Policies nicht wiederverwendbaren Zeitfenster wird das XML-Element `ValidityDef` mit einer entsprechenden Zeitfenster-Definition verwendet (Zeilen 35 – 38). Die Syntax der Zeitfensterdefinition basiert hauptsächlich auf dem *Policy Core Information Model* [MESW01] und wurde in Anlehnung an die *DB2 Maintenance Policies* [IBM08a] um einige syntaktische Elemente wie `DayOfWeek`, `DayOfMonth` bzw. `MonthOfYear` erweitert. In den Zeilen 11 – 14 erfolgt beispielsweise die Angabe eines globalen Zeitfensters namens *WorkingHours*, welches sich auf die Wochentage Montag bis Samstag, jeweils 8 – 18 Uhr bezieht. Verzichtet man bei der Definition eines Zeitfensters auf die Angabe einzelner Elemente, so erweitert sich das Zeitfenster über alle Werte dieses Elements. Beim Weglassen des Elements `DayOfWeek` würde sich beispielsweise das Zeitfenster über alle Wochentage erstrecken.

Definition der Ressourcen-Qualifier. Einzelne Tuning-Richtlinien können sich auf spezifische Datenbank-Ressourcen beziehen. Zur Definition der Zuordnung müssen diese Ressourcen daher eindeutig identifiziert werden. Dies wird mittels impliziter (Zeilen 23 – 26) bzw. expliziter (Zeilen 29 – 32) Qualifier umgesetzt. Ähnlich wie bei der Definition der Zeitfenster kann auch hier auf die Angabe einzelner Qualifier-Elemente verzichtet werden. In diesem Fall erfüllen Objekte mit beliebigen Werten des weggelassen Qualifier-Elements die Qualifier-Definition. In Zeilen 85 – 91 erfolgt beispielsweise die Definition des `IndexTypeQualifier`, welcher alle Tabellen namens *myOrders*, die sich in beliebigen Schemas der Datenbank *DATA*, auf der Instanz *NODE0001* befinden, identifiziert.

Definition der Direction Policies. Die Angaben bezüglich der Tuning-Richtung (*Direction Policies*) erfolgt mittels des XML-Elements `Direction`, welches unterhalb des Elements `Directions` eingeordnet wird. In Zeilen 22 – 40 erfolgt die Definition von *Direction Policies* zum Tunen der Nebenläufigkeit („concurrency“, Zeilen 27 – 33) sowie des Bufferpools („bufferpool“, Zeilen 34 – 39). Dabei lassen sich die einzelnen Richtlinien mit Prioritäten versehen, die zur Konfliktauflösung beispielsweise bei sich zeitlich überlappenden Policies berücksichtigt werden können. *Direction Policies* können für unterschiedliche Datenbanken angegeben werden, es erfolgt daher eine Zuordnung der Policies zu den entsprechenden Ressourcen mittels Qualifier.

Definition der Information Policies. Weiterführende Informationen, wie beispielsweise das Wissen über die anliegende Workload bzw. andere Charakteristika der Systemumgebung, lassen sich mittels der XML-Elemente `Information`, die unterhalb des Elements `Knowledge` geschachtelt werden, angeben. Aktuell werden Workload-spezifische Angaben (Zeilen 44 – 51) bezüglich des Typs der anliegenden Workload (`WorkloadClass`, Zeile 46) und bezüglich der Anzahl der am System parallel arbeitenden Nutzer (`NumParallelUsers`, Zeile 50) sowie die Angabe von für die Ausführung bestimmter Klassen von Tuning-Plänen (`TuningPlanClass`, Zeile 54) definierten Zeitfenstern (`TimeSlot`, Zeilen 52 – 55) unterstützt.

Definition der Priority Policies. Die Angabe von *Priority Policies* erfolgt mittels der XML-Elemente namens `Priority`. Dabei werden Datenbankobjekten bzw. Anwendungsprogrammen, die mittels Qualifier eindeutig identifiziert sind, entsprechende Prioritäten und Gültigkeitszeiträume für die Prioritätsangaben zugewiesen. Während des Tuning können Prioritätsangaben entsprechend berücksichtigt werden, indem beispielsweise Tuning-Pläne, die zu einer Performance-Steigerung der hochpriorisierten Datenbank-Objekte beitragen, bevorzugt ausgeführt werden. In Zeilen

		Einsatz im ATE		
		Überwachung/ Analyse	**Tuning-Plan-Auswahl**	**Tuning-Plan-Ausführung**
Policy-Art	**Direction Policies**	+	+	−
	Information Policies	+	−	−
	Priority Policies	−	+	−
	Constraint Policies	+	+	+

Tabelle 6.1: Anwendung von Tuning-Richtlinien im ATE

61 – 68 wird beispielsweise definiert, dass der Tablespace *TBS2* der Datenbank *DATA* auf der Instanz *NODE0001* während des Zeitfensters `workingHours` die Priorität 1 hat.

Definition der Constraint Policies. Zur Angabe von *Constraint Policies* dient das Element `Constraint` (Zeilen 72 – 93). Durch die Verwendung von `SizingConstraint` lassen sich Beschränkungen bezüglich der Ressourcenallokation festlegen. Die Identifikation von betroffenen Ressourcen erfolgt dabei mittels des Qualifier-Mechanismus. Die eigentliche Allokationsbeschränkung wird mittels des XML-Elements `Expression` umgesetzt (Zeile 80). Bei der Definition des entsprechenden Ausdrucks werden Prädikate „Greater“, „GreaterEqual“, „Less“, „LessEqual“ sowie „Equal“ unterstützt. In Zeilen 73 – 81 erfolgt die Angabe eines *Sizing Constraints* für den der Datenbank *DATA* (Instanz *NODE0001*) zugewiesenen Bufferpool namens *BUFFERPOOL2*, dessen Größe den Wert von 10.000 Seiten nicht unterschreiten darf.

Die sogenannten Ausführungsbeschränkungen ermöglichen die Umsetzung eines Verbots der Erstellung, Modifikation bzw. Löschung bestimmter Datenbankobjekte (`ExecutionConstraints`, Zeilen 82 – 92). Durch das Attribut `mode` wird dabei angegeben, ob die Erstellung (`create`), Modifikation (`modify`) oder Löschung (`delete`) verboten werden sollen. Die Identifikation der entsprechenden Objekte, auf die sich der Verbot erstreckt, erfolgt wieder mittels des bereits bekannten Qualifier-Mechanismus. In Zeilen 82 – 92 erfolgt beispielsweise die Angabe eines *Execution Constraints*, welcher die Erzeugung von Indizes für alle Tabellen namens *myOrders*, die sich in verschiedenen Datenbankschemas der Datenbank *DATA* befinden können, verhindern soll.

6.2.4 Anwendung von Tuning-Richtlinien im autonomen Datenbank-Tuning

In den nachfolgenden Abschnitten wird gezeigt, wie sich die einzelnen Policy-Arten im Rahmen des autonomen Datenbank-Tuning mit *Autonomic Tuning Expert* einsetzen lassen. Einen Überblick darüber, welche Policy-Arten welche Phasen des MAPE-Kreislaufs unterstützen, bietet dabei ***Tabelle 6.1***.

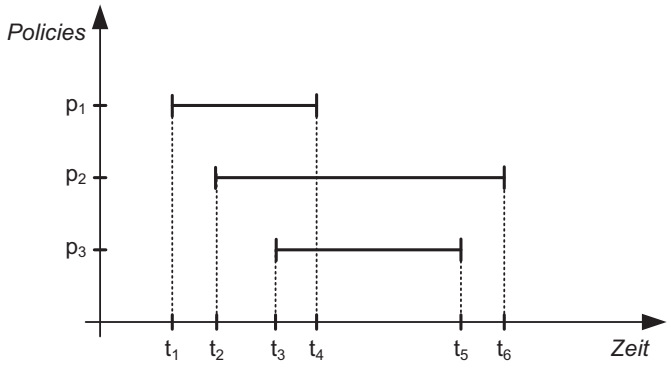


Abbildung 6.3: Überprüfung der aktuell gültigen Zeitfenster

6.2.4.1 Einfluss von Tuning-Richtlinien auf die Überwachung und Analyse

Die in vorangegangenen Abschnitten vorgestellten Tuning-Richtlinien lassen sich beispielsweise zur Reduktion des Overheads bei Monitoring, Analyse und Planung durch ein kontextabhängiges Deaktivieren von nicht für die Auswahl sowie Ausführung in Frage kommenden Tuning-Plänen mit entsprechenden Events eingesetzt werden. Dieses Vorgehen bezeichnen wir als *Pruning*. Liegen die Gründe für die Deaktivierung der Tuning-Pläne samt ihrer Events nicht mehr vor, so müssen die deaktivierten Tuning-Pläne wieder aktiviert werden. Eine Überprüfung und gegebenenfalls Anpassung der Liste der aktiven Tuning-Pläne und ihrer Events (Aktivierung/Deaktivierung) erfolgt dabei in folgenden Fällen:

- **Wechsel des Typs der anliegenden Workload.** Dabei können sowohl die durch den *Workload Classifier* erkannten als auch die durch die Administratoren mittels *Information Policies* angekündigten Workload-Typ-Wechsel berücksichtigt werden. Erfolgt beispielsweise ein Wechsel von der OLTP-Workload zur OLAP-Workload, so können alle OLTP-spezifischen Tuning-Pläne deaktiviert (da sie bei falschem Workload-Typ sowieso keine Performance-Verbesserungen bringen würden) und stattdessen die OLAP-spezifischen Tuning-Pläne aktiviert werden.

- **Aktivierung eines neuen bzw. Deaktivierung eines aktuell gültigen Zeitfensters.** Mit jeder neu in die Liste der aktiven Policies aufgenommenen bzw. aus dieser Liste entfernten Policy muss überprüft werden, welche Nebenbedingungen aktuell gültig sind. Wie in ***Abbildung 6.3*** veranschaulicht, würde diese Überprüfung zu Zeitpunkten t_1, t_2, ..., t_6 stattfinden. In Abhängigkeit von den in den jeweiligen Zeitabschnitten gültigen Nebenbedingungen erfolgt anschließend die Deaktivierung der diesen Nebenbedingungen widersprechenden bzw. Aktivierung der diese Nebenbedingungen erfüllenden Tuning-Pläne sowie der mit ihnen verknüpften Events.

Initiiert durch die Policies lassen sich nicht nur die Aktivierung bzw. Deaktivierung von Tuning-Plänen mit zugehörigen Events, sondern insbesondere auch Modifikationen der Events bzw. der Tuning-Pläne vornehmen. So können beispielsweise atomare Events, Korrelationsmuster oder einzelne Schritte der Tuning-Pläne entsprechend der mittels *Information Policies* angegebenen Workload angepasst werden.

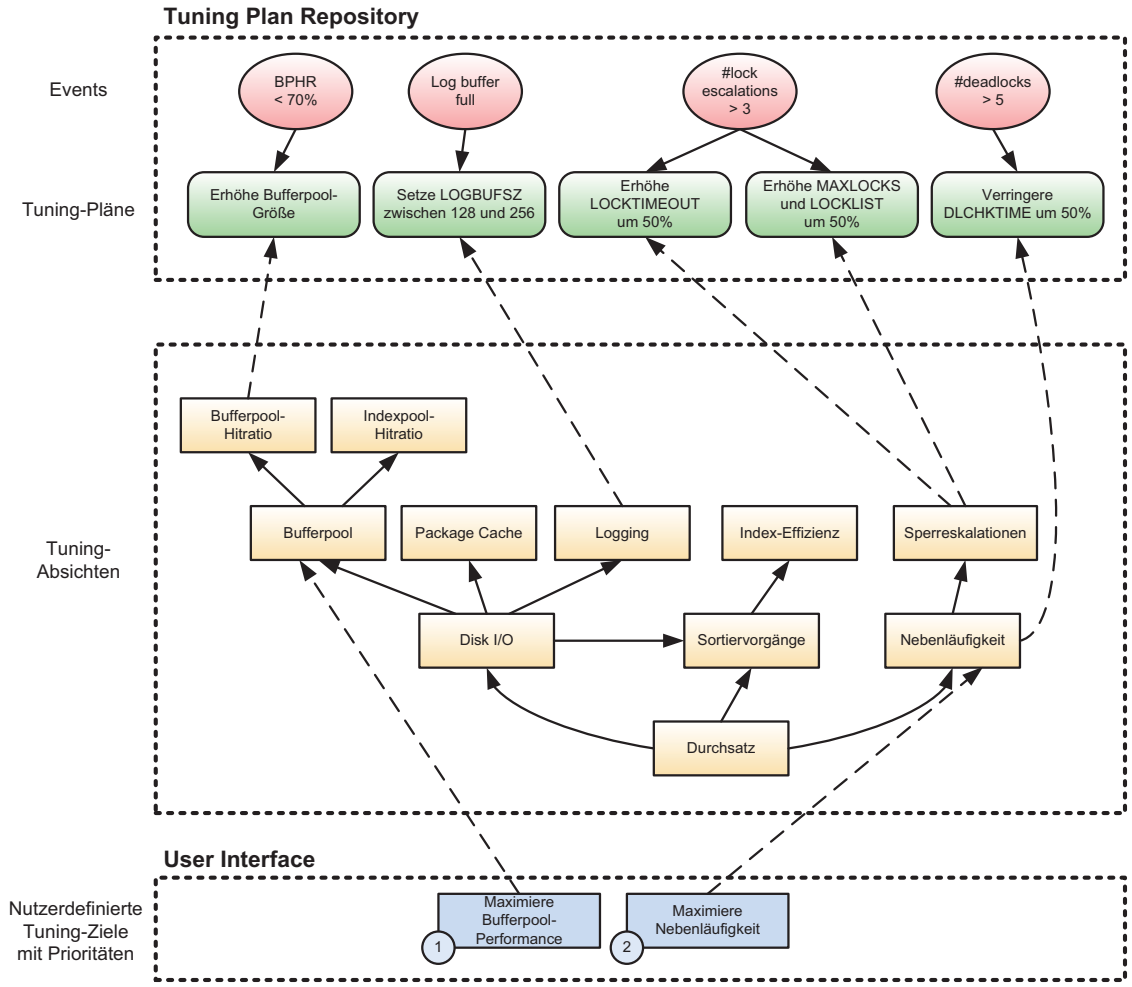


Abbildung 6.4: Einfluss von *Direction Policies* auf die Monitoring- bzw. Analyse-Phase

In ***Abbildung 6.4*** wird exemplarisch veranschaulicht, wie mittels *Direction Policies* die Aktivierung bzw. Deaktivierung von Tuning-Plänen und zugehörigen Events vorgenommen werden kann. Wie in *Abschnitt 6.1.2* erläutert, werden einzelne Tuning-Pläne mit Tuning-Absichten annotiert. Auch die vom Administrator vorgegebenen und priorisierten Tuning-Ziele werden zunächst automatisch mit entsprechenden Elementen des Tuning-Absichten-Graphen verknüpft.

Durch die Traversierung des Tuning-Absichten-Graphen erfolgt nun eine schrittweise Abbildung der abstrakten Tuning-Richtlinien auf spezifische, diesen Tuning-Richtlinien entsprechende Tuning-Pläne. Dabei gilt, dass alle Tuning-Pläne, die bei der Traversierung des Tuning-Absichten-Graphen angefangen von einer Zielvorgabe erreichbar sind, aktiviert werden müssen, da sie zum Erreichen dieses Ziels auf irgendeine Art beitragen können. Alle anderen Tuning-Pläne mit ihren zugehörigen Events können dagegen deaktiviert werden, wodurch sowohl der Overhead des Monitorings gesenkt als auch der Suchraum bei der Tuning-Plan-Auswahl in der Planungsphase reduziert werden.

6.2.4.2 Einfluss von Tuning-Richtlinien auf die Auswahl von Tuning-Plänen

Basierend auf dem im vorangegangenen *Abschnitt 6.2.4.1* eingeführten Tuning-Absichten-Modell lässt sich auch die Tuning-Plan-Auswahl umsetzen. Liegen mehrere Tuning-Pläne vor, die dem gleichen Event zugeordnet sind, so kann der Tuning-Absichten-Graph von diesem Event angefangen bis hin zu den abstrakten priorisierten Zielvorgaben traversiert werden. Aus den Prioritäten dieser Zielvorgaben lässt sich indirekt die Priorisierung der Tuning-Pläne ableiten (Tuning-Pläne, die von einem höher priorisierten Tuning-Ziel aus erreichbar sind, haben eine entsprechend höhere Priorität). Die somit ermittelten Prioritäten der Tuning-Pläne lassen sich nun neben den diesen Tuning-Plänen zugeordneten Metadaten bei der Tuning-Plan-Auswahl berücksichtigen.

Liegen mittels *Priority Policies* spezifizierte Prioritätsangaben für einzelne Datenbankobjekte vor, so lassen sich bei der Tuning-Plan-Auswahl diejenigen Tuning-Pläne bevorzugen, die die hoch priorisierten Objekte optimieren.

Auch die *Constraint Policies* lassen sich in bestimmten Fällen zur Steuerung der Tuning-Plan-Auswahl einsetzen. Konzentrieren sich Tuning-Pläne auf das Tuning von Datenbankobjekten, deren Konfiguration laut *Constraint Policies* nicht verändert werden darf und wurden diese Tuning-Pläne in der Pruning-Phase nicht deaktiviert (vgl. *Abschnitt 6.2.4.1*), so können sie bei der Auswahl der auszuführenden Tuning-Pläne unberücksichtigt bleiben.

6.2.4.3 Einfluss von Tuning-Richtlinien auf die Ausführung von Tuning-Plänen

Constraint Policies finden auch bei der Ausführung von Tuning-Plänen Anwendung. Ermittelt ein Tuning-Plan erst zur Laufzeit, welche Datenbankobjekte und wie optimiert werden müssen, so besteht keine Möglichkeit, diesen Tuning-Plan bei einer Verletzung von aktiven *Constraint Policies* zu deaktivieren. Versucht nun ein Tuning-Schritt Änderungen an Datenbankobjekten vorzunehmen, die durch die *Constraint Policies* als unzulässig gekennzeichnet sind, so lassen sich diese Tuning-Schritte zurückweisen ohne die Konfiguration betroffener Ressourcen zu verändern.

6.2.5 Erweiterung der ATE-Architektur zur Berücksichtigung von Tuning-Richtlinien

Zur Ermöglichung einer Berücksichtigung der in vorangegangenen Abschnitten vorgestellten Tuning-Richtlinien im Rahmen des Datenbank-Tuning durch *Autonomic Tuning Expert* wurde eine neue Komponente namens *Tuning Coordinator* entwickelt und in die bestehende ATE-Architektur integriert (***Abbildung 6.5***). Der *Tuning Coordinator* ist einerseits für das Einlesen und Verwalten von Policies verantwortlich. Andererseits stellt er sicher, dass aktive Policies durch andere Komponenten berücksichtigt werden und damit den Tuning-Prozess beeinflussen.

So erfolgt beispielsweise bei Bedarf eine Rekonfiguration des *Performance Expert*, indem die Definition einzelner Exceptions angepasst bzw. ganze Sätze von Exceptions Workload-abhängig umgeschaltet werden (vgl. *Abschnitt 5.2.6*). Auch die Korrelationsmuster des

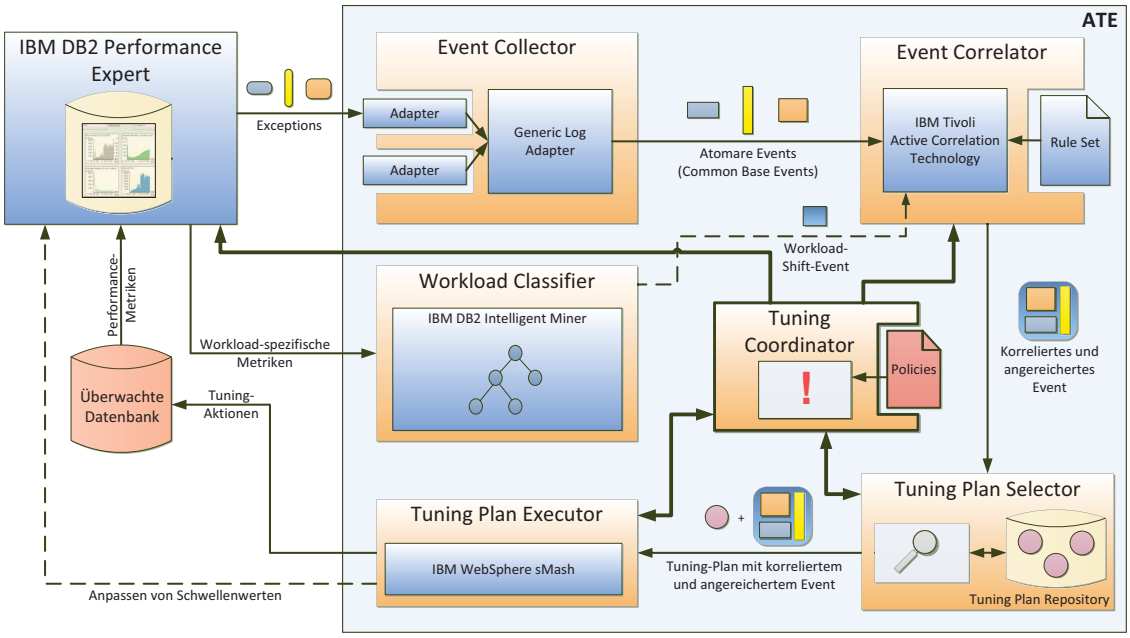


Abbildung 6.5: ATE-Architektur, erweitert um *Tuning Coordinator*

Event Correlators lassen sich durch den *Tuning Coordinator* modifizieren bzw. deaktivieren und bei Bedarf wieder aktivieren (vgl. *Abschnitt 6.2.4.1*).

Dem *Tuning Plan Selector* werden entsprechende Methoden zur Verfügung gestellt, um Tuning-Pläne zu identifizieren, die im Konflikt zu aktiven Policies stehen und daher nicht ausgeführt werden dürfen. Durch die Bereitstellung von Methoden zum Zugriff auf das Tuning-Absichten-Modell und interne Policy-erfassende Metadaten wird weiterhin die Suche nach Tuning-Plänen unter Berücksichtigung von aktiven Policies unterstützt (vgl. *Abschnitt 6.2.4.2*).

Mit Hilfe der eigens für den *Tuning Plan Executor* bereitgestellten Methoden kann überprüft werden, ob einzelne Tuning-Schritte den durch die aktiven Policies definierten Nebenbedingungen widersprechen. Durch eine entsprechende Anpassung der mit *WebSphere sMash* umgesetzten Tuning-Aktivitäten lässt sich beispielsweise vor der Ausführung ihrer Instanzen, sprich Tuning-Schritte, feststellen, ob die durch sie vorgenommenen Konfigurationsänderungen zulässig sind oder nicht. Bei Tuning-Schritten, die eine Reallokation von Ressourcen vornehmen und dabei die mittels der *Constraint Policies* spezifizierten Oberschranken bzw. Unterschranken überschreiten bzw. unterschreiten, besteht die Möglichkeit einer automatischen Anpassung dieser Schritte, um die Rekonfiguration betroffener Ressourcen mit einer kleineren, die Policies nicht verletzenden Schrittweite vorzunehmen (vgl. *Abschnitt 6.2.4.3*).

Auf die Darstellung der Ergebnisse der Testläufe des um die soeben vorgestellten Funktionalitäten erweiterten Prototyps wurde an dieser Stelle aus Platzgründen verzichtet. Für weiterführende Informationen sei daher auf [Rei08] verwiesen.

6.3 Hypothesenbäume

Während Metadaten sowie Tuning-Richtlinien unter anderem dazu verwendet werden können, die Planungsphase indirekt zu beeinflussen, bietet der nachfolgend vorgestellte Ansatz dem Administrator die Möglichkeit einer unmittelbaren Einflussnahme auf die Planung durch die Modellierung des gesamten Planungsablaufs an. Dabei wird auf das Konzept der sogenannten *Hypothesenbäume* [Are09] zurückgegriffen. In Anlehnung an die Fehlerbäume [Thu04, Eri99], die eine Analyse möglicher Ursachen für aufgetretene Probleme ermöglichen, fassen auch die Hypothesenbäume mögliche Problemursachen der durch die Wurzelknoten der Bäume repräsentierten Probleme zusammen. Einen wichtigen Bestandteil der Hypothesenbäume bilden jedoch neben den Konstrukten, die eine Problemdiagnose unterstützen, die Aktionsknoten, die das durch eine Traversierung des Hypothesenbaums eingegrenzte Problem zu beheben versuchen. Bei den Aktionsknoten handelt es sich um Blätter der Hypothesenbäume. Sie bestehen aus den eigentlichen Tuning-Plänen, die im Anschluss an die Planung ausgeführt werden.

Durch eine spezielle Ausführungslogik wird in diesem Ansatz sichergestellt, dass nicht nur die Reihenfolge, in der die Hypothesenbaumknoten abgearbeitet werden (im Folgenden als *Traversierungsreihenfolge* bezeichnet), zur Laufzeit bestimmt werden kann, sondern weiterhin sowohl einzelne Teilbäume als auch der gesamte Hypothesenbaum bei Bedarf mehrfach ausgeführt werden können. Wird beispielsweise bei der Traversierung eines Hypothesenbaums bzw. nach der Ausführung eines Tuning-Plans festgestellt, dass die getroffene Annahme nicht bestätigt werden konnte bzw. dass der Tuning-Plan nicht zur erwarteten Systemzustandsverbesserung beigetragen hat, so kann mittels spezieller Backtracking-Schritte zurück im Hypothesenbaum gegangen und ein alternativer Pfad gewählt werden.

Die durch einen fälschlicherweise ausgeführten Tuning-Plan vorgenommenen Änderungen lassen sich dabei in einigen Fällen kompensieren. Zur Ermöglichung der Tuning-Plan-Kompensation müssen die entsprechenden Kompensationsschritte vorliegen. Diese können unter Umständen aus den Tuning-Plan-Schritten automatisch abgeleitet bzw. bereits bei der Erstellung von Tuning-Plänen durch die DBA bereitgestellt werden. Von weiteren Betrachtungen der Tuning-Plan-Kompensation wurde im Rahmen dieser Arbeit jedoch abgesehen.

Der vorliegende Abschnitt basiert auf den Ergebnissen der in [Are09] durchgeführten Untersuchungen der Fehlerbäume und einer Entwicklung sowie Integration des Hypothesenbaumkonzepts in den *Autonomic Tuning Expert*. Nachfolgend soll ein kurzer Einblick in dieses Konzept gegeben werden. Nach einer kurzen Vorstellung von Knotenarten eines Hypothesenbaums in *Abschnitt 6.3.1* befasst sich der *Abschnitt 6.3.2* mit den Spezifika der Hypothesenbaumausführung. Anschließend wird in *Abschnitt 6.3.3* das Konzept der Hypothesenbäume anhand eines kleinen Beispiels veranschaulicht.

6.3.1 Knotenarten eines Hypothesenbaums

Nachfolgend werden die einzelnen Knotenarten (*Top Event*, *Precondition*, *Intermediate Step*, *Intermediate Event*, *Hypothesis*, *Branch*, *Tuning Plan*) eines Hypothesenbaums erläutert:

Top Event. Die Wurzel eines Hypothesenbaums bildet das *Top Event*, welches die Ausführung des Hypothesenbaums initiiert. Dieses *Top Event* beschreibt typischerweise eine ganze Problemklasse, auf deren Lösung der damit verknüpfte Hypothesenbaum spezialisiert ist. Während der Abarbeitung des Hypothesenbaums erfolgt eine schrittweise Eingrenzung des vorliegenden Problems bis schließlich seine Ursachen identifiziert und behoben werden können.

Precondition. Mit Hilfe von *Precondition*-Knoten lassen sich Bedingungen definieren, die für die Abarbeitung des Unterbaums erfüllt sein müssen. Die Auswertung dieser Bedingungen erfolgt unmittelbar vor der Ausführung des Hypothesenbaums, so dass die Bedingungen sich auf statischen oder zumindest zum Zeitpunkt der Auswertung vorliegenden Systemmetriken beziehen dürfen. Wird eine *Precondition* nicht erfüllt, so bedeutet es, dass der gesamte Unterbaum für die aktuelle Hypothesenbaum-Abarbeitung nicht relevant ist und daher nicht betrachtet wird. Die *Preconditions* ermöglichen somit ein initiales Pruning des Hypothesenbaums, indem nicht in Frage kommende Unterbäume ausgeblendet werden. Zu den klassischen Beispielen für *Preconditions* zählen unter anderem die eingesetzte Hardware, das Betriebssystem bzw. bestimmte Versionen der verwendeten Softwaresysteme. Für ein dynamisches Ausblenden von Unterbäumen des Hypothesenbaums zur Laufzeit lassen sich *Intermediate Events* bzw. *Branches* einsetzen.

Intermediate Step. *Intermediate Steps* ermöglichen die Ausführung von Aufgaben, die das eigentliche Tuning, welches durch die Ausführung von Tuning-Plänen umgesetzt wird, unterstützen. Damit lassen sich beispielsweise Prozesse initiieren, die für die Ausführung von Tuning-Plänen bzw. weitere Traversierung des Hypothesenbaums notwendige Daten sammeln. Die *Intermediate Steps* beeinflussen den Kontrollfluss zwar nicht direkt, die durch sie bereitgestellten Daten können jedoch beispielsweise in Verzweigungen (engl. *Branches*) ausgewertet werden und damit die weitere Hypothesenbaum-Traversierung verändern.

Intermediate Event. Während das Auftreten eines abstrakten *Top Events* die Ausführung eines entsprechenden Hypothesenbaums initiiert, ist es oft von Bedeutung, während der Abarbeitung des Hypothesenbaums weitere aufgetretene Events auszuwerten, um dadurch beispielsweise das Problem enger einzugrenzen. Das wird mit *Intermediate Events* ermöglicht. Mit Hilfe eines *Intermediate-Event*-Knotens kann an einer beliebigen Stelle im Hypothesenbaum überprüft werden, ob ein bestimmter Event innerhalb eines vordefinierten Zeitintervalls aufgetreten ist. Ist der Event aufgetreten, so kann die Abarbeitung des Pfades unterhalb des *Intermediate-Event*-Knotens fortgesetzt und dabei bei Bedarf auf die den aufgetretenen Event beschreibenden Metadaten Bezug genommen werden. Ist der Event jedoch nicht bis zum Ablauf der vorgegebenen Zeit aufgetreten, so wird der aktuelle Pfad nicht weiter verfolgt und das Backtracking wird eingeleitet.

Weiterhin lassen sich *Intermediate Events* zur Umsetzung der Synchronisation einsetzen. Wird beispielsweise durch einen *Intermediate Step* ein nebenläufiger Datensammlungsprozess gestartet, so kann mittels eines entsprechenden, in den Hypothesenbaum integrierten *Intermediate Events* die Möglichkeit bereitgestellt werden, auf die Rückmeldung dieses Prozesses zu warten und zu reagieren. Diese Rückmeldung kann dabei einerseits eine Statusmeldung bezüglich der erfolgreich abgeschlossenen Ausführung oder andererseits auch die gesammelten Daten bzw. die Angaben zur

Lokation dieser Daten beinhalten. Da die Ausführungsdauer solcher nebenläufiger Prozesse nur schwer vorhersehbar ist, ist die Wahl einer geeigneten Zeitschranke sehr wichtig. Liegt das Ergebnis des nebenläufigen Prozesses innerhalb einer vorgegebenen Zeitspanne nicht vor, so wird die Abarbeitung des Hypothesenbaums mittels eines Backtracking-Schritts fortgesetzt. Eine spätere Rückkehr in den gleichen Unterbaum und somit eine erneute Überprüfung, ob die Ergebnisse mittlerweile vorliegen, werden jedoch durch den Backtracking-Schritt nicht ausgeschlossen (vgl. *Abschnitt 6.3.2.2*).

Hypothesis. Mit Hilfe von *Hypothesen-Knoten* können die auf dem Weg von der Baumwurzel bis zu diesen Knoten getroffenen Annahmen, auf welchen die Logik ihrer Unterbäume basiert, dokumentiert werden. Diese Knoten sind nicht für eine automatische Auswertung durch die Hypothesenbaum-Ausführungskomponente konzipiert. Vielmehr sollen diese Knoten die Lesbarkeit der Hypothesenbäume erhöhen, was insbesondere aufgrund des in *Kapitel 7* skizzierten Ansatzes zum Austausch von Tuning-Plänen bzw. Hypothesenbäumen in einer Community von großer Bedeutung ist.

Branch. Eine der wichtigsten Knotenarten eines Hypothesenbaums sind die Verzweigungsknoten (*Branch*-Knoten). Während die Knoten jeder anderen Knotenart jeweils nur ein Kindknoten haben und somit den Traversierungsablauf grundsätzlich nicht beeinflussen[28], ermöglichen die *Branch*-Knoten eine dynamische Auswahl des als nächstes abzuarbeitenden Pfades, der die beste Lösung für das aufgetretene Problem verspricht, basierend auf benutzerspezifizierten Ausdrücken.

Ein *Branch*-Knoten hat zwei oder mehrere Ausgänge. Jeder Ausgang ist mit einer entsprechenden Bedingung (*Bedingungsknoten*) verbunden. Die Bedingung kann dabei beispielsweise mittels einer SQL-Anweisung bzw. eines logischen Ausdrucks umgesetzt werden. Liefert die SQL-Anweisung mindestens ein Tupel oder wird der Ausdruck als wahr ausgewertet, so gilt die Bedingung als erfüllt und der Ausgang wird aktiviert, so dass der entsprechende Unterbaum für die weitere Hypothesenbaum-Traversierung verwendet werden kann.

Sind in einer Verzweigung mehrere Ausgänge aktiv, so muss eine Entscheidung bezüglich des besten Pfades für den Abstieg im Baum getroffen werden. In den Entscheidungsprozess können dabei einerseits die den Ausgängen zugewiesenen Prioritäten und andererseits die Metadaten, wie beispielsweise Informationen über die gewählten Pfade bei den vorangehenden Traversierungen oder Informationen über die Effektivität der sich in den jeweiligen Unterbäumen befindenden Tuning-Pläne, einbezogen werden.

Tuning Plan. *Tuning-Plan*-Knoten bilden die Blätter eines Hypothesenbaums. Die einzelnen Traversierungspfade beginnen somit bei der Wurzel des Hypothesenbaums, sprich dem aufgetretenen Problem, und enden je nach Auswertungsergebnissen einzelner Bedingungen in den besuchten Knoten bei einem Tuning-Plan, der die effektivste Lösung unter Berücksichtigung sämtlicher Nebenbedingungen verspricht.

Es ist jedoch möglich, dass obwohl der Tuning-Plan die effektivste Lösung eines Problems versprach, dieses nach der Tuning-Plan-Ausführung weiterhin bestehen

[28] Das Einleiten von Backtracking bei Nichterfüllung von Bedingungen der *Intermediate Events* bzw. beim Auswerten der Effektivität ausgeführter Tuning-Pläne sei an dieser Stelle nicht betrachtet.

bleibt. Aus diesem Grund müssen Metadaten zur Erfassung der Effektivität des Tuning-Plans im Anschluss an die Tuning-Plan-Ausführung unter Berücksichtigung der Einpegelungszeit des Tuning-Plans ausgewertet werden, um festzustellen, ob das Problem bereits gelöst werden konnte oder gegebenenfalls eine weitere Traversierung des Hypothesenbaums notwendig ist. Ist eine weitere Abarbeitung des Hypothesenbaums notwendig, so wird das Backtracking eingeleitet und die Traversierung des Hypothesenbaums fortgesetzt.

6.3.2 Ausführungslogik eines Hypothesenbaums

6.3.2.1 Traversierung

Der Hypothesenbaum wird grundsätzlich entsprechend dem Vorgehen einer Tiefensuche traversiert. Dabei gibt es jedoch einige Abweichungen von der typischen Tiefensuche, wie beispielsweise ein vorzeitiger Abbruch des Abstiegs und eine darauf folgende Suche nach Alternativen (Backtracking) bzw. die Wahl des Abstiegspfades anhand von ausgewerteten Ausdrücken, die an einzelne Verzweigungen geknüpft sind. Nachfolgend wird die Vorgehensweise bei der Traversierung des Hypothesenbaums kurz skizziert:

- Die Traversierung beginnt bei dem am weitesten oben im Baum liegenden Knoten (Top Event).
- Jeder Knoten wird immer vor seinen Kindern und Nachfolgern (Kindeskindern) besucht.
- Die Kinder und Nachfolger eines Knotens werden vor seinen Geschwisterknoten besucht. Durch die Auswertung von Bedingungen beispielsweise in *Intermediate-Event*-Knoten und einer entsprechenden Modifikation der Traversierungsreihenfolge kann es dennoch zu einer Abweichung von dieser Regel kommen.
- Bei Verzweigungen (Knotenart *Branch*) erfolgt in Abhängigkeit von der Auswertung der Verzweigungsbedingungen die Bestimmung der Reihenfolge, in der die Kindknoten mit ihren Unterbäumen besucht werden. Einzelne Pfade können dabei aufgrund von nicht erfüllten Bedingungen entfallen. Verbleiben dennoch mehrere potentiell wählbare Pfade, so sind die Prioritäten einzelner Pfade bzw. weitere Metadaten zur Bestimmung der Traversierungsreihenfolge hinzuzuziehen.
- Wird an irgendeiner Stelle im Hypothesenbaum eine durch einen *Intermediate-Event*-Knoten definierte Bedingung (Auftreten eines bestimmten Events in einem vordefinierten Zeitintervall) nicht erfüllt, so wird auf die Abarbeitung des Unterbaums verzichtet und das Backtracking eingeleitet. Dabei wird der aufgrund der nicht erfüllten Bedingung nicht vollständig abgearbeitete Pfad entsprechend gekennzeichnet, so dass eine spätere Neuauswertung der Bedingung und gegebenenfalls eine Abarbeitung des Unterbaums ermöglicht werden. Dabei kann der Pfad einerseits für den gesamten Traversierungslauf von den weiteren Betrachtungen ausgeschlossen werden. Andererseits kann es unter Umständen sinnvoll sein, dass der Pfad nach Ablauf einer bestimmten Zeit bei einer erneuten Auswertung der Verzweigungsbedingungen

wieder gewählt werden kann. Dadurch wird erneut überprüft, ob das entsprechende *Intermediate Event* aufgetreten ist und, wenn es der Fall ist, der Abstieg im Hypothesenbaum fortgesetzt.

- Die Blätter des Hypothesenbaums bilden die Tuning-Pläne, die bei der Erfüllung sämtlicher Bedingungen auf dem Weg von der Wurzel des Hypothesenbaums bis zum Tuning-Plan selbst ausgeführt werden. Im Anschluss an die Ausführung eines Tuning-Plans erfolgt die Überprüfung seiner Wirksamkeit anhand von den die Effektivität des Tuning-Plans repräsentierenden Metadaten. Gegebenenfalls erfolgt dabei die Auswertung dieser Metadaten erst nach Verstreichen der Einpegelungszeit des Tuning-Plans. Wird basierend auf diesen Metadaten bzw. auf der Tatsache, dass das Problem seit der Tuning-Plan-Ausführung nicht mehr aufgetreten ist, festgestellt, dass das Problem durch die Tuning-Plan-Ausführung gelöst werden konnte, so kann die Traversierung des Hypothesenbaums beendet werden. War die Tuning-Plan-Ausführung hingegen nicht erfolgreich (das Problem besteht weiterhin), so wird das Backtracking eingeleitet und die Suche nach einer besseren Problemlösung wird fortgesetzt.

- Ist die Traversierung des gesamten Hypothesenbaums abgeschlossen und das Problem noch nicht behoben, kann eine erneute Traversierung des Baums eingeleitet werden. Die Steuerung dieses Vorgehens erfolgt durch eine entsprechende Richtlinie des Hypothesenbaums.

- Für einzelne Tuning-Pläne kann die maximale Anzahl ihrer Ausführungen angegeben werden. Erfolgt nun eine mehrfache Traversierung des Hypothesenbaums und wurde dabei für einen Tuning-Plan die zulässige Anzahl von Ausführungen erreicht, so wird auf eine weitere Ausführung des Tuning-Plans verzichtet. Stattdessen wird entweder das Backtracking eingeleitet oder die Ausführung des Hypothesenbaums beendet.

Die Steuerung der Traversierung kann bei Bedarf mittels spezieller Hypothesenbaum-Richtlinien, die vom Autor des Hypothesenbaums angegeben werden, gesteuert werden. Von weiteren Betrachtungen dieser Richtlinien soll im Rahmen der vorliegenden Arbeit jedoch abgesehen werden.

6.3.2.2 Backtracking

Wird während der Traversierung des Hypothesenbaums an irgendeiner Stelle festgestellt, dass entweder die Fortsetzung der Ausführung auf dem aktuellen Pfad nicht mehr möglich ist (z. B. *Intermediate Event* ist im vorgegebenen Zeitraum nicht eingetreten) oder die Ausführung eines Tuning-Plans zur Behebung des bestehenden Problems nicht ausreichte, so ist die Suche nach einem alternativen Lösungsweg notwendig. Dazu greifen wir auf das Backtracking – ein bewährtes Konzept der Algorithmik zur Suche von alternativen Problemlösungen – zurück. Die Suche nach einer Alternative erfolgt hier durch einen Aufstieg im Hypothesenbaum bis zur nächsten Verzweigung.

Im Gegensatz zur klassischen Backtracking-Vorgehensweise, bei der von der Verzweigung ein noch nicht besuchter Pfad gewählt werden würde, erfolgt an dieser Stelle in unserem Ansatz eine erneute Auswertung der Verzweigungsbedingungen und basierend darauf wird

eine Entscheidung bezüglich der Pfadauswahl getroffen. Anhand von Metadaten und gültigen Richtlinien des Hypothesenbaums kann dabei unter Umständen ein bereits besuchter oder sogar der gleiche Pfad, aus welchem der Aufstieg soeben erfolgte, gewählt werden.

Wurde das Backtracking eingeleitet, so erfolgt ein Aufstieg im Hypothesenbaum bis die erste Verzweigung erreicht ist. Beim Aufstieg kommen dabei zwei Strategien in Frage: Zum einen kann beim Aufstieg im Hypothesenbaum eine Kompensation sämtlicher, sich auf das zu tunende System auswirkender Änderungsoperationen vorgenommen werden. Bei solchem *kompensierenden Backtracking* ist jedoch lediglich eine Kompensation von Tuning-Plänen notwendig, da andere Knotenarten des Hypothesenbaums per Definition keine Veränderungen am Zustand des zu tunenden Systems vornehmen. Zur Ermöglichung der Tuning-Plan-Kompensation sind in den Tuning-Plänen entsprechende Kompensationsoperationen bzw. -abläufe zu hinterlegen.

Zum anderen ist ein Aufstieg ohne Kompensation möglich. Das *nicht-kompensierende Backtracking* ist insbesondere im Kontext des Datenbank-Tuning hilfreich, da hier eine Kompensation der ausgeführten Operationen nicht immer möglich bzw. notwendig ist. So ist beispielsweise eine physische Reorganisation der internen Tabellenspeicherung (`Reorg`) nicht kompensierbar. Die Entscheidung bezüglich der verwendeten Backtracking-Strategie muss vom Autor des Hypothesenbaums getroffen und mittels spezieller Hypothesenbaum-Richtlinien definiert werden.

6.3.3 Beispiel eines Hypothesenbaums

In ***Abbildung 6.6*** wird ein Beispiel-Hypothesenbaum zur Diagnose und Auflösung von Sperrproblemen dargestellt. Der dargestellte Hypothesenbaum konzentriert sich dabei größtenteils auf die bereits in *Abschnitt 4.2* diskutierten Fälle und veranschaulicht dabei die Funktionsweise sämtlicher Konstrukte des Hypothesenbaums aus *Abschnitt 6.3.1*.

Die Wurzel des Hypothesenbaums (*Knoten 0*) bildet das Ereignis, welches besagt, dass die Anzahl von Sperreskalationen einer Tabelle über einem bestimmten, systemabhängigen Schwellenwert liegt und damit zu hoch ist. Dieses Ereignis wird durch die entsprechenden ATE-Komponenten erkannt und anschließend wird ein diesem Ereignis zugeordneter Hypothesenbaum identifiziert und an die Hypothesenbaum-Ausführungskomponente übergeben.

Mit Hilfe eines *Precondition*-Knotens (*Knoten 1*) wird sichergestellt, dass der vorliegende Hypothesenbaum ausschließlich auf einem *DB2 LUW*-System ausgeführt werden kann. Handelt es sich bei dem eingesetzten DBMS beispielsweise um ein Oracle-System oder *DB2 for z/OS*, so wird die Ausführung des Hypothesenbaums abgebrochen[29].

Die *Knoten 2*, *2a* sowie *2b* dienen einer Fall-Unterscheidung, ob einzelne Anwendungen mehr Sperren als durch den Wert des Parameters `maxlocks` erlaubt ist, belegen oder die Sperrliste durch die Gesamtheit der Sperren aller Anwendungen vollgelaufen ist, ohne dass die `maxlocks`-Schranke durch die einzelnen Anwendungen überschritten wurde. Die Abgrenzung beider Fälle wird mittels entsprechender SQL-Anweisungen umgesetzt.

[29] In der Praxis würden solche *Preconditions* in der Regel bereits bei der Auswahl von Hypothesenbäumen berücksichtigt werden.

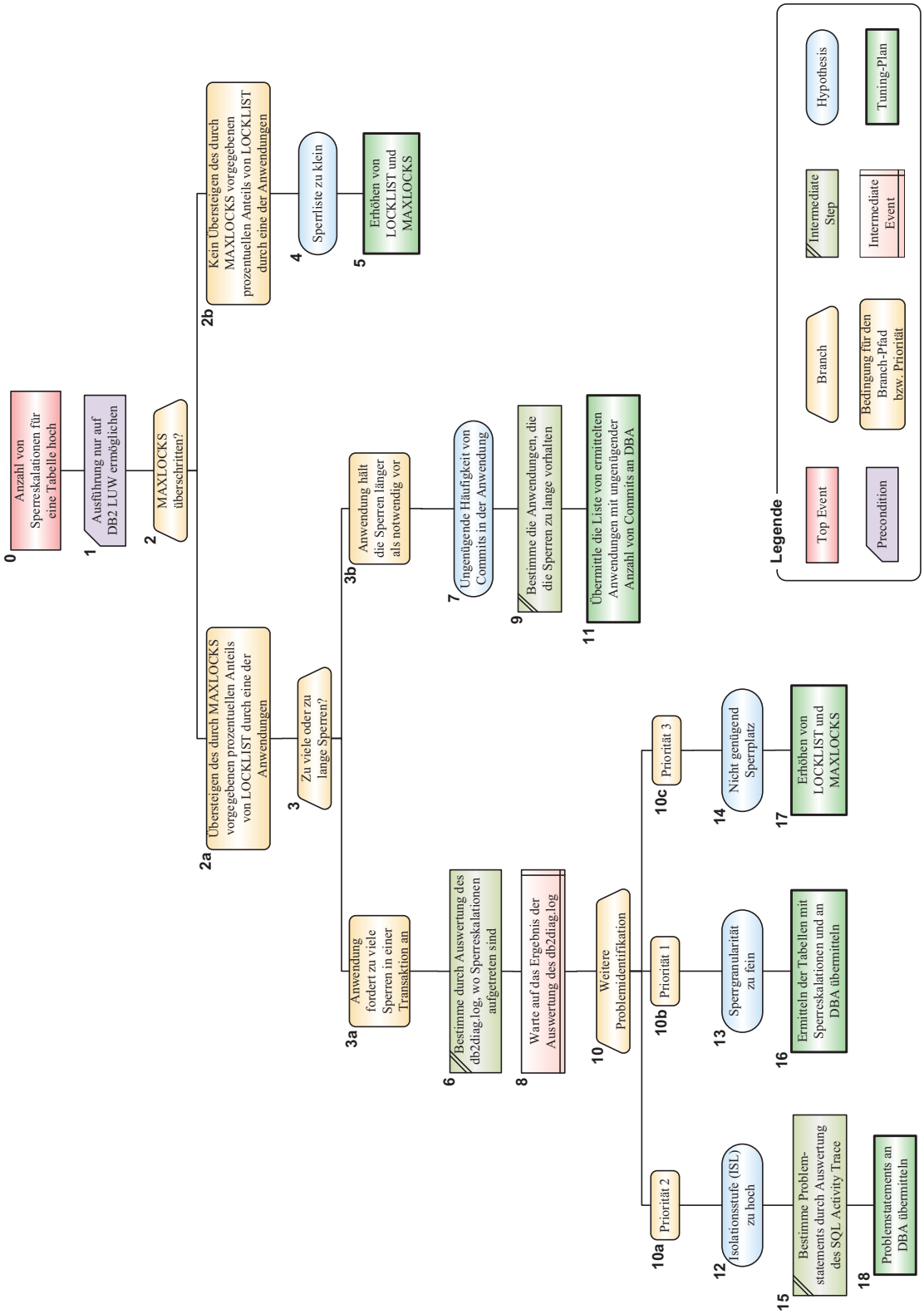


Abbildung 6.6: Beispiel eines Hypothesenbaums zur Lösung des Sperreskalationen-Problems

Die Unterscheidung, ob zu viele oder zu lange Sperren von den Anwendungen gehalten werden, wird mittels des *Branch*-Knotens 3 sowie der entsprechenden *Bedingungsknoten 3a* bzw. *3b* umgesetzt. Auch hier werden die Bedingungen durch die SQL-Anweisungen definiert.

Im Fall der zu vielen Sperren wird mittels eines *Intermediate Steps* (*Knoten 6*) das *db2diag.log* ausgewertet und bestimmt, wo die Sperreskalationen aufgetreten sind. Da es sich bei diesem *Intermediate Step* um einen *asynchronen Step* handelt, kann die Traversierung des Hypothesenbaums parallel zur Ausführung des *Intermediate Steps* fortgesetzt werden. Mittels des *Intermediate Events* (*Knoten 8*) kann nun abgefragt werden, ob das Ergebnis des *Intermediate Steps* bereits vorliegt oder nicht. Werden die nötigen Informationen nicht innerhalb einer bestimmten Zeit bereitgestellt, so wird an dieser Stelle das Backtracking eingeleitet. Bei einem eventuellen späteren Abstieg in den gleichen Unterbaum kann der bereits ausgeführte *Intermediate Step* (*Knoten 6*) übersprungen und anhand des *Intermediate Events* (*Knoten 8*) festgestellt werden, ob die geforderten Daten mittlerweile vorliegen. Erst wenn diese Informationen vorliegen, kann die Traversierung des Baums unterhalb des *Intermediate-Event*-Knotens fortgesetzt werden.

Mit den *Knoten 10*, *10a*, *10b* und *10c* wird die weitere Problemidentifikation umgesetzt. Die einzelnen *Bedingungsknoten* haben in diesem Fall keine Ausdrücke, sondern sind mit Prioritäten versehen, die die Reihenfolge der Pfadabarbeitung festlegen. Als erstes wird also der Annahme nachgegangen, dass die Sperrgranularität zu fein ist (*Knoten 13*) und ein entsprechender Tuning-Plan wird ausgeführt. Bringt dieser Tuning-Plan nicht die gewünschte Wirkung, so wird das Backtracking eingeleitet und der Pfad mit der nächst höheren Priorität wird gewählt (*Knoten 12*, *15*, *18*).

Der *Intermediate Step* zur Auswertung des *SQL Activity Trace* (*Knoten 15*) injiziert im Gegensatz zum *Intermediate Step* zur Auswertung des *db2diag.log* die Ergebnisse nicht in den ATE-Kreislauf, sondern speichert diese in eine Datei, auf welche der Tuning-Plan (*Knoten 18*) Zugriff bekommt. Bei diesem *Intermediate Step* handelt es sich also um einen *synchronen Step*: Die weitere Traversierung des Hypothesenbaums und somit die Ausführung des Tuning-Plans im *Knoten 18* erfolgt erst nach der Fertigstellung dieses *Intermediate Steps*. Sollte die Ausführung dieses Tuning-Plans nicht die gewünschte Wirkung erzielt haben, so wird als nächstes der Pfad mit der Priorität 3 abgearbeitet. Die entsprechende Hypothese, dass nicht genügend Sperrplatz vorliegt, wird mit dem Tuning-Plan in *Knoten 17* adressiert. Die Abarbeitung weiterer Pfade des vorliegenden Hypothesenbaums erfolgt analog und soll daher an dieser Stelle nicht weiter erläutert werden.

6.3.4 Abschließende Bemerkungen zu Hypothesenbäumen

Das im vorliegenden Abschnitt vorgestellte Konzept der Hypothesenbäume unterstützt die Administratoren bei der Modellierung der Planungsabläufe. Eine entsprechende Hypothesenbaum-Ausführungskomponente traversiert die Hypothesenbäume zur Laufzeit und identifiziert durch die Auswertung der Hypothesenbaumknoten die möglichen Ursachen für das vorliegende Problem. Die Blätter eines Hypothesenbaums bilden die Tuning-Pläne, die nach einer Identifikation des zu lösenden Problems ausgeführt werden. Wurden auf dem Pfad von der Wurzel des Hypothesenbaums bis zum aktuellen Blattknoten falsche Annahmen getroffen, so dass der ausgeführte Tuning-Plan keine Verbesserung des Systemzustands erreichte, so wird das Backtracking eingeleitet und die Annahmen werden erneut

überprüft, was typischerweise in der Bestimmung eines alternativen Pfades im Hypothesenbaum und einer anschließenden Ausführung eines anderen Tuning-Plans resultiert.

Während Tuning-Pläne also typisches DBA-Verhalten repräsentieren und sich auf die Betrachtung eines bestimmten *isolierten* Problems, welches durch einen spezifischen Event repräsentiert wird, konzentrieren, ermöglichen die Hypothesenbäume die Betrachtung einer *gesamten* Problemklasse, die durch das *Top Event* repräsentiert wird. Zur Laufzeit erfolgt eine Auswertung weiterer Daten, was die Eingrenzung der möglichen Teilprobleme und ihrer Ursachen ermöglicht, die schließlich durch entsprechende Tuning-Pläne behoben werden können.

Bei der Umsetzung des Hypothesenbaum-Konzepts stellt sich die Frage, ob ein herkömmliches Workflow-Management-System für die Modellierung bzw. Ausführung von Hypothesenbäumen geeignet ist oder nicht. In [Are09] wurden aus diesem Grund einige Workflow-Management-Systeme bezüglich ihrer Einsetzbarkeit bei der Umsetzung des Hypothesenbaum-Konzepts untersucht. Neben der Unterstützung aller Knotenarten (vgl. Abschnitt 6.3.1) sind bei der Umsetzung dieses Konzepts insbesondere die Spezifika der Ausführungslogik der Hypothesenbäume zu berücksichtigen. So muss eine Hypothesenbaum-Ausführungskomponente mit den Backtracking-Schritten umgehen und die Traversierungsreihenfolge anhand von ausgewerteten Bedingungen bestimmen können. Diese Funktionalitäten mittels eines herkömmlichen Workflow-Management-Systems zu modellieren, ist sehr aufwendig und wenig praktikabel, so dass hier auf die Verwendung eines WfMS verzichtet wurde. Für weitere Ausführungen zum Konzept der Hypothesenbäume sowie den damit verbundenen Erweiterungen des *Autonomic Tuning Expert* sei hier auf [Are09] verwiesen.

6.4 Zusammenfassung

In diesem Kapitel wurde eine Auswahl von Konzepten vorgestellt, die die Planungsphase des MAPE-basierten autonomen Datenbank-Tuning unterstützen und um neue Funktionalitäten erweitern. Mit Hilfe von in *Abschnitt 6.1* eingeführten Metadaten können beispielsweise sowohl die Auswahl von Tuning-Plänen als auch ihre Ausführung optimiert werden. Durch die Verwaltung einer Historie von ausgeführten Tuning-Plänen sowie ihrer Operationen können weiterhin Schwankungen zwischen Systemzuständen (Oszillationen) vermieden werden, indem eine abwechselnde Ausführung einander widersprechender Tuning-Pläne verhindert wird. Ein Beispiel für solche Tuning-Pläne, deren abwechselnde Ausführung zu einer Oszillation führt, sind $TP1$, der die Allokation einer Ressource um einen bestimmten Wert erhöht, sowie $TP2$, der die Allokation der gleichen Ressource verringert. Solche Oszillationen lassen sich beispielsweise aus den Metadaten bezüglich der durch die einzelnen Tuning-Pläne vorgenommenen Änderungen der Ressourcenallokationen erkennen und vorbeugen.

Mit den auf den Metadaten aufbauenden Tuning-Richtlinien und einer entsprechenden Komponente, die diese zur Laufzeit auswertet, wurde eine Möglichkeit geschaffen, um die von Nutzern vorgegebenen abstrakten Tuning-Präferenzen zur Laufzeit auf die entsprechenden Tuning-Pläne bzw. sogar Tuning-Schritte abzubilden (*Abschnitt 6.2*). Durch die Berücksichtigung von Tuning-Richtlinien können sämtliche Phasen des MAPE-Kreislaufs optimiert und an den Zielvorgaben der DBA ausgerichtet werden. Die Tuning-Richtlinien

bieten somit eine Schnittstelle, um das Tuning des Systems zur Laufzeit indirekt zu beeinflussen.

Mit Hilfe von Tuning-Richtlinien lassen sich beispielsweise die den Nutzervorgaben widersprechenden bzw. aus anderen Gründen uninteressanten Ereignisse deaktivieren und somit die Effizienz der Überwachungs- und Analyse-Phasen steigern. Die Tuning-Plan-Auswahl lässt sich durch die Informationen bezüglich der Tuning-Plan-Semantik bzw. durch die von DBA vorgegebenen Einschränkungen bezüglich des Tuning der Datenbankobjekte sowie ihre Priorisierung steuern. Auch die Tuning-Plan-Ausführung wird durch solche Einschränkungen beeinflusst, indem ganze Tuning-Pläne oder nur ihre einzelnen Tuning-Schritte zurückgewiesen werden, falls sie den Nutzervorgaben widersprechen. Bei der Zurückweisung einzelner Tuning-Schritte muss berücksichtigt werden, dass dadurch die Atomarität entsprechender Tuning-Pläne nicht mehr gewährleistet ist. Sollte also die Forderung nach der Atomarität ausgewählter Tuning-Pläne bestehen, so sind entsprechende Anpassungen am System vorzunehmen, damit diese Tuning-Pläne nicht partiell, sondern nur als Ganzes zurückgewiesen werden können.

Das ebenfalls in diesem Kapitel vorgestellte Konzept der Hypothesenbäume ermöglicht es den Administratoren, die gesamte Planungslogik manuell zu erfassen und dem System zur Verfügung zu stellen (*Abschnitt 6.3*). Metadaten bzw. Tuning-Richtlinien lassen sich bei diesem Ansatz nach wie vor in die Planung miteinbeziehen. Die Planungsabläufe können nun jedoch speziell auf einzelne Problembereiche zugeschnitten werden, was in den meisten Fällen eine deutliche Beschleunigung des Problemdiagnose- und -behebungsprozesses ermöglicht.

Zusammenfassend lässt sich sagen, dass durch die Integration der im vorliegenden Kapitel vorgestellten Konzepte sich nicht nur die Qualität der Planungsphase, sondern auch der Autonomie-Grad des *Autonomic Tuning Expert* deutlich steigern lässt.

Kapitel 7

Zentralisierte Verwaltung und Austausch von Tuning–Plänen

Datenbankadministratoren nutzen heutzutage verschiedene Medien, um ihre bewährten Datenbank-Tuning-Praktiken untereinander auszutauschen. So stehen vielfältige Informationen in elektronischer Form z. B. in News-Groups, IT-Blogs, Online-Magazinen, „IBM Redbooks“-Veröffentlichungen oder Produkthandbüchern zur Verfügung. All diese Informationen sind jedoch zum größten Teil unstrukturiert, das heißt nicht formalisiert und somit auch nicht maschinell abarbeitbar [RWRA08].

Durch die Konzeption einer webbasierten *Community-Plattform* soll der Prozess der Informationsakquise unterstützt werden, indem einerseits eine kollaborative Entwicklung von Tuning-Plänen und andererseits ihr Austausch unter den Community-Mitgliedern ermöglicht werden.

Dieses Kapitel verfolgt das Ziel, Konzepte zu erarbeiten, die einen Austausch von Datenbank-Tuning-Praktiken in einer Community ermöglichen. Zu den an die hier betrachtete Community-Plattform gestellten Hauptanforderungen gehören dabei:

- Effiziente Speicherung von Tuning-Plänen auf dem Server,
- Möglichkeit zur kontrollierten Evolution von Tuning-Plänen,
- Leichte, intuitive Durchsuchbarkeit des Repositorys sowie
- Funktionalitäten zur Motivation der Nutzer, sich am Wissensaustausch zu beteiligen.

Die Community-Plattform hat somit zwei integrale Bestandteile: Das Tuning-Plan-Repository (*Abschnitt 7.1*) sorgt für eine effiziente Speicherung, Suche und Evolution von Tuning-Plänen. Mit Hilfe der umzusetzenden Community-Funktionen (*Abschnitt 7.2*) werden die Nutzer motiviert, sich an der Weiterentwicklung der Tuning-Pläne zu beteiligen und für ihre Beteiligung belohnt. In *Abschnitt 7.3* findet sich anschließend ein Vorschlag zur Integration der in diesem Kapitel vorgestellten Konzepte in die bestehende ATE-Architektur. *Abschnitt 7.4* fasst letztendlich die in diesem Kapitel vorgestellten Konzepte zusammen.

7.1 Ein Repository zur Verwaltung von Tuning-Plänen

Zur Ermöglichung einer effizienten Speicherung von Tuning-Plänen, zur Bereitstellung von Mechanismen zur ihren kontrollierten Evolution und schließlich zur Umsetzung einer intuitiven Suche nach gespeicherten Tuning-Pläne bieten sich die bewährten Konzepte

der Repository-Systeme an. Nach einer kurzen Einführung in die Welt der Repository-Systeme finden sich daher Untersuchungen und Konzepte zur Umsetzung ausgewählter Repository-Funktionen, die sowohl eine effiziente Verwaltung von Tuning-Plänen als auch die Unterstützung der Nutzer bei ihrer Erstellung und Weiterentwicklung ermöglichen.

Repositories finden insbesondere in der Softwareentwicklung beispielsweise zur Verwaltung des Quellcodes oft Anwendung. Ihr Einsatz bringt viele Vorteile. Eine Auswahl dieser Vorteile findet sich im Folgenden:

- Redundante Dokumente werden vermieden und die Qualität der Entwicklung wird verbessert, da der Zugang zu Informationen von dem Repository gesteuert wird.
- Die Wartung des Softwaresystems wird deutlich einfacher, weil alle Ergebnisse der Softwareentwicklung in dem Repository verwaltet werden.
- Durch die von einem Repository bereitgestellten Mechanismen zur Suche nach und Verwaltung von Informationen wird die Wiederverwendung von verwalteten Informationen vereinfacht.
- Das Repository enthält Regeln zur Sicherung der Datenintegrität, so dass die Qualität der gespeicherten Daten auf lange Sicht gesteigert wird.
- Durch eine Erfassung und Verwaltung von Auditinformationen lassen sich Änderungen an Informationen nachverfolgen.

7.1.1 Aufgaben eines Repository

In Anlehnung an [BD94] lassen sich die Aufgaben eines Repository in zwei Klassen einteilen:

- die Datenbank-Funktionen und
- die spezifischen Repository-Funktionen.

Im Folgenden findet sich zunächst eine kurze Zusammenfassung der wichtigsten Repository-Aufgaben. Auf die einzelnen Repository-Aufgaben und auf die Möglichkeiten zu ihrer Umsetzung, um eine effiziente Verwaltung und einen Austausch von Tuning-Plänen zu ermöglichen, wird in den kommenden Abschnitten näher eingegangen.

Datenbank-Funktionen

Zu den für ein Repository notwendigen Datenbank-Funktionen zählen unter anderem: Abfragemöglichkeiten des Repositorys, Administration und Nutzerverwaltung sowie Sicherheits- und Zugriffsmechanismen.

Repository-Funktionen

Versionsverwaltung. Bei der Versionsverwaltung handelt es sich um eine Disziplin, die es ermöglicht, den Entstehungsprozess von verwalteten Objekten zu protokollieren. Dabei lassen sich nicht nur Änderungen der Objekte erfassen und archivieren, auch alte Stände dieser Objekte können jeder Zeit wiederhergestellt werden, um damit beispielsweise versehentliche Änderungen rückgängig zu machen. Die Versionsverwaltung stellt weiterhin Mechanismen bereit, die den Mehrbenutzerbetrieb ermöglichen. Weiterführende Informationen zu ausgewählten Konzepten der Versionsverwaltung sowie Untersuchungen bezüglich des Einsatzes dieser Konzepte bei der Verwaltung und Versionierung von Tuning-Plänen finden sich in *Abschnitt 7.1.2*.

Benachrichtigungsfunktionen. Benachrichtigungsdienst ist ein Mechanismus, der beim Auftreten bestimmter Ereignisse (wie Veränderung eines Objekts) vordefinierte Aktionen ausführen kann (*Abschnitt 7.1.3*). Anwendungsgebiete sind dabei beispielsweise die Änderungspropagierung bei redundanter Datenspeicherung oder bei abgeleiteten Varianten.

Präsentation von Änderungen. Oft fällt es den Nutzern schwer, einen Überblick über die Evolutionshistorie von Objekten zu behalten, die vielen Änderungen unterlagen. Zu ihrer Unterstützung bieten die Repository-Systeme Werkzeuge an, die eine übersichtliche Darstellung der Änderungshistorie unterstützen (*Abschnitt 7.1.4*).

Mehrbenutzerbetrieb. Die Arbeit an den Informationen aus einem Repository dauert meistens über einen längeren Zeitraum (Tage, Monate) an. Daher ist es unpraktisch, diese Aktivitäten als eine klassische Transaktion im Datenbanksinn zu behandeln. Durch die Einführung von *Checkout*- und *Checkin*-Operationen lassen sich Objekte über längere Zeiträume sperren (*pessimistisches Locking*) bzw. zuerst in neue Objekte umkopieren und anschließend, nach erfolgten lokalen Änderungen, die Objekte wieder zusammenführen (*optimistisches Locking*). Das pessimistische Locking ist auch bekannt als das *Sperren–Ändern–Freigeben–Prinzip* (engl.: *Lock–Modify–Unlock*). Das optimistische Locking entspricht dagegen dem *Kopieren–Ändern–Zusammenführen–Prinzip* (engl.: *Copy–Modify–Merge*). Nähere Informationen zum Mehrbenutzerbetrieb finden sich in *Abschnitt 7.1.5*.

Die nachfolgenden Unterabschnitte befassen sich nun mit ausgewählten Repository-Konzepten, die sich im Bereich der Tuning-Plan-Verwaltung und -Weiterentwicklung einsetzen lassen.

7.1.2 Versionsverwaltung

Bei der Versionsverwaltung handelt es sich um eine Teildisziplin des Konfigurationsmanagements. In der Softwareentwicklung ist das Hauptziel des Konfigurationsmanagements die Steigerung der Produktivität und Qualität der Software. Dieses Ziel wird unter anderem durch das Einhalten von Richtlinien wie beispielsweise *ISO 9000*, *Capability Maturity Model (CMM)* oder *Capability Maturity Model Integration (CMMI)* [Som01] bzw. durch den Einsatz von gewissen Mechanismen erreicht. Diese Mechanismen helfen dabei, einen Überblick über die vielen Komponenten eines zu entwickelnden Systems und die vielen Versionen dieser Komponenten zu behalten und ermöglichen eine gleichzeitige Überarbeitung

von Systemkomponenten durch mehrere Entwickler. Weiterhin sorgen sie für die Verwaltung aller Änderungen an Dokumenten, die der Versionierung unterliegen [Whi91].

Während beim Software-Konfigurationsmanagement der gesamte Lebenszyklus eines Softwaresystems im Vordergrund steht, konzentriert sich die Versionsverwaltung auf die Betrachtung einzelner Objekte, deren Entstehungsprozess es zu protokollieren gilt. Den Kern einer Versionsverwaltung bildet das Versionierungsmodell. Es umfasst die Konstrukte zur Beschreibung von Versionen und ist nach [CW98] primär für folgende Aspekte verantwortlich:

- Definition von zu versionierenden Objekten (Versionierungsobjekten).
- Konzepte zur Identifikation und Organisation von Versionen.
- Beschreiben von Eigenschaften, die über alle Versionen eines Objekts gleich sind (auch bekannt als gemeinsame Eigenschaften bzw. Invarianten).
- Beschreiben von Unterschieden (Deltas) zwischen den einzelnen Versionen eines Objekts.
- Festlegung der Verwaltungsspezifika von Versionsmengen einzelner Objekte.
- Einführung von Evolutionsdimensionen wie Revisionen bzw. Varianten (siehe unten).
- Festlegung, ob die Versionierung zustandsorientiert oder änderungsorientiert erfolgen soll.
- Festlegung einer passenden Repräsentation der Versionsmengen (beispielsweise als Versionsgraphen).
- Definition einer Menge von Operationen zum lesenden Zugriff auf alte bzw. aktuelle Versionen und zur Erzeugung (beispielsweise mittels Zusammenführung mehrer Versionen) von neuen Versionen.

Auf die Struktur der zu versionierenden Elemente (Dokumente, Dokumentfragmente) existieren unterschiedliche Sichten. So wird beispielsweise in [CW97] zwischen einem Produkt- und einem Versionsraum unterschieden:

Produktraum. Der Produktraum beschreibt die Beziehungen zwischen allen unversionierten Elementen eines Softwareprodukts. Dazu zählen beispielsweise alle Beziehungen zwischen einzelnen Diagrammen aus der Entwurfsphase sowie die Beziehungen zwischen einzelnen Klassen. Die Existenz von Versionen bleibt bei dieser Betrachtung unberücksichtigt. Da sich die vorliegende Arbeit mit der Versionierung von Tuning-Plänen und nicht von gesamten Softwareprodukten beschäftigt, soll hier auf die Beschreibung des Produktraums nicht weiter eingegangen werden.

Versionsraum. Der Versionsraum beschreibt die Beziehungen zwischen den Versionen eines Elements. Die Versionen lassen sich anhand von Versionsidentifizierern unterscheiden. Im Folgenden konzentrieren wir uns auf weitere Betrachtungen ausschließlich von Versionsraum-Konzepten.

Anhand der Struktur des Versionsraums lassen sich zwei Arten von Versionen unterscheiden: Revisionen und Varianten.

Revision. Eine Revision beschreibt einen bestimmten Änderungszustand eines Objekts. Sie entsteht durch Änderung (Revidierung) einer alten Revision des Entwurfsobjekts (*neu = alt +/– Änderungen*). Bei den Änderungen handelt es sich typischerweise um Korrekturen von Fehlern bzw. um Weiterentwicklungen des Objekts (Funktionalitätserweiterungen). Eine Revision ist dafür bestimmt, die Vorgängerversion, auf welcher sie aufbaut, logisch zu ersetzen. Dabei erfolgt keine physische Ersetzung der alten Revisionen. Sie werden aufgehoben, um die Wartung des Produkts zu vereinfachen (Rückkehr auf die Vorgängerversion im Fehlerfall) bzw. einfach zu Dokumentationszwecken. Die zeitliche Ordnung der Revisionen ist dabei von großer Bedeutung und lässt sich beispielsweise durch die Revisionsnummerierung abbilden.

Variante. Eine Variante stellt eine alternative Umsetzung eines Objekts dar. Varianten (auch als Alternativen bekannt) existieren parallel zueinander und setzen die gleichen Funktionalitäten für leicht abweichende Situationen bzw. Systemumgebungen um. Durch Einsatz von alternativen Datenstrukturen lassen sich beispielsweise Varianten entwickeln mit für bestimmte Gegebenheiten optimiertem Speicherplatzverbrauch bzw. Laufzeitverhalten. Oft unterscheidet man zwischen permanenten und temporären Varianten. Während permanente Varianten erzeugt werden, um über längeren Zeitraum zu existieren (beispielsweise Anpassung eines Produktes an unterschiedliche Kundenanforderungen), entstehen die temporären Varianten bei einer kollaborativen Entwicklung einer Version und entsprechen einer Verzweigung im Versionierungsgraphen. Anschließend werden die temporären Varianten zu einer Version (Revision oder Variante) zusammengeführt.

An dieser Stelle soll erwähnt werden, dass im Sprachgebrauch oft die Grenzen zwischen Versionen und Revisionen verschwimmen und oft von Versionen gesprochen wird, wenn eigentlich Revisionen gemeint sind. In nachfolgenden Abschnitten wird dennoch eine saubere Trennung der verwendeten Begriffe angestrebt.

Des Weiteren sollen an diese Stelle weitere Begriffe eingeführt werden, die zum besseren Verständnis der in diesem Kapitel vorgestellten Konzepte beitragen sollen:

Versionierungsobjekt (oft auch als *Entwurfsobjekt*, *Design-Objekt* bzw. *versioned item* bekannt) bezeichnet ein Objekt, welches versioniert wird.

Initiale Revision ist die erste Revision, die für ein Objekt verwendet wurde.

Letzte Revision ist die zuletzt erzeugte Revision eines Objekts.

Aktuelle Revision ist die aktuelle Arbeitsrevision vom Anwender. Typischerweise entspricht die aktuelle Revision der letzten Revision. Es sind jedoch auch Szenarien denkbar, wo eine ältere Revision zur aktuellen Revision gemacht wird.

Konfiguration beschreibt die Zusammensetzung eines komplexen Objekts, wie beispielsweise eines Software-Produkts, aus spezifischen Revisionen seiner Komponenten [BM93, Cat94].

In folgenden Unterabschnitten werden nun ausgewählte Konzepte und Verfahren der Versionierung vorgestellt. Im Vordergrund der Betrachtungen stehen allerdings primär die für die Konzeption und Entwurf eines Tuning-Plan-Repository-Systems relevanten Aspekte.

7.1.2.1 Versionierungsarten

Extensionale bzw. intensionale Versionierung. [CW97] unterscheiden zwischen der extensionalen und intensionalen Versionierung. Bei der *extensionalen Versionierung* existiert zu jedem Entwurfsobjekt eine Menge von explizit definierten und zuvor in das Repository eingecheckten Versionen ($V = \{v_1, v_2, \ldots, v_n\}$). Diese Versionen lassen sich über ihre Versionsnummer identifizieren. Neue Versionen müssen durch explizite Zusammenführung von Versionen erstellt werden.

Die *intensionale Versionierung* kommt im Gegensatz dazu bei der Notwendigkeit einer flexiblen, automatischen Erstellung von konsistenten Versionen zum Einsatz. Ein Entwurfsobjekt wird nicht mehr durch eine Auflistung aller seiner Versionen, sondern durch ein Prädikat $V = \{v \mid vd(v) \wedge c(v)\}$ zur impliziten Bestimmung aller potenziellen Versionen, die das Prädikat erfüllen, definiert. vd ist dabei eine benutzerdefinierte Versionsbeschreibung und c beschreibt zusätzliche Bedingungen, die für alle Versionen von V gelten müssen. Sowohl vd als auch c sind dabei boolesche Ausdrücke, die über den Selektionsattributen definiert sind und Wertebelegungen dieser Attribute enthalten. Beispiele für solche Selektionsattribute sind das Betriebssystem-Attribut, um das Ziel-Betriebssystem der zu erstellenden Version anzugeben oder das Fix-Attribut, um anzugeben, ob ein bestimmter Fix in die zu erstellende Version mit aufzunehmen ist. Die Versionen werden somit anhand von Beschreibungsmerkmalen identifiziert. Gesteuert wird die intensionale Versionierung durch einen Satz von Regeln, die beschreiben, wie die Versionen zu erstellen sind.

Ein Beispiel für die intensionale Versionierung bietet die *bedingte Kompilierung* (*Conditional Compilation*), die unter anderem von der Programmiersprache *C* bekannt ist. Der Präprozessor erzeugt die Quellcode-Dateien in Abhängigkeit von den Werten der Präprozessor-Variablen. Alle Fragmente des Quellcodes, deren Bedingungen als falsch ausgewertet werden, werden aus dem endgültigen Quellcode ausgeschlossen.

Zustandsorientierte bzw. änderungsorientierte Versionierung. Eine weitere Möglichkeit zur Klassifikation der Versionierungsarten bietet die Unterscheidung zwischen einer zustandsorientierten und einer änderungsorientierten Versionierung [CW97].

Bei der *zustandsorientierten Versionierung* werden die Versionen grobgranular als Objekte beschrieben. Typischerweise erfolgt dabei eine Verwaltung der Informationen, ob es sich bei den einzelnen Objekten um Revisionen bzw. Varianten von anderen Objekten handelt. Es erfolgt keine Nachverfolgung der vorgenommenen Änderungen. Jede Änderung führt zu einer neuen Revision der Vorgängerversion. Eine Bestimmung von Unterschieden zwischen zwei Versionen ist nur durch einen aufwendigen Vergleich beider Versionen möglich.

Bei der *änderungsorientierten Versionierung* dagegen werden die Versionen mit Hilfe von Änderungen bezüglich einer vorgegebenen Baseline (Basisversion) beschrieben. Jede Änderung wird dabei als eine Sequenz von Änderungsoperationen angewendet auf die Basiselemente des Modells repräsentiert. Beispiele für solche Änderungsoperationen sind *„Element hinzufügen“*, *„Element löschen“* bzw. *„Element ändern“*. Die Änderungsoperationen werden oft mit weiteren Metadaten versehen, die die Änderungen annotieren. Durch solche Annotationen können auch die Gründe der

	Zustandsorientierte Versionierung	**Änderungsorientierte Versionierung**
Intensionale Versionierung	• Explizite Erzeugung und Speicherung von Versionen • Annotation jeder Version mit beschreibenden Attributen • Dynamische Zusammensetzung bestimmter *Komponentenversionen* zu einer Konfiguration (beispielsweise mittels eines Selektionsprädikats)	• *Ableitung neuer Versionen* durch Anwendung beliebiger Änderungsoperationen auf die Basisversion
Extensionale Versionierung	• Explizite Erzeugung und Speicherung von Versionen aller Komponenten • Dynamische Zusammensetzung bestimmter *Konfigurationsversionen* aus den entsprechenden Versionen einzelner Komponenten	• Explizite Erzeugung und Speicherung von Versionen aller Komponenten • Protokollierung aller erfolgten Änderungen (Änderungsoperationen) zur Erzeugung jeder Version • Verwendung der Änderungsinformationen für *Dokumentationszwecke* bzw. zur *Suche* nach bestimmten Versionen

Tabelle 7.1: Arten der Versionierung

vorgenommenen Änderungen beschrieben werden. Eine Angabe der Änderungsgründe ist beispielsweise durch Verknüpfungen zu den entsprechenden *Change Requests* möglich. Somit kann eine Version durch die *Change Requests*, welche sie umsetzt, repräsentiert werden.

Die Unterscheidungen zwischen der intensionalen und der extensionalen bzw. der änderungsorientierten und der zustandsorientierten Versionierung sind orthogonal zu einander. Eine kurze Erklärung zu jeder der vier möglichen Kombinationen findet sich in ***Tabelle 7.1***. Es ist anzumerken, dass die meisten existierenden Versionierungssysteme eine extensionale, zustandsorientierte Versionierung umsetzen.

Auch bei der Versionierung von Tuning-Plänen erscheint eine Entscheidung für die *extensionale Versionierung* sinnvoll. Die Versionen, die von einzelnen Nutzern des Community-Systems erzeugt werden, werden *explizit* erzeugt und ins Repository eingepflegt. Die Identifizierung der Versionen erfolgt mit Hilfe eines globalen Tuning-Plan-Identifiers.

In Abhängigkeit von der Integration des Tuning-Plan-Editors und des Versionierungssystems ist zu prüfen, ob eine extensionale zustandsorientierte oder extensionale änderungsorientierte Versionierung in Frage kommt. Die extensionale änderungsorientierte Versionierung hat dabei eine Reihe von Vorteilen, weswegen diese in weiteren Ausführungen präferiert wird.

Die *extensionale änderungsorientierte Versionierung* von Tuning-Plänen lässt sich wie folgt umsetzen: Durch einen Eingriff in den Tuning-Plan-Editor können Änderungsoperationen,

die bei der Erstellung einer neuen Revision des Tuning-Plans vorgenommen wurden, automatisch protokolliert und als zusätzliche Metadaten der Version zu Dokumentationszwecken abgespeichert werden (vgl. *Abschnitt 7.1.2.2*). Anhand von diesen Metadaten kann anschließend der Versionsvergleich erfolgen. Ist eine Erweiterung des Tuning-Plan-Editors dagegen nicht möglich, so können die vorgenommenen Änderungen durch Parsen und Vergleichen beider Versionen ermittelt werden (vgl. *Abschnitt 7.1.2.8*).

Auch bei der *extensionalen zustandsorientierten Versionierung* werden die Tuning-Pläne durch die Autoren explizit erzeugt. Bei der Verwaltung der Tuning-Plan-Versionen kann jedoch nicht auf die Unterschiede zwischen den Versionen zurückgegriffen werden. Stattdessen werden die Tuning-Plan-Versionen als nicht zerlegbare, atomare Einheiten betrachtet und entsprechend abgespeichert. Auf diese Art der Versionierung soll daher nur dann zurückgegriffen werden, wenn die Protokollierung bzw. die Bestimmung von Änderungsoperationen zu aufwendig oder explizit nicht erwünscht ist.

Eine der möglichen Erweiterungen des im Rahmen dieser Arbeit vorgestellten best-practices-orientierten Ansatzes ist eine automatische Erzeugung von Tuning-Plänen durch Komposition von vordefinierten, im Repository abgelegten, Tuning-Schritten und eine Anpassung dieser Tuning-Pläne zur Laufzeit. Zur Versionierung solcher automatisch erzeugten Tuning-Pläne eignet sich der *intensionale Ansatz*. Das Prädikat zur Erzeugung der Tuning-Plan-Version kann beispielsweise das identifizierte Problem beschreiben. Anhand der im System hinterlegten Regeln werden dabei Tuning-Pläne erzeugt, die das Problem lösen können. Die Aufgabe der Planungskomponente (*Abschnitt 5.2.4*) ist somit die Auswahl des optimalen Tuning-Plans.

7.1.2.2 Speicherung von Versionen

Während wir vorher nur Versionierungskonzepte betrachtet haben, die für Anwender der Versionierungssysteme direkt erkennbar sind (*externe Versionierung*), sollen in diesem Abschnitt die Konzepte zur internen Realisierung der Versionierung (*interne Versionierung*) betrachtet werden. Die Art der internen Versionierung bestimmt, wie die Anwender auf das Versionierungssystem zugreifen bzw. wo und wie die versionierten Daten gespeichert sind.

Nach [Leb95] kann zwischen drei Ansätzen zur Speicherung der Versionen unterschieden werden (*Wo* wird gespeichert). Die Versionen lassen sich in Vaults, in virtuellen Dateisystemen bzw. in Repositories speichern.

Vault. Als *Vault* (*engl.*, Kellergewölbe oder Tresor) wird ein bestimmter Bereich im Dateisystem bezeichnet, in welchem die Versionen abgelegt werden. Aus diesem Bereich werden die Versionen bei Bedarf in die Arbeitsbereiche der Anwender umkopiert. Die in einem *Vault* abgespeicherten Daten sind nicht von direkten Zugriffen (vorbei an der Schnittstelle des *Vaults*) der Anwender geschützt. Die Daten können daher versehentlich oder mutwillig beschädigt werden, was auch zu Inkonsistenzen zwischen der Version im *Vault* und den Versionen in Arbeitsbereichen führen kann. Diese Art der Speicherung wird von einfachen dateibasierten Versionierungssystemen wie *RCS* [Tic85] oder *CVS* [FB04] verwendet.

Virtuelles Dateisystem. Bei der Verwendung von *virtuellen Dateisystemen* erfolgt die Speicherung von Versionen auf einem separaten persistenten Speicher. Der Zugriff auf die Versionen erfolgt dabei über ein virtuelles Dateisystem. Die Verwaltung der Versionen ist dabei für die Anwender transparent und alle Änderungen an den gespeicherten Versionen werden vom Versionierungssystem überwacht. Das Versionierungssystem *ClearCase* [WBTT04, BM05] greift auf diese Art der Speicherung zurück.

Für die beiden soeben vorgestellten Verfahren werden keine speziellen Werkzeuge benötigt, da die meisten existierenden Werkzeuge auf die versionierten Daten zugreifen können, ohne etwas vom Versionierungssystem wissen zu müssen. Es wird jedoch viel Interaktion mit den Anwendern benötigt, was zugleich ein Nachteil dieser Verfahren ist. Die nach diesen beiden Ansätzen umgesetzte Versionierungssysteme kennen in der Regel keine interne Struktur der versionierten Daten und können daher beispielsweise kaum Unterstützung bei der Differenzbestimmung bieten.

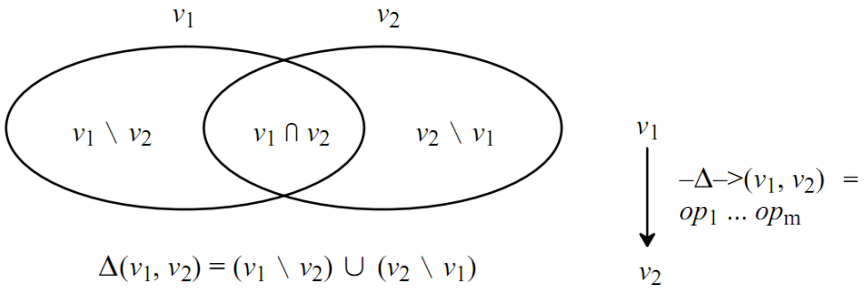
Repository. Im Gegensatz zu den beiden anderen Varianten bieten die *Repositories* eine Programmierschnittstelle (API) zum Datenzugriff. Die verwalteten Elemente mit allen ihren Versionen werden gekapselt und der Zugriff darauf kann ausschließlich über die API erfolgen. Dies ermöglicht zugleich eine stärkere Integration von Werkzeugen und Repositories. Dadurch bekommen die Werkzeuge mehr Informationen über die interne Struktur der versionierten Daten bzw. die Versionsgeschichte und können sogar den Versionierungsprozess automatisieren. Aufgrund der Vorteile dieser Speicherungsart werden in unserem Ansatz die Tuning-Pläne in einem Repository gespeichert. Auf eine weitere Betrachtung von *Vaults* bzw. virtuellen Dateisystemen wird stattdessen verzichtet.

Bei der Frage, *wie* die Versionen gespeichert werden, ist ein Kompromiss bezüglich der Speicherplatzverwendung und der Zugriffsgeschwindigkeit zu treffen. Einerseits benötigt eine kompakte Art der Speicherung von Daten mehr Zeit, um die Daten zu speichern bzw. zu rekonstruieren. Eine vollständige Speicherung der Versionen ermöglicht dagegen einen schnelleren Zugriff auf die Daten, benötigt jedoch mehr Speicherplatz.

Dokumentenbasierte Speicherung. Die dokumentenbasierte Versionsspeicherung impliziert eine Komplettspeicherung jeder einzelnen Version. Der Speicherplatzbedarf für diese Speicherungsart ist entsprechend hoch, weil selbst bei sich nur marginal unterscheidenden Versionen jeweils ihre komplette Speicherung erfolgt. Die Zugriffsgeschwindigkeit ist jedoch ebenfalls als hoch und konstant einzustufen.

Delta-Speicherung. Üblicherweise sind die Unterschiede zweier Versionen eines Dokuments nur lokal auf wenige Stellen begrenzt. Deshalb liegt es nahe, nicht jede Version komplett abzuspeichern, sondern die Gemeinsamkeiten auszunutzen und die Speicherung auf die Unterschiede zwischen den zwei Versionen zu beschränken. Alle Unterschiede zwischen zwei Versionen bilden ein *Delta*. Die Deltas können auf unterschiedliche Arten gebildet werden, am gebräuchlichsten sind byteweise (Vergleich auf Byte-Ebene), zeilenweise bei Textdateien (Vergleich auf Zeilenebene) und syntaxorientierte (Vergleich auf Ebene von syntaktischen Einheiten) Deltas.

Zur Repräsentation von Deltas lassen sich zwei Vorgehensweisen unterscheiden. Zum einen können Deltas mengenwertig und zum anderen operational repräsentiert werden [Ohs04]. Bei einer *mengenwertigen Repräsentation* von Deltas werden die Un-



(a) Symmetrisches Delta (b) Gerichtetes Delta

Abbildung 7.1: Symmetrisches und gerichtetes Delta [CW98]

terschiede zwischen zwei Versionen als eine Menge von Tupeln aus den Positionen und den inhaltlichen Unterschieden zwischen den Versionen beschrieben. Diese Darstellungsform entspricht einer zustandsbasierten Beschreibung der Unterschiede und ist auch als *symmetrisches Delta* bekannt (***Abbildung 7.1a***).

Die *operationale Repräsentation* eines Deltas kann dagegen als eine Sequenz von elementaren Änderungsoperationen $op_1, \ldots, op_m$ repräsentiert werden, durch deren Anwendung auf die Ausgangsversion v_1 eine abgeleitete Version v_2 erzeugt werden kann (***Abbildung 7.1b***). Da die Überführung von einer Version in die andere durch diese Sequenz von Änderungsoperationen immer nur in eine Richtung funktioniert, bezeichnet man diese Delta-Repräsentation auch als *gerichtetes Delta*. Diese Technik ist somit eine mögliche Art der internen Umsetzung der änderungsorientierten Versionierung (*Abschnitt 7.1.2.1*). Sie lässt sich idealerweise bei einer starken Integration der Werkzeuge und des Versionierungssystem einsetzen. Andernfalls ist die automatische Bestimmung der ausgeführten Operationen sehr aufwendig und nur unter Einschränkungen möglich (*Abschnitt 7.1.2.8*). Der Einsatz des gerichteten Deltas bringt einige Vorteile beispielsweise bei der Zusammenführung von Versionen bzw. bei der Benachrichtigung der Community über die erfolgten Änderungen (*Abschnitt 7.1.3*).

Gerichtete Deltas können sowohl die Änderung von einer alten zu einer neueren Version beschreiben (*Vorwärtsdelta*), wie auch die umgekehrt (*Rückwärtsdelta*). Um den Zugriff auf bestimmte Versionen zu beschleunigen, kann zusätzlich an wichtigen Stellen eine Vollspeicherung erfolgen. Grundsätzlich ist jedoch der Zugriff auf die mittels Vorwärts- bzw. Rückwärtsdeltas gespeicherten Versionen im Gegensatz zur dokumentenbasierten Versionsspeicherung relativ langsam. ***Abbildung 7.2*** illustriert die Verwendung von Vorwärts- und Rückwärtsdeltas in Kombination mit Vollspeicherungen [SR01].

Durch die Art der in dieser Arbeit betrachteten Community-Plattform ist zu erwarten, dass eine hohe Anzahl an ähnlichen Tuning-Plan-Versionen verwaltet werden muss, weshalb ein besonderes Augenmerk auf eine speicherplatzsparende Speicherung von Tuning-Plänen gerichtet wird. Dazu wird versucht, die Informationen über die Gemeinsamkeiten einzelner Versionen zu berücksichtigen.

Bei der Wahl der Art der internen Speicherung ist daher von einer ausschließlichen Verwendung einer dokumentenbasierten Vollspeicheurng von Tuning-Plan-Versionen abzusehen

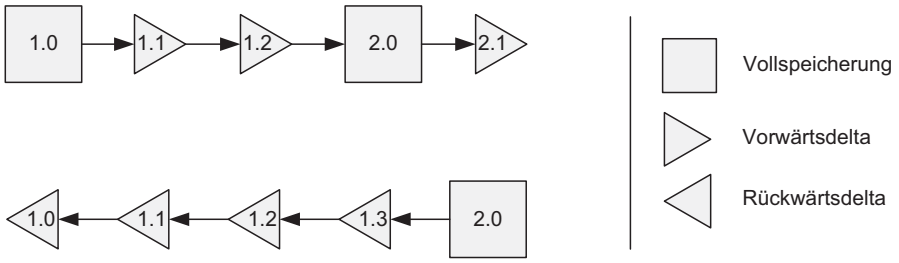


Abbildung 7.2: Vorwärts- und Rückwärtsdelta (in Anlehnung an [SR01])

und eine Delta-Speicherung zu bevorzugen. Hier ist zunächst ein Kompromiss zwischen einem effizienten lesenden, aber aufwendigen schreibenden (Rückwärtsdelta) und einem effizienten schreibenden aber aufwendigen lesenden Zugriff (Vorwärtsdelta) auf die letzte Revision eines Tuning-Plans zu finden.

Bei der Verwendung von *Rückwärtsdeltas* liegt die letzte Revision r_{last} jedes Tuning-Plans immer materialisiert vor (Vollspeicherung), was einen effizienten lesenden Zugriff darauf gewährleistet. Um eine neue Revision eines Tuning-Plans r_{new} anzulegen, muss zwischen dieser neuen Revision und der aktuell noch materialisierten Vorgängerrevision das Delta bestimmt werden. Anschließend muss die Vorgängerrevision r_{last} auf die Versionierungsdatenstruktur abgebildet werden, während die neue Revision r_{new} materialisiert wird. Ein weiterer wichtiger Nachteil ist die Notwendigkeit einer „Spiegelung“ der protokollierten oder gegebenenfalls ermittelten Änderungsoperationen. Ist eine Revision r_2 durch Hinzufügen eines Tuning-Schrittes t_1 und einer entsprechenden Kante k_1 aus der Revision r_1 entstanden, so muss nun die Revision r_1 basierend auf der Revision r_2 gespeichert werden (r_1 ergibt sich durch Entfernen des Tuning-Schrittes t_1 und der Kante k_1 ausgehend von r_2).

Bei der Verwendung eines *Vorwärtsdeltas* muss die neuste Revision eines Tuning-Plans durch die Anwendung einer Sequenz von Änderungsoperationen auf die aktuellste materialisierte Revision ermittelt werden. Durch erzwungene Materialisierungen von einigen Revisionen kann die Performance des lesenden Zugriffs gesteigert werden, da dadurch die Sequenz der anzuwendenden Änderungsoperationen verkürzt wird. In ***Abbildung 7.3*** erkennt man beispielsweise, dass die letzte Revision von Tuning-Plan *TP2* (Revision *2.4*) aus der Revision *2.1* durch die Anwendung von Änderungsoperationen $c7$, $c8$, $c9$, $c10$, $c11$ und $c14$ erzeugt werden muss, während die letzte Revision *1.5* des Tuning-Plans *TP1* aus der materialisierten Revision *1.4* durch die Anwendung von Änderungsoperationen $c12$ und $c13$ ermittelt werden kann. Bei einem Zugriff auf eine ältere Revision ist zunächst die letzte materialisierte Revision *vor* der gesuchten Revision zu bestimmen. Ausgehend von dieser Revision erfolgt dann die Erzeugung der gesuchten Revision. So kann die Revision *2.2* durch die Anwendung der Änderungsoperationen $c7$, $c8$ und $c9$ direkt aus der materialisierten Vorgängerrevision *2.1* ermittelt werden.

In Anbetracht der Tatsache, dass viele schreibende Zugriffe auf das Repository zu erwarten sind und künftig eine starke Integration des Tuning-Plan-Editors und des Repositorys zur Ermöglichung einer Protokollierung der im Tuning-Plan-Editor vorgenommenen Änderun-

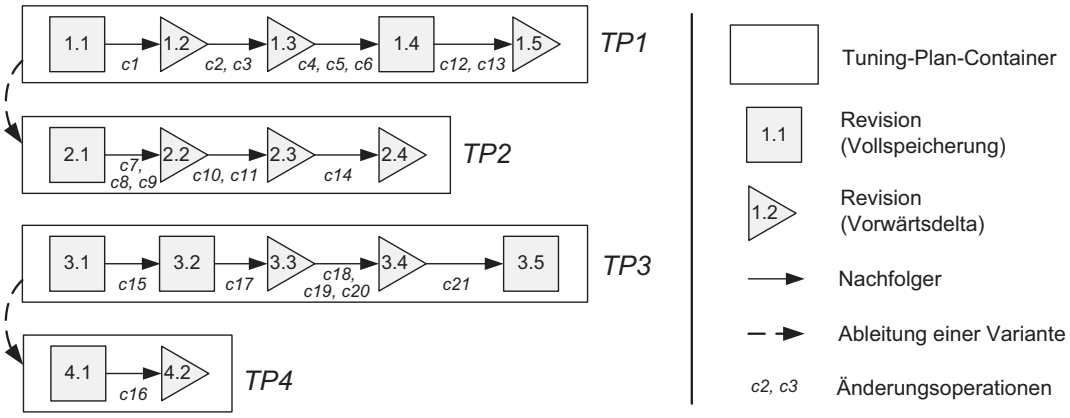


Abbildung 7.3: Versionierung von Tuning-Plänen – Versionsgraph

gen angestrebt werden soll[30], ist auf die Verwendung von Rückwärtsdeltas zu verzichten. Stattdessen soll die interne Speicherung von Tuning-Plänen unter Zuhilfenahme von Vorwärtsdeltas mit den in regelmäßigen Abständen platzierten, materialisierten Revisionen durchgeführt werden.

Als kleinste atomare Einheiten für die Delta-Speicherung (*Delta-Granulat*) sollen Tuning-Schritte bzw. Kanten, die den Kontrollfluss umsetzen, verwendet werden. Für die Tuning-Schritt- sowie Kanten-Änderungen sind daher entsprechende Änderungsoperationen zu definieren (*Abschnitt 7.1.2.5*).

7.1.2.3 Tuning-Plan-Container, -Revisionen und -Varianten

Um eine Versionierung der Tuning-Pläne zu ermöglichen, werden zunächst für jeden zur Lösung einer bestimmten Problemklasse erstellten Tuning-Plan ein entsprechendes virtuelles Versionierungsobjekt namens *Tuning-Plan-Container* sowie eine erste Revision dieses Tuning-Plans angelegt. Bei einer Weiterentwicklung dieses Tuning-Plans lassen sich nun entweder neue Revisionen (bei Funktionalitätserweiterung bzw. Fehlerkorrekturen) oder neue Varianten (bei Umsetzung von alternativen Tuning-Plänen, die beispielsweise auf eine andere Systemumgebung zugeschnitten sind) anlegen. Neue Revisionen werden dabei im gleichen Tuning-Plan-Container angelegt. Die von diesem Tuning-Plan abgeleiteten Varianten werden stattdessen als neue Tuning-Plan-Container angelegt, die wiederum eigene Revisionen beinhalten. Die erste Revision eines Tuning-Plan-Containers wird dabei stets materialisiert gespeichert (Vollspeicherung). Dieser Zusammenhang ist in *Abbildung 7.3* veranschaulicht.

Die Zusammenfassung aller Revisionen eines Tuning-Plans zu einem generischen Objekt – dem virtuellen Tuning-Plan-Container – ermöglicht beispielsweise eine effiziente Verwaltung von bestimmten, für alle Revisionen eines Tuning-Plans geltenden Metadaten wie Tuning-Plan-Name bzw. -Beschreibung (*Abschnitt 7.1.6*) oder Beziehungen (beispielsweise zwischen dem Vater-Tuning-Plan-Container und den davon abgeleiteten Varianten).

[30] Die Integration des Tuning-Plan-Editors und des Repositorys ist jedoch nicht Gegenstand der vorliegenden Arbeit.

Anstatt solche Informationen an jede Revision zu knüpfen, erfolgen solche Verknüpfungen an die entsprechenden Tuning-Plan-Container.

Beim Zugriff auf den Tuning-Plan-Container kann eine dynamische Auswahl der betroffenen Tuning-Plan-Revision nach einem bestimmten Kriterium, das die auszuwählende Revision beschreibt, erfolgen. So lässt sich beispielsweise die aktuelle Revision bestimmen (vgl. *Abschnitt 7.1.2*) oder die neueste Revision im Zustand *released* (vgl. *Abschnitt 7.1.5.5*).

7.1.2.4 Ein Datenmodell zur Speicherung von Tuning-Plänen

Wie bereits in *Abschnitt 4.5.2* erklärt wurde, erfolgt in unserem Prototyp die Modellierung von Tuning-Plänen mit *IBM WebSphere sMash (WsM)*. Entsprechend wird auch im vorliegenden Kapitel zunächst davon ausgegangen, dass die Tuning-Pläne im WsM-spezifischen XML-Format vorliegen. Dadurch orientieren sich zwar die nachfolgenden Ausführungen an WsM-spezifischen Konzepten und Begriffen, die hier vorgestellten Datenmodelle zur Speicherung bzw. Versionierung von Tuning-Plänen sind jedoch universell und zur Verwaltung bzw. Versionierung von Tuning-Abläufen in beliebigen Formaten geeignet.

Der in diesem Abschnitt vorgestellte Ansatz ermöglicht eine effiziente, semantikreiche Speicherung von Tuning-Plänen im Community-Repository, indem Tuning-Pläne in ihre Basiskonstrukte wie Tuning-Schritte bzw. Kontrollstrukturen aufgeteilt und anschließend in einem entsprechenden Datenmodell abgespeichert werden. Das Datenmodell basiert dabei auf den im Rahmen unseres Projekts in [Krü09] durchgeführten Untersuchungen.

Diese Vorgehensweise setzt nicht nur eine effiziente Speicherung von Tuning-Plänen unter Ausnutzung der Informationen über ihre Ähnlichkeiten um, sondern bietet insbesondere auch einen Einblick in die Tuning-Plan-Semantik durch eine explizite Verwaltung einzelner Tuning-Plan-Basiskonstrukte. Zu einem weiteren wichtigen Vorteil dieses Ansatzes zählt die hohe Flexibilität bezüglich der eingesetzten Formate der Tuning-Pläne. Durch eine Speicherung von Tuning-Plänen in einem Werkzeug-unabhängigen Format und Bereitstellung verschiedener Zerlegungs- bzw. Zusammensetzungsalgorithmen können im vorliegenden Datenmodell Tuning-Pläne in beliebigen Quellformaten abgespeichert und anschließend mit beliebigen Ausführungswerkzeugen abgearbeitet werden. Von dieser Flexibilität kann insbesondere in einer Community profitiert werden, deren Mitglieder verschiedene Tuning-Plan-Modellierungs- und -Ausführungswerkzeuge einsetzen und dennoch einen werkzeugübergreifenden Tuning-Plan-Austausch anstreben.

In ***Abbildung 7.4*** findet sich das *Entity/Relationship-Diagramm (E/R-Diagramm)* des Datenmodells, welches eine feingranulare Speicherung von Tuning-Plänen unterstützt. Zur Wahrung der Übersichtlichkeit wurde im Diagramm auf die Darstellung einiger Attribute verzichtet. Sofern in den nachfolgenden Abschnitten Bezug auf dieses E/R-Diagramm genommen wird und weitere Attribute zur Umsetzung bestimmter Konzepte benötigt werden, wird explizit darauf hingewiesen.

Des Weiteren ist zu erwähnen, dass im vorliegenden E/R-Diagramm das Konzept der schwachen Entitätstypen Anwendung fand. Diese wurden dabei durch doppelt gerahmte Rechtecke repräsentiert und ihre Beziehung zum übergeordneten Entitätstyp durch eine doppelt umrahmte Raute markiert.

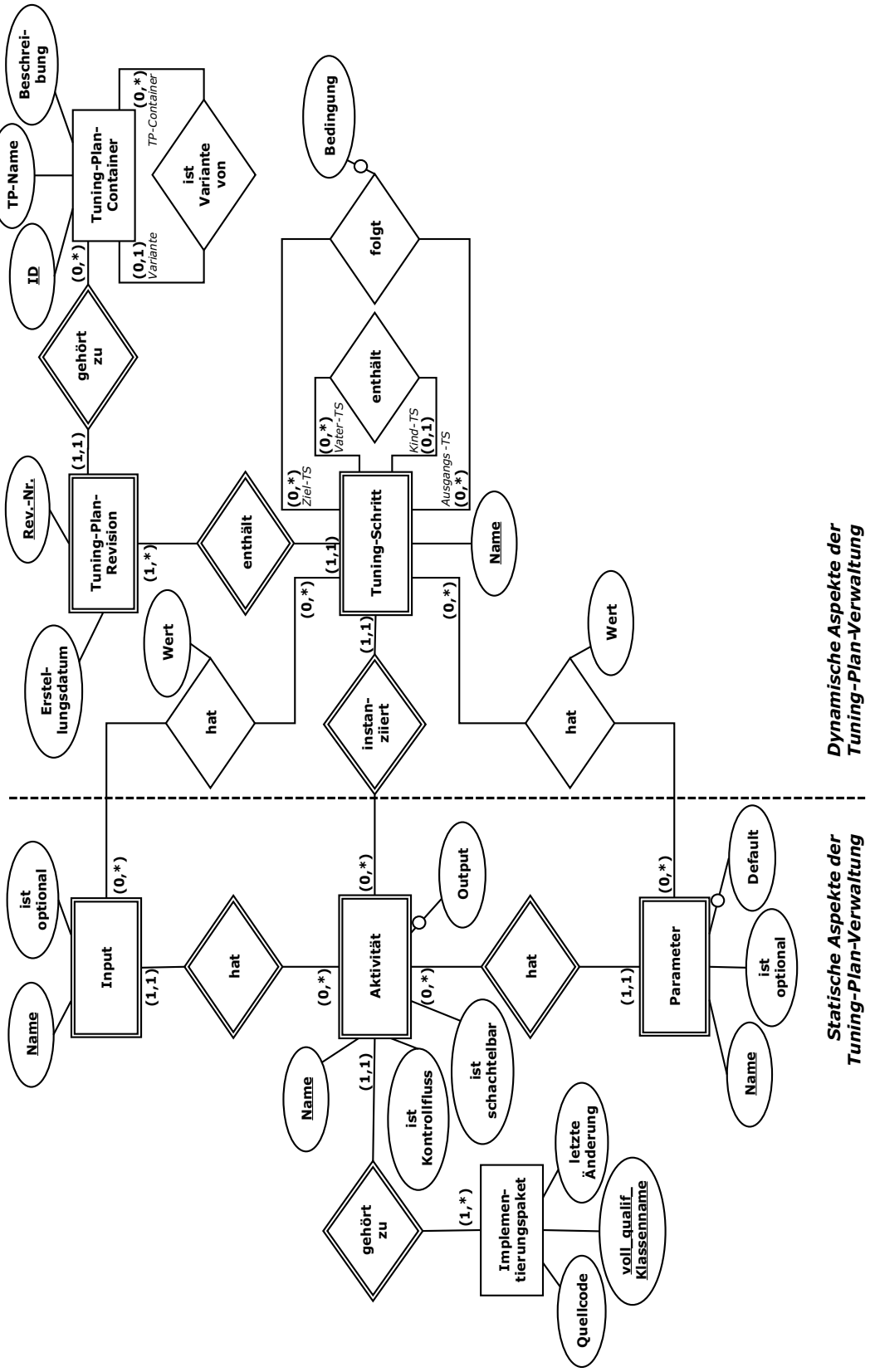


Abbildung 7.4: Datenmodell zur feingranularen Verwaltung und Speicherung von Tuning-Plänen

Das vorliegende E/R-Diagramm ist in zwei Bereiche eingeteilt. Die *statischen Aspekte der Tuning-Plan-Verwaltung* umfassen die Tuning-Aktivitäten (vgl. *Abschnitt 4.4.1*) mit dem zugehörigen Implementierungsquellcode sowie die entsprechenden Parameter und Inputs (vgl. *Abschnitt 4.5.2*). Diese Daten bleiben typischerweise zur Systemlaufzeit unverändert. Eine Ausnahme bilden dabei lediglich das Hinzufügen neuer bzw. das Verändern oder Löschen existierender Aktivitäten, sofern die Mächtigkeit der bereitgestellten Tuning-Aktivitäten nicht ausreicht.

Die *dynamischen Aspekte der Tuning-Plan-Verwaltung* umfassen die Tuning-Pläne mit allen ihren Basisbausteinen. Dazu gehören die Tuning-Schritte (Instanzen vordefinierter Tuning-Aktivitäten), die instantiierten Parameter bzw. Inputs sowie die den Kontrollfluss repräsentierenden Kanten. Bei der Erstellung von neuen Tuning-Plänen werden also lediglich die vordefinierten Tuning-Aktivitäten instantiiert, parametrisiert und mit Hilfe von Kontrollstrukturen, die sich schließlich auf Kanten mit entsprechenden Bedingungen abbilden lassen, miteinander zu Tuning-Plänen verknüpft. Hier sind also hauptsächlich Einfügungen (bei der Erstellung von neuen Tuning-Plänen) bzw. lesender Zugriff (beim Auslesen von gespeicherten Tuning-Plänen) zu erwarten.

Tuning-Plan-Container, Revisionen, Varianten. Ein Tuning-Plan-Container ist ein generisches Objekt, das alle Revisionen eines Tuning-Plans zusammenfasst. Diesem Tuning-Plan-Container lassen sich beispielsweise die Tuning-Plan-spezifischen Metadaten zuordnen. Tuning-Plan-Revisionen werden aufsteigend durchnummeriert, was indirekt den Zeitaspekt ihrer Entstehung repräsentiert: Die Tuning-Plan-Revision mit der höchsten Nummer ist die letzte Revision. Von einzelnen Tuning-Plänen lassen sich neue Tuning-Plan-Varianten ableiten. In diesem Fall erfolgt eine Protokollierung dieser Beziehung zwischen den beiden Tuning-Plan-Containern (Beziehungstyp *ist Variante von*). Wird später die Information benötigt, von welcher konkreten Tuning-Plan-Revision die Variante abgeleitet wurde, so kann dies durch einen Vergleich des Zeitstempels der ersten Revision des abgeleiteten Tuning-Plan-Containers mit den Zeitstempeln aller Revisionen des Vater-Tuning-Plan-Containers ermittelt werden.

Aktivitäten und Tuning-Schritte. Durch Aktivitäten werden Basisaktionen repräsentiert, die einem Datenbankadministrator zur Verfügung stehen und in den Tuning-Plänen verwendet werden können. Der die gewünschte Funktionalität der Aktivitäten umsetzende Java-Code wird in den entsprechenden Java-Klassen abgelegt. Dabei kann der Quellcode mehrerer Aktivitäten in einer Klasse abgelegt werden (Entitätstyp *Implementierungspaket*). Durch ein Mitführen des Zeitpunktes der letzten Änderung eines Implementierungspakets wird eine vereinfachte Versionierung der (unter Umständen von Nutzern modifizierten) Aktivitäten umgesetzt.

Wie bereits bekannt, werden instantiierte Aktivitäten als Tuning-Schritte bezeichnet. Diese lassen sich mit geringem Aufwand mittels Kontrollstrukturen wie Sequenzen, Verzweigungen oder Schleifen zu neuen Tuning-Plänen verknüpfen. Durch eine geeignete Namensgebung der Aktivitäten ist es außerdem möglich, Rückschlüsse auf ihre Semantik und somit die Semantik der gesamten Tuning-Pläne zu ziehen (vgl. *Abschnitt 4.4.1*).

Neben den Aktivitäten, die für die Umsetzung von Datenbank-Tuning-spezifischen Basisaktionen verantwortlich sind, existieren auch Kontrollfluss-spezifische Aktivi-

täten wie beispielsweise das WsM-Schleifenkonstrukt `for-each` (*Abschnitt 4.5.2*). Diese werden durch das entsprechende Attribut *ist kontrollfluss* gekennzeichnet. Des Weiteren lässt sich zwischen atomaren und schachtelbaren Aktivitäten unterscheiden (Attribut *ist schachtelbar*). Mit Hilfe von schachtelbaren Aktivitäten lassen sich beliebige Aktivitäten zusammenfassen. Typischerweise sind schachtelbare Aktivitäten Kontrollfluss-spezifisch.

Parameter und Inputs. Der Austausch von Daten zwischen den einzelnen Aktivitäten kann mit Hilfe der Ausgabe- sowie Eingabeparametern von Aktivitäten bzw. über den *Global Context* (*Abschnitt 4.5.2*) erfolgen.

Jeder Tuning-Schritt bekommt einen implizit erzeugten Ausgabeparameter, der den Namen dieses Tuning-Schritts trägt, einen aus dem Rückgabewert automatisch ermittelten Datentyp besitzt und einen ausschließlich lesenden Zugriff auf den Variableninhalt durch andere Tuning-Schritte erlaubt.

Die Eingabeparameter der Aktivitäten müssen dagegen explizit definiert werden [IBM11]. Dabei lässt sich zwischen Parametern und Inputs unterscheiden. Die Inputs werden mit dem XML-Kindelement `input` definiert und haben einen Namen, einen konstanten bzw. aus dem Ausgabewert eines Vorgänger-Tuning-Schritts ermittelten Wert und einen entsprechenden aus dem Wert abgeleiteten Datentyp (vgl. *Abschnitt 4.5.2*).

Im Gegensatz zu den Inputs können Parameter ausschließlich Werte primitiver Datentypen aufnehmen und sind primär für eine (zum Programmierzeitpunkt vorgesehene) Parametrisierung von Aktivitäten gedacht. Sie werden als Attribute des die Aktivität repräsentierenden XML-Elements definiert.

Kontrollfluss. Die Reihenfolge der Ausführung von Aktivitäten wird durch den Kontrollfluss festgelegt. Dieser lässt sich durch Erstellen von Kanten zwischen den Tuning-Schritten und gegebenenfalls Versehen dieser mit Bedingungen bzw. durch Kapseln von Tuning-Schritten in schachtelbare, Kontrollfluss-spezifische Tuning-Schritte definieren.

7.1.2.5 Ein Datenmodell zur Versionierung von Tuning-Plänen

Nachdem im *Abschnitt 7.1.2.4* ein Modell zur semantikreichen, feingranularen Speicherung von Tuning-Plänen in einer Datenbank vorgestellt wurde, wird dieses Modell im aktuellen Abschnitt entsprechend erweitert, um eine änderungsorientierte Versionierung der Tuning-Pläne sowie eine speicherplatzsparende Delta-Speicherung der Tuning-Plan-Revisionen zu ermöglichen. Dabei wird die Erfassung der erfolgten Änderungen auf der Ebene von durch die Änderungen betroffenen Tuning-Plan-Basiskonstrukten vorgenommen. Dazu werden alle möglichen Änderungsoperationen identifiziert und im Datenmodell berücksichtigt. Durch eine, damit ermöglichte, semantische Erfassung von Änderungen wird unter anderem ihre Nachvollziehbarkeit für die Nutzer gewährleistet. Einen anderen Anwendungsgebiet einer solchen Delta-Speicherung bietet das Nachfahren von Sequenzen von Änderungsoperationen auf parallelen Entwicklungszweigen (also anderen Varianten des Tuning-Plans), um damit beispielsweise die auf einem Entwicklungszweig stattgefundenen Fehlerkorrekturen auf andere Entwicklungszweige zu übertragen. Aber auch bei der Zusammenführung von Versionen bzw. bei der Benachrichtigung der Community über

die erfolgten Änderungen bietet das vorliegende Versionierungsmodell Vorteile (vgl. *Abschnitt 7.1.2.2*).

Ein relevanter Ausschnitt aus dem entsprechenden Datenmodell findet sich in ***Abbildung 7.5***. Dabei ist das vorliegende Diagramm als eine Ergänzung zum E/R-Diagramm aus *Abbildung 7.4* zu verstehen. Aus diesem Grund wurde hier auch auf eine Angabe von Attributen für Entitätstypen, die bereits in *Abbildung 7.4* enthalten sind, verzichtet.

Neben einer Einführung von neuen Entitäts- und Beziehungstypen beinhaltet die *Abbildung 7.5* eine kleine Modifikation des Datenmodells aus der *Abbildung 7.4*: Zum einen wurde hier der Entitätstyp *TP-Revisions-Container* hinzugefügt, der alle Revisionen eines Tuning-Plans zusammenfasst. Zum anderen erfolgt eine Unterscheidung zwischen den Rollback-Referenzen (Entitätstyp *Rollback-Referenz*) und den Tuning-Plan-Revisionen (Entitätstyp *Tuning-Plan-Revision*). Die Tuning-Plan-Revisionen lassen sich weiterhin in *versionierte* bzw. *materialisierte*, also im Datenmodell aus *Abschnitt 7.1.2.4* gespeicherte, Tuning-Plan-Revisionen unterteilen (Entitätstypen *versionierte TP-Revision* bzw. *materialisierte TP-Revision*). Da die vorgenommenen Modifikationen primär zur Umsetzung einer Versionierung eingeführt wurden, soll darauf ausschließlich in diesem Abschnitt Bezug genommen werden. In den nachfolgenden Abschnitten, wo die Versionierung von Tuning-Plänen keine oder nur eine untergeordnete Rolle spielt, wird aus Aufwandsgründen von dem Tuning-Plan-Speicherungsmodell aus *Abbildung 7.4* ausgegangen.

Änderungsoperationen. Die Basis für das Versionierungsmodell sollen die Änderungsoperationen bilden, auf Sequenzen welcher alle Veränderungen von Tuning-Plänen zurückgeführt werden können. Die Operationen decken dabei alle möglichen Veränderungen an den Tuning-Schritten, Inputs, Parametern und Kanten – den kleinsten atomaren Einheiten der Tuning-Pläne – ab und lassen sich wie folgt zusammenfassen:

- Hinzufügen eines Tuning-Schritts
- Löschen eines Tuning-Schritts
- Umhängen eines geschachtelten Tuning-Schritts (Ändern des Tuning-Schritt-Vaters)
- Ändern eines Parameterwerts (Ändern eines existierenden Parameterwerts bzw. Setzen eines Parameterwerts, falls kein Wert gesetzt)
- Ändern eines Input-Werts (analog der Parameterwert-Änderung)
- Hinzufügen einer Kante
- Löschen einer Kante
- Ändern einer Kante (Ändern des Ausgangs- sowie Ziel-Tuning-Schritts und der Kantenbedingung)

Es fällt auf, dass keine Operation für die Änderung von Tuning-Schritten definiert wurde. Dies liegt in der Tatsache begründet, dass Tuning-Schritte Instanzen von Aktivitäten sind und daher nicht geändert werden können. Es können daher lediglich Tuning-Schritte gelöscht und anschließend durch neue Tuning-Schritte (Instanzen *anderer* Aktivitäten) ersetzt werden.

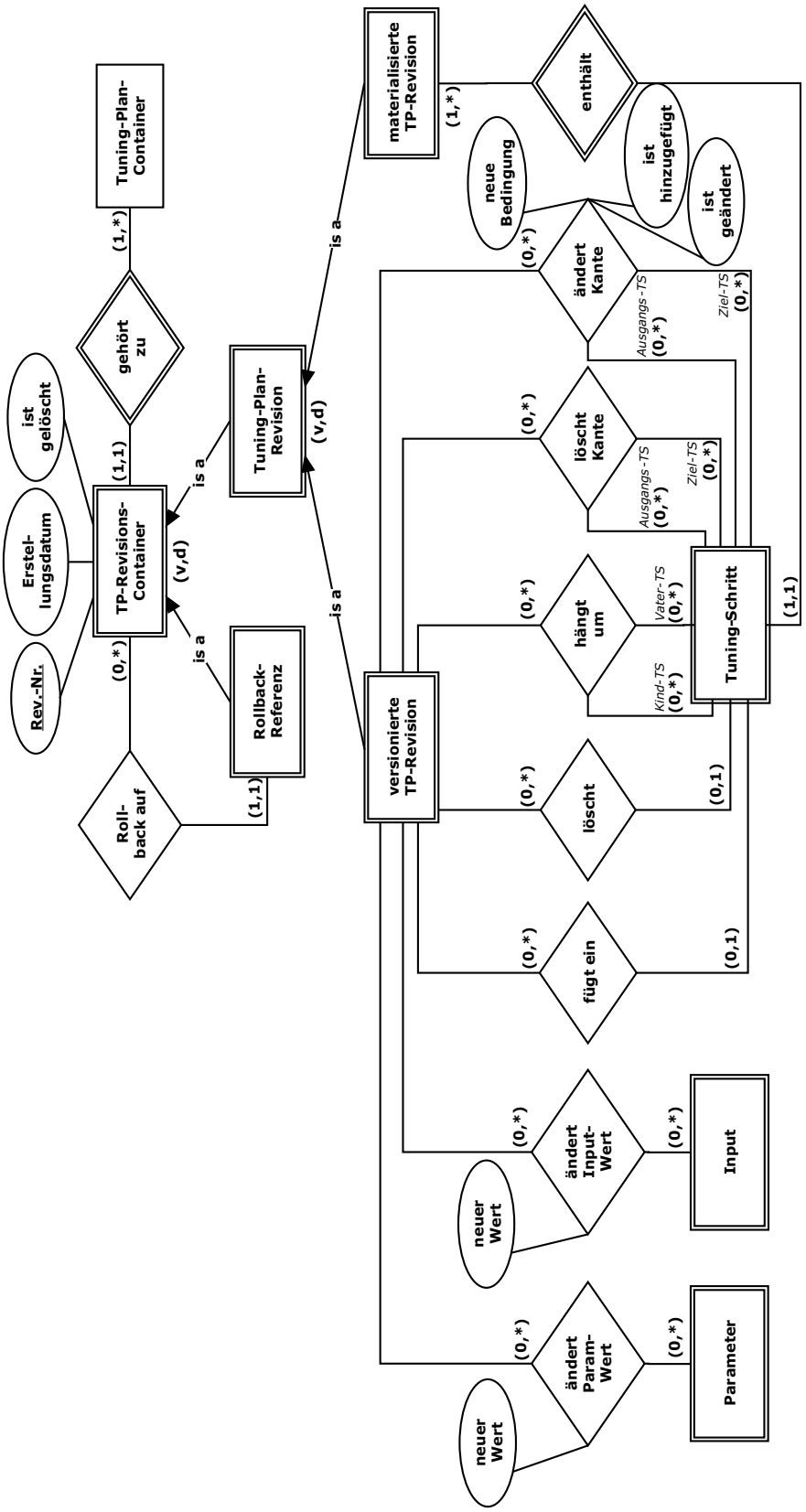


Abbildung 7.5: Datenmodell zur Versionierung von Tuning-Plänen – Ein Ausschnitt

Löschung von Parameter- sowie Input-Werten ist ebenfalls nicht notwendig, da diese innerhalb von Tuning-Schritten geschachtelt werden und nicht separat gelöscht bzw. hinzugefügt werden können. Neue Parameter- bzw. Input-Werte können ausschließlich beim Hinzufügen von neuen Tuning-Schritten angegeben werden, was wir durch die Operation zur Änderung von Parameter- bzw. Input-Werten umsetzen.

Im E/R-Diagramm in *Abbildung 7.5* wurden die soeben vorgestellten Änderungsoperationen als entsprechende Beziehungstypen umgesetzt. Aufgrund starker Ähnlichkeit zwischen den Operationen *„Hinzufügen einer Kante“* und *„Ändern einer Kante“* wurden sie auf einen einzigen Beziehungstyp *ändert Kante* zurückgeführt. Mit Hilfe von entsprechenden Flag-Attributen (*ist hinzugefügt* bzw. *ist bearbeitet*) erfolgt eine Unterscheidung, ob die Kante mittels einer Hinzufügung bzw. einer Änderung entstanden ist.

Bedingt durch die Funktionsweise des *WebSphere sMash* können keine IDs für die Tuning-Plan-Objekte angegeben werden. Die Identifizierung der Objekte erfolgt basierend auf den Objektnamen. Aus diesem Grund berücksichtigt auch unser Ansatz keine Namensänderungen der Tuning-Plan-Objekte.

Rollback. Als besonders wichtig wurde die Möglichkeit des Zurückkehrens auf eine ältere Revision (*Rollback*) eingestuft. Die klassische Umsetzung eines Rollbacks von der Revision r_j auf eine ältere Revision r_i impliziert ein Verwerfen aller Änderungen nach dem Zeitpunkt des Erstellens dieser Revision r_i, was einer Löschung aller Nachfolgerevisionen von r_i entspricht. Für unsere Zwecke ist eine solche Umsetzung jedoch nicht ausreichend, da wir in der Lage sein wollen, ein fälschlicherweise erfolgtes Rollback wieder rückgängig zu machen („Rollback des Rollbacks“). Aus diesem Grund werden Rollback-Referenzen eingeführt. Diese ermöglichen eine Verlinkung von der letzten Revision auf eine ältere Revision, ohne dabei alle Zwischenrevisionen zu löschen.

Durch die Verwendung dieses Ansatzes ergibt sich eine gewisse Einschränkung im Arbeitsablauf: Soll eine ältere (und nicht die letzte) Revision weiterentwickelt werden, so muss diese zunächst zur letzten und zugleich aktuellen Revision des Tuning-Plans gemacht werden, indem eine neue Revision erstellt wird, die eine Rollback-Referenz auf diese ältere Revision enthält. Zur Steigerung der Nebenläufigkeit im Mehrbenutzerbetrieb (*Abschnitt 7.1.5*) kann dabei der Zeitpunkt des Anlegens der Rollback-Referenz von der Anwendung bestimmt werden. So kann die neue Revision mit einer Rollback-Referenz auf eine ältere Revision unmittelbar vor der Speicherung der aus dieser älteren Revision abgeleiteten neuen Revision erstellt werden. Dadurch wird sichergestellt, dass der zwischenzeitlich erfolgte Rollback für andere Nutzer nicht sichtbar ist. Die Möglichkeit, von älteren Revisionen neue Tuning-Plan-Varianten abzuleiten, bleibt von dieser Einschränkung jedoch unberührt.

Abbildung 7.6 zeigt die Verwendung von Rollback-Referenzen anhand des Beispiel-Tuning-Plans *TP2* aus der *Abbildung 7.3*. Dabei wurde auf die Darstellung der Speicherungsart (Vollspeicherung bzw. Vorwärtsdelta) sowie des Tuning-Plan-Containers verzichtet. Im veranschaulichten Beispiel wurde zunächst ein Rollback auf die Revision *2.3* vorgenommen. Dazu wurde eine neue Revision *2.5* angelegt, die mittels einer Rollback-Referenz auf die Revision *2.3* verweist. Somit entspricht die Revision *2.5* der Revision *2.3*. Da die Revision *2.5* dabei lediglich zur Umsetzung

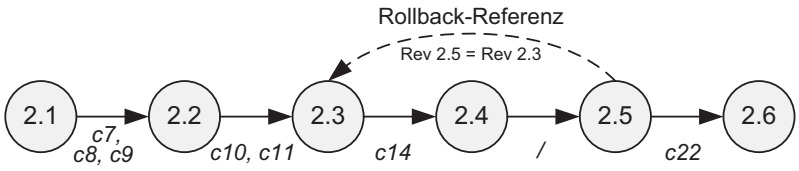


Abbildung 7.6: Verwendung von Rollback-Referenzen

eines Rollbacks auf die Revision *2.3* erzeugt wurde und beim Rollback keine Änderungsoperationen protokolliert bzw. berechnet werden, wurde dies im Diagramm durch ein „/“ gekennzeichnet.

Anschließend wurde aus der Revision *2.5* eine neue Revision *2.6* durch die Anwendung der Änderungsoperation $c22$ erstellt. Die Revision *2.6* lässt sich nun aus der Revision *2.1* durch die Anwendung von Änderungsoperationen $c7$, $c8$, $c9$, $c10$, $c11$ und $c22$ ableiten. Will man nun alle in diesem Beispiel stattgefundenen Änderungen seit der Revision *2.4* verwerfen und darauf zurückkehren, so ist eine neue Revision *2.7* anzulegen und mit einer Rollback-Referenz auf die Revision *2.4* zu versehen.

Zur Unterscheidung zwischen den Rollback-Referenzen und den „tatsächlichen“ Tuning-Plan-Revisionen wurden in *Abbildung 7.5*, wie eingangs erwähnt, der Entitätstyp *TP-Revisions-Container* sowie zwei davon abgeleitete Entitätstypen *Rollback-Referenz* und *Tuning-Plan-Revision* eingeführt. Bei der Ableitung handelt es sich um eine vollständige und disjunkte Spezialisierung, was im Diagramm durch ein „*(v,d)*“ gekennzeichnet wurde.

Löschung von Tuning-Plan-Revisionen. Da es sich hier um ein Community-System handelt, wo die Tuning-Pläne von einer Community weiterentwickelt werden, ist nicht auszuschließen, dass einige vorgenommenen Aktionen zu einem späteren Zeitpunkt wieder rückgängig gemacht werden müssen. Aus diesem Grund soll unter anderem auch auf Löschungen von Tuning-Plänen verzichtet werden.

Während eine Löschung von ganzen Tuning-Plan-Containern samt ihrem Inhalt verboten werden kann, sind Löschungen von einzelnen Tuning-Plan-Revisionen unter Umständen dennoch zu erlauben. Gründe für solche Löschungen können beispielsweise Copyright-Verletzungen sein. Durch die Verwendung bestimmter Algorithmen oder Technologien können einzelne Revisionen von solchen Copyright-Verletzungen betroffen sein. Die entsprechenden Tuning-Plan-Revisionen könnten in diesen Fällen zwar gelöscht werden, dies wurde jedoch die spätere Nachvollziehbarkeit der vorgenommenen Änderungen beeinflussen. Desweiteren müsste die erste u.U. bereits entstandenen Nachfolgerevision, die von der Copyright-Verletzung nicht betroffen ist, materialisiert werden, da eine Zusammensetzung dieser anhand von Änderungsoperationen nicht mehr möglich wäre.

Stattdessen werden die zu löschenden Revisionen in Analogie zum in [RCR94] skizzierten Ansatz durch ein entsprechendes Flag als „gelöscht“ markiert und damit deaktiviert (Attribut *ist gelöscht* am Entitätstyp *TP-Revisions-Container*). Deaktivierte Revisionen sind für die Nutzer nicht mehr sichtbar bzw. verwendbar. Die u.U.

in der Community bereits verteilten Revisionen des Tuning-Plans sind nach ihrer Deaktivierung automatisch durch alternative Tuning-Plan-Revisionen zu ersetzen.

Materialiserung bzw. Versionierung von Tuning-Plan-Revisionen. Wie bereits in *Abschnitt 7.1.2.2* erwähnt wurde, wird in dieser Arbeit eine auf Vorwärtsdeltas basierende Speicherung von Tuning-Plänen favorisiert. Zur Steigerung der Performance beim lesenden Zugriff auf die Tuning-Plan-Revisionen erfolgt dabei eine periodische Materialisierung von einzelnen Tuning-Plan-Revisionen.

Die erste Revision eines Tuning-Plans wird immer materialisiert. Dabei erfolgt die Speicherung der Revision entsprechend den Ausführungen aus *Abschnitt 7.1.2.4*. Diese Revision bildet die Grundlage für die nachfolgenden Revisionen, bis eine der Nachfolgerevisionen wieder materialisiert wird. Für die im Zuge einer Weiterentwicklung des Tuning-Plans entstandenen Revisionen erfolgt zunächst eine Ermittlung der erfolgten Änderungsoperationen. Dies kann beispielsweise durch eine Protokollierung bzw. Versionenvergleich umgesetzt werden (*Abschnitt 7.1.2.8*). Anschließend erfolgt die Abbildung dieser Änderungsoperationen auf die entsprechenden Beziehungstypen, wie in *Abbildung 7.5* zu sehen.

7.1.2.6 Zerlegung bzw. Zusammensetzung von materialisierten Tuning-Plänen

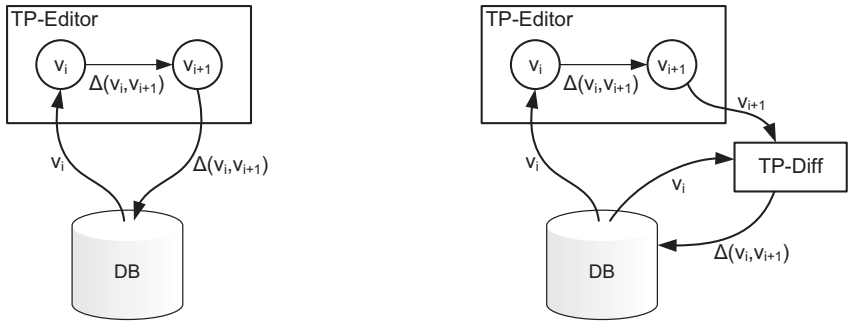
Um eine Verwendung der in vorigen Abschnitten vorgestellten Datenstrukturen zur Speicherung (*Abschnitt 7.1.2.4*) sowie Versionierung von Tuning-Plänen (*Abschnitt 7.1.2.5*) zu ermöglichen, wurde in [Krü09] eine entsprechende Mapper-Komponente entwickelt. Ihre Aufgaben sind:

- Zerlegung von Tuning-Plänen, die in einem speziellen XML-basierten, WsM-spezifischen Format vorliegen, in Basisobjekte (Tuning-Schritte, Inputs, Parameter, Kanten) mit anschließender Speicherung dieser Objekte in der in *Abschnitt 7.1.2.4* vorgestellten Datenstruktur
- Ermittlung aller für die Zusammensetzung eines Tuning-Plans notwendigen Daten aus der Datenstruktur mit anschließender Generierung einer entsprechenden WsM-spezifischen XML-Datei

Da es sich bei den umgesetzten Algorithmen primär um Parsen bzw. Generieren von XML-Dateien handelt, wurde auf ihre genauere Beschreibung verzichtet. Details finden sich in [Krü09].

7.1.2.7 Zusammensetzung von versionierten Revisionen

Bei der Zusammensetzung versionierter Tuning-Plan-Revisionen wird zunächst die letzte materialisierte Revision, die vor der zu generierenden Revision liegt, bestimmt. Diese wird dann aus der Datenstruktur ausgelesen. Anschließend erfolgt ein Nachfahren aller darauf erfolgten Änderungsoperationen, bis der Zustand der angefragten Revision erreicht wird. Der Tuning-Plan in angefragter Version ist dann nur noch in das WsM-spezifische XML-Format umzuwandeln und auszugeben. Auch hier wird auf die Vorstellung von weiterführenden Details im Rahmen dieser Arbeit verzichtet.



(a) Protokollierung der Änderungen

(b) Versionenvergleich

Abbildung 7.7: Differenzen zwischen Tuning-Plan-Versionen

7.1.2.8 Bestimmung von Differenzen zwischen Tuning-Plan-Revisionen

Die Grundvoraussetzung für die Berücksichtigung von Vorwärtsdeltas bei der Speicherung von Tuning-Plan-Revisionen (vgl. *Abschnitt 7.1.2.5*) ist die Bestimmung der Differenzen zwischen zwei aufeinander folgenden Tuning-Plan-Versionen in Form von Sequenzen von Änderungsoperationen, die dazu verwendet werden können, um eine Revision in eine andere zu überführen.

Dazu gibt es zwei grundlegende Möglichkeiten. Zum einen kann der entsprechende Tuning-Plan-Editor dahingehend erweitert werden, dass bei jeder vorgenommenen Änderungsoperation eine *Protokollierung* dieser erfolgt. Nach Fertigstellung des Bearbeitungsprozesses kann somit die neue Revision mit den entsprechenden Änderungsoperationen abgespeichert werden (***Abbildung 7.7a***). Eine solche starke Integration des Modellierungswerkzeugs und des Versionierungssystems ist ein sehr vielversprechender Ansatz, der jedoch hier nicht weiterverfolgt werden konnte. Da zur Modellierung von Tuning-Plänen kein spezielles Modellierungswerkzeug neuentwickelt wurde, sondern auf die Verwendung und Erweiterung des *IBM WebSphere sMash (WsM)* Editors zurückgegriffen wurde (*Abschnitt 4.5.2*), ließ sich die Protokollierung von Änderungsoperationen nicht in den Editor integrieren.

Eine alternative Vorgehensweise ist die Ermittlung der Änderungsoperationen durch einen *Versionenvergleich*. Bei der Differenzbestimmung durch Versionenvergleich kann dabei sowohl auf den Inhalt der zu vergleichenden Dokumente als auch auf ihre Metadaten zurückgegriffen werden. Je mehr Metadaten über die Struktur der Dokumente vorliegen, umso leichter lassen sich die Differenz-Informationen ermitteln. Für verschiedene Dokumenttypen existieren zahlreiche Algorithmen. So finden sich beispielsweise Algorithmen zur Berechnung von Differenzen für Textdokumente bzw. für strukturierte Dokumente. Für die strukturierten Dokumente gibt es dabei eine Reihe von Algorithmen, die speziell auf die Dokumenttypen zugeschnitten sind und beispielsweise die Differenzbestimmung von beliebigen Bäumen oder Graphen bzw. speziell von XML-Dokumenten unterstützen [Ohs04].

Strukturierte Dokumente. Während bei den textuellen Dokumenten die Semantik durch die Reihenfolge der Textzeilen festgelegt wird, bestimmt sich diese bei den *struk-*

turierten Dokumenten durch ein zugrunde liegendes Modell. Die Algorithmen zur Differenzbestimmung arbeiten dabei idealerweise auf den Instanzen dieses Modells und nicht auf den eigentlichen strukturierten Dokumenten, welche lediglich externe Repräsentationen dieser Instanzen sind. Der Grund dafür sind mögliche syntaktische Veränderungen der externen Repräsentationen nach einem Laden-Speichern-Zyklus[31].

Die Differenzbestimmung wird hier auf eine Bestimmung von korrespondierenden Knoten mit anschließendem Vergleich zurückgeführt. Dabei muss unterschieden werden, ob die Knoten eindeutige Identifier besitzen, die über die gesamte Lebensdauer der Knoten unverändert bleiben oder nicht. Existieren solche Identifier, dann ist die Bestimmung von korrespondierenden Knoten trivial. Anderenfalls müssen spezielle Techniken zur Suche der korrespondierenden Knoten angewendet werden [WHW90]. Dazu gehören unter anderem

- Bestimmung von Schlüsselattributen[32] der Knoten, die einen Vergleich von Knoten ermöglichen.
- Verwendung von *Fuzzy Keys*, die eine Erkennung von ähnlichen Attributwerten ermöglichen, die nur zu einem bestimmten Grad übereinstimmen.
- Algorithmen aus der Graphentheorie beispielsweise zur Bestimmung des Teilgraph-Isomorphismus[33] [KN09].
- Verwendung einer *Look-Up Tabelle*, die die Äquivalenz von Knoten in den zu vergleichenden Dokumenten beschreibt.

Wurden die korrespondierenden Knoten ermittelt, können die Differenzen zwischen den Dokumenten bestimmt werden. Die Unterschiede in den Attributwerten lassen sich durch Vergleichen ihrer Werte feststellen. Bei textuellen Attributen kann man auf die Verfahren zur Differenzbestimmung in Textdokumenten zurückgreifen.

Bäume. Handelt es sich bei den strukturierten Dokumenten speziell um Bäume, so lassen sich diese zusätzliche Strukturinformation zur Steigerung der Effizienz der Differenzbestimmungsalgorithmen verwenden. Diese können dabei in zwei Kategorien eingeteilt werden: Algorithmen für *geordnete Bäume* berücksichtigen die Reihenfolge der Teilbäume, da diese einen Einfluss auf die Semantik der Dokumente hat. Die Algorithmen für *ungeordnete Bäume* dagegen ignorieren die Reihenfolge der Teilbäume und sind somit beispielsweise für XML-Dokumente geeignet, da die Reihenfolge der XML-Elemente implementierungsabhängig ist und somit variieren kann. Im Allgemeinen ist dabei die Bestimmung von Differenzen auf ungeordneten Bäumen NP-vollständig [ZSS92]. Durch Spezialisierung auf bestimmte Anwendungsbereiche, Datenformate oder ähnliches lassen sich jedoch Algorithmen sogar mit polynomialer Laufzeit entwickeln [WDC03].

[31] Eine solche Veränderung der WsM-basierten Tuning-Pläne findet nach einem Laden-Speichern-Zyklus ebenfalls statt. Eine dokumentenbasierte Versionierung ist daher in diesem Fall nicht sinnvoll.

[32] Als Schlüsselattribute von Knoten werden Attributmengen bezeichnet, die die Knoten eindeutig identifizieren.

[33] Im Allgemeinen ist das Problem der Bestimmung des Teilgraph-Isomorphismus NP-vollständig. Die Schwierigkeit liegt daher darin, eine effiziente Implementierung zur Bestimmung des Teilgraph-Isomorphismus, die auch die speziellen Eigenschaften der zu vergleichenden Dokumente berücksichtigen kann, zu finden.

Die Algorithmen für ungeordnete Bäume [Sel77, Tai79, ZS89, CRGMW96, CGM97] bzw. geordnete Bäume [BCD95] unterscheiden sich nicht nur bezüglich ihrer Arbeitsweise, sondern insbesondere auch bezüglich der auf den Bäumen definierten Operationen wie Hinzufügen, Löschen bzw. Ändern von Knoten, Hinzufügen, Löschen bzw. Verschieben von Teilbäumen etc. Bei der Entscheidung bezüglich der unterstützten Operationen sind auch die Operationskosten zu berücksichtigen. So kann beispielsweise die Teilbaum-Verschiebeoperation durch ein Löschen und ein anschließendes Einfügen des Teilbaums repräsentiert werden. Sind aber die Kosten einer Verschiebeoperation geringer als die aufsummierten Kosten der Lösch- und Einfüge-Operationen, so ist eine explizite Unterstützung dieser Operation sinnvoll.

XML-Dokumente. Wie oben erwähnt, lassen sich die Algorithmen zur Differenzbestimmung in Bäumen auch für XML-Dokumente verwenden. Diese Algorithmen haben keine besonderen Anforderungen an die Dokumente und berücksichtigen somit auch die besonderen Eigenschaften der XML-Dokumente nicht. Bindet man jedoch die für die XML-Dokumente spezifischen Informationen, wie beispielsweise die in XML definierten Knoten-Identifier, in die Algorithmen mit ein, so lässt sich ihre Performance steigern.

Ähnlich wie bei den Algorithmen zur Bestimmung von Differenzen in Bäumen, interpretieren einige Algorithmen die XML-Dokumente als ungeordnete [WDC03] bzw. geordnete [CAM02, Lin01] Bäume. Des Weiteren lassen sich auch hier die Algorithmen in Hinsicht auf die auf den Dokumenten definierten Operationen unterscheiden.

Der *XDiff-Algorithmus* [WDC03] operiert auf DOM-Bäumen [W3C04] und berechnet Signaturen für alle Knoten, die den Kontext der Knoten beschreiben. Diese errechnen sich aus den Knotennamen auf dem Pfad von der Wurzel des XML-Dokuments bis zum entsprechenden Knoten sowie dem Typ dieses Knotens. Zu jedem Knoten werden anschließend korrespondierende Knoten im zweiten, zu vergleichenden Dokument durch einen Signaturenvergleich gesucht. Danach werden für beide Bäume mögliche Sequenzen von Änderungsoperationen berechnet, welche Transformation eines Baums in einen anderen beschreiben, sowie ihre Kosten (*Transformationskosten*). Anschließend erfolgt die Auswahl einer entsprechenden Sequenz mit minimalen Kosten.

Der *XyDiff-Algorithmus* [CAM02] hat eine ähnliche Vorgehensweise. Der Unterschied besteht jedoch im Berücksichtigen der Reihenfolge von Knoten sowie in der Tatsache, dass der Algorithmus eine nicht optimale Sequenz von Änderungsoperationen bestimmt. Sein Ziel, möglichst kurze Laufzeiten zu erzielen, erreicht der Algorithmus jedoch durch Zurückgreifen auf Heuristiken beim Teilbaumvergleich sehr gut.

Basierend auf diesen Erkenntnissen wurde eine entsprechende Komponente für einen Vergleich von Tuning-Plan-Revisionen entwickelt, welche die ursprüngliche Tuning-Plan-Revision v_i sowie ihre Nachfolgerrevision v_{i+1} übergeben bekommt und eine *mögliche* Sequenz von Änderungsoperationen ermittelt, die eine Überführung von v_i in v_{i+1} durch ein Nachfahren dieser Operationen ermöglicht (***Abbildung 7.7b***). Zwar kann dabei nicht davon ausgegangen werden, dass die ermittelte Sequenz von Änderungsoperationen exakt der tatsächlich vorgenommenen Sequenz von Änderungsoperationen entspricht, die Anwendung der ermittelten Änderungsoperationen liefert jedoch die Revision v_{i+1}. Im Gegensatz zum

operationsbasierten Ansatz (Protokollierung der Änderungsoperationen) handelt es sich bei diesem Verfahren um einen *zustandsbasierten Ansatz*.

Zur Umsetzung dieser Komponente konnte beispielsweise auf den für ungeordnete XML-Dokumente entwickelten XDiff-Algorithmus (*Abschnitt 7.1.2.8*) zurückgegriffen werden. Bei den mit einem WsM-basierten Tuning-Plan-Editor entwickelten Tuning-Plänen handelt es sich jedoch um keine gewöhnlichen XML-Dokumente. Das Speicherungsformat von WsM ist sehr spezifisch, so dass durch einen speziell auf dieses Format zugeschnittenen Algorithmus eine bessere Qualität und Performance erzielt werden konnten. Da die Vorgehensweise dieses Algorithmus sich nur geringfügig von der Vorgehensweise des XDiff-Algorithmus unterscheidet, wurde im Rahmen dieser Arbeit auf die Vorstellung dieses Algorithmus verzichtet.

Es bleibt zu erwähnen, dass die Differenzen zwischen zwei Revisionen eines Tuning-Plans nicht nur die Grundlage für ihre effiziente Speicherung (*Abschnitt 7.1.2.2*) bilden, sondern auch für die Umsetzung des Benachrichtigungsdienstes (*Abschnitt 7.1.3*) bzw. für die Präsentation von Änderungen (*Abschnitt 7.1.4*).

7.1.3 Benachrichtigungsdienst

Bei erfolgten Modifikationen von Tuning-Plänen (Anlegen einer neuer Version, Zurückkehren auf eine alte Version etc.) kann die Community über die vorgenommenen Änderungen benachrichtigt werden.

Dabei kann zwischen einer Message-basierten und einer Flag-basierten Benachrichtigung unterschieden werden [CK86]. Bei einer *Message-basierten Benachrichtigung* wird eine Nachricht verschickt, die die Nutzer über die erfolgte Änderung benachrichtigt. Dies kann entweder nach dem *Broadcast-Prinzip* oder nach dem *Publish-Subscribe-Prinzip* geschehen. Während bei einer Benachrichtigung nach dem *Broadcast-Prinzip* alle Community-Mitglieder über die erfolgte Änderung informiert werden, beschränkt sich die Benachrichtigung nach dem *Publish-Subscribe-Prinzip* ausschließlich auf die an dem veränderten Tuning-Plan interessierten Community-Mitglieder, die ihr „Interesse“ am Tuning-Plan vorab durch ein Abonnement (*Subscription*) beurkundet haben. Bei einer Message-basierten Benachrichtigung ist weiterhin zwischen einer *sofortigen*, unmittelbar nach der Durchführung von Änderungen erfolgenden, und einer *verzögerten*, erst nach Ablauf einer bestimmten Zeit nach Durchführung von Änderungen erfolgenden Benachrichtigung zu unterscheiden.

Bei einer *Flag-basierten Benachrichtigung* erfolgt dagegen keine unmittelbare Benachrichtigung der Community-Mitglieder. Es werden lediglich die geänderten Objekte entsprechend markiert. Nur interessierte Nutzer, die auf ein solches Objekt zugreifen, würden anhand dieser Markierung feststellen, dass das Objekt geändert wurde. Somit ist die Flag-basierte Benachrichtigung immer eine verzögerte Benachrichtigung.

Benachrichtigung von Autoren. Zur Benachrichtigung von (potenziellen) Autoren eignet sich am besten eine Kombination aus den oben genannten Ansätzen. Bei einer Weiterentwicklung des Tuning-Plans, von welchem andere Varianten abgeleitet wurden, macht es Sinn die Entwickler der abgeleiteten Varianten über die erfolgten Änderungen zu informieren, da die vorgenommenen Änderungen u.U. auch für die Varianten

in Frage kommen. Dabei sind die Autoren aller Revisionen jeder der abgeleiteten Varianten zu informieren. Dennoch sollen auch die potenziellen Autoren von weiteren Revisionen dieser Varianten über die erfolgten Änderungen benachrichtigt werden. Daher sind die Änderungsinformationen an die Tuning-Plan-Container der jeweiligen Varianten zu knüpfen und den interessierten Community-Mitgliedern zu präsentieren.

Benachrichtigung von Nutzern. Bei der Identifikation von Community-Mitgliedern, die über Versionsänderungen zu benachrichtigen sind, ist sinnvollerweise zwischen lediglich interessierten Mitgliedern und Mitgliedern, die eine Kopie des gerade veränderten Tuning-Plans auf ihrem System vorhalten, zu unterscheiden. Eine Benachrichtigung dieser „betroffenen“ Nutzer hat eine höhere Priorität. Sind es kritische Änderungen, die am Tuning-Plan vorgenommen wurden, dann sind die lokalen Kopien durch die aktualisierte Version des Tuning-Plans zu ersetzen (implizite Updates). Im Fall von weniger kritischen Änderungen können die Benutzer entscheiden, ob sie die lokale Kopie durch die aktualisierte ersetzen wollen oder nicht (explizite Updates). Des Weiteren ist ein Informieren der Autoren der vorhergehenden Versionen des Tuning-Plans sinnvoll, um ihnen gegebenenfalls ein Reagieren auf die Änderungen zu ermöglichen.

7.1.4 Präsentation von Änderungen

Die Darstellung der Änderungen bekommt insbesondere in kollaborativen Arbeitsumgebungen eine immer größere Bedeutung [Haa96]. Den Nutzern fällt es oft relativ schwer, den Überblick über die Evolutionshistorie zu behalten. In einer Studie zur kooperativen Dokumentenentwicklung werden die Wünsche der Nutzer in Bezug auf eine kollaborative Entwicklungsumgebung zusammengefasst [BLNP97, Len00]:

- Eine Evolutionshistorie des gesamten Dokuments sollte zur Verfügung stehen.
- Einzelne Entwicklungsschritte sollten nachvollziehbar sein.
- Eine Betrachtung ausgewählter Evolutionsintervalle sollte möglich sein.
- Falls erfolgte Änderungsoperationen protokolliert wurden, so sollen diese leicht verständlich und bei Bedarf sichtbar für den Benutzer sein.

Daraus wird die bestehende Nachfrage nach einer flexiblen Evolutionsbeschreibung und -darstellung erkenntlich. Bei der Präsentation der erfolgten Änderungen sind die Benutzer insbesondere vom eigentlichen Vergleich der Dokumente zu entlasten, so dass sie sich auf die Semantik der Änderungen konzentrieren können [NCK+92].

In das zu entwickelnde Community-System ist daher eine Komponente zur Visualisierung von Unterschieden zwischen den Tuning-Plan-Versionen zu integrieren. Bei der Visualisierung der Änderungen ist zwischen Strukturänderungen (z. B. Verschiebung von Tuning-Schritten aus einem schachtelbaren Tuning-Schritt in einen anderen) und expliziten Inhaltsveränderungen (z. B. Löschung bzw. Einfügung von Tuning-Schritten) zu unterscheiden.

7.1.5 Mehrbenutzerbetrieb

7.1.5.1 Nebenläufigkeitskontrolle

Wird auf Daten durch mehrere Prozesse zugegriffen, so können diese Zugriffe typischerweise parallel (oder nebenläufig) ablaufen. Bei solchen nebenläufigen Zugriffen auf gemeinsame Ressourcen kann es jedoch unter Umständen zu Konflikten kommen. Beispiele für solche Konflikte sind *Lost Updates*, *Uncommited Reads*, *Non-repeatable Reads* und *Phantom Reads* [Vos08]. Um solche Probleme zu vermeiden, müssen die konkurrierenden Zugriffe koordiniert werden. Diese Koordination wird mit dem Begriff *Nebenläufigkeitskontrolle* bezeichnet. Grundsätzlich lässt sich zwischen zwei Vorgehensweisen der Nebenläufigkeitskontrolle unterscheiden:

Konsistenzerhaltung durch Konfliktvorbeugung. Die Verfahren dieser Kategorie werden in der Literatur oft als *pessimistische* oder *konservative Verfahren* bezeichnet. Dazu zählen beispielsweise die in der Datenbankwelt weit verbreiteten Sperrverfahren. Dabei werden die Objekte vor ihrer Bearbeitung gesperrt. Andere Nutzer sind dadurch nicht in der Lage, die gesperrten Objekte zu bearbeiten und müssen auf die Sperrfreigabe warten. Durch solche Verfahren kann zwar die Konsistenz gesichert werden, die Nebenläufigkeit nimmt jedoch stark ab. Sie lässt sich zum einen durch eine Unterscheidung zwischen Lese- und Schreibsperren wieder steigern. Zum anderen schafft eine Verkleinerung der Objektgranularität, beispielsweise durch eine Einführung von hierarchischen Sperren, Abhilfe [SHS05].

Konsistenzerhaltung durch Konfliktauflösung. Verfahren, die in diese Kategorie reinfallen, sind auch als *optimistische Verfahren* bekannt. Sie basieren auf der Beobachtung, dass die Wahrscheinlichkeit für den konfligierenden Zugriff zweier Prozesse auf ein Objekt relativ gering ist. Die Benutzer können daher zu jedem Zeitpunkt Änderungen an den verwalteten Objekten vornehmen. Naheliegenderweise lassen sich die Konflikte jedoch nicht ganz ausschließen. Sie entstehen insbesondere bei gleichzeitig an gleichen Objekten durchgeführten Änderungen. Bei der Speicherung eines geänderten Objekts kann daher geprüft werden, ob das Objekt zwischenzeitlich geändert wurde oder nicht. Wenn das der Fall ist, so kann versucht werden, den Konflikt automatisch aufzulösen [SS07]. Sollte eine automatische Konfliktauflösung nicht möglich sein, so kann der Nutzer aufgefordert werden, den Konflikt manuell aufzulösen und die geänderten Objekte zusammenzuführen. Durch eine Verkleinerung der Objektgranularität und dadurch eine „feinere Buchführung“ kann auch hier die Anzahl von Konflikten reduziert werden.

7.1.5.2 Checkout/Checkin

Bei der Problematik des Mehrbenutzerbetriebs in Repository-Systemen muss berücksichtigt werden, dass die Versionsobjekte üblicherweise über längere Zeiträume bearbeitet werden müssen. Eine Einführung von sogenannten langen Transaktionen [LL95, HR01] ist daher an dieser Stelle sinnvoll. Mit Hilfe einer *Checkout*-Operation wird ein Element aus dem Repository ausgecheckt (Transaktionsbeginn). Dabei wird auf dem Arbeitssystem eine Kopie des ausgecheckten Objekts angelegt. Nach einer unter Umständen langen Bearbeitungsphase auf dem Arbeitssystem muss das Objekt mittels einer *Checkin*-Operation in

das Repository wieder eingecheckt werden (Transaktionsende). Dabei wird eine Kopie des eingecheckten Objekts aus dem Arbeitssystem in das Repository übertragen und dauerhaft gespeichert. Darauf kann anschließend jederzeit wieder lesend bzw. schreibend zugegriffen werden. Zur Einschränkung des Zugriffs kann dabei vom Rechtekonzept Gebrauch gemacht werden.

In Abhängigkeit vom verwendeten Mechanismus zur Umsetzung des Mehrbenutzerbetriebs verändert sich die Funktionsweise der Checkout-/Checkin-Operationen.

Pessimistisches Locking. In der Welt der Repository-Systeme ist dieser Mechanismus auch unter dem Namen *Sperren–Ändern–Freigeben* (engl.: *Lock–Modify–Unlock*) bekannt [CSFP09]. Die Checkout-Operation erzeugt hier für jedes ausgecheckte Element eine Sperre. Andere Nutzer müssen somit warten, bis der Besitzer der Sperre diese wieder freigibt. Durch Ausführung der Checkin-Operation erfolgt zunächst eine Aktualisierung des modifizierten Objekts und anschließend eine Freigabe der Sperre.

Optimistisches Locking. Dieser Mechanismus findet sich oft unter dem Namen *Kopieren–Ändern–Zusammenführen* (engl.: *Copy–Modify–Merge*). Hier erzeugt die Checkout-Operation lediglich eine Kopie des ausgecheckten Elements auf dem Arbeitssystem [CSFP09]. Die Quellversion des Objekts wird nicht gesperrt und kann somit auch von anderen Nutzern ausgecheckt werden. Mehrere Nutzer können nun gleichzeitig an einem Element in ihren Arbeitskopien arbeiten. Bei der Ausführung der Checkin-Operation muss überprüft werden, ob sich die im Repository abgelegte Quellversion seit dem Auschecken verändert hat, das heißt jemand seine Änderungen früher eingecheckt hat. Ist das der Fall, so müssen die beiden Versionen miteinander verglichen werden. Wurden die Änderungen an unterschiedlichen Stellen durchgeführt, können beide Versionen automatisch zusammengeführt und gespeichert werden. Änderungen an den gleichen Stellen können in der Regel nicht automatisch aufgelöst werden. Zwar könnten solche Konflikte sich beispielsweise mittels Prioritäten, die den einzelnen Versionen zugewiesen sind, auflösen lassen, dies ist jedoch oft nicht ausreichend. In solchen Fällen sind die Konflikte durch den Nutzer vor dem Einchecken manuell aufzulösen.

7.1.5.3 Architekturen kollaborativer Systeme

Nach [Ger07] lässt sich bei den auf einen Mehrbenutzerbetrieb ausgelegten Architekturen zwischen zentralisierten, verteilten und hybriden Architekturen unterscheiden:

Zentralisierte Architektur. Die zentralisierte Architektur ist bei verteilten Mehrbenutzersystemen am meisten verbreitet. Bei dieser Architektur verwaltet ein zentraler Server die im Zugriff mehrerer Clients befindlichen Daten. Nicht nur die Datenspeicherung, sondern auch der Zugriff auf die Daten wird dabei vom Server geregelt. Je nach Umsetzung laufen auf dem Server ein oder mehrere Prozesse. Der Einsatz eines einzigen Server-Prozesses schließt dabei Dateninkonsistenzen aus und minimiert die Nebenläufigkeit. Da der Server-Prozess nur einen Request pro Zeiteinheit bedienen kann, ist ein gleichzeitiger Zugriff mehrerer Clients auf eine Ressource ausgeschlossen. Zur Steigerung der Nebenläufigkeit lassen sich auch mehrere Prozesse auf dem Server verwenden. Die potenziellen Konflikte können dabei mit den in *Abschnitt 7.1.5.1* vorgestellte Verfahren vermieden bzw. aufgelöst werden.

Verteilte Architektur. Die Verwendung eines zentralen Servers, der die einzige Kopie der Daten bereithält, auf die von vielen Clients zugegriffen wird, schränkt die Nebenläufigkeit sehr stark ein[34]. Diese kann durch den Einsatz von Replikationsmechanismen deutlich gesteigert werden. Bei einer verteilten (oder replizierten) Architektur existiert keine zentrale Datenquelle. Stattdessen werden die Daten unter allen Clients verteilt. Dabei verfügt jeder einzelne Client entweder über eine Kopie aller Daten oder nur eines für diesen Client relevanten Datenausschnitts. Die Clients können nun ungestört auf dedizierten Objektkopien arbeiten, ohne dabei einander zu behindern. Es bleibt dabei jedoch das Problem der Konsistenzsicherung zwischen den sich unter Umständen überlappenden Replikaten. Dazu läuft auf jedem Client ein Prozess, der ähnlich wie der Server-Prozess für den Datenzugriff und die Synchronisation der Daten zuständig ist. Dazu ist eine Kommunikation der Client-Prozesse untereinander notwendig.

Der Vorteil einer replizierten Architektur ist neben einer aufgrund von mehreren Datenkopien geringerer Ausfallwahrscheinlichkeit die hohe Performance und eine schnelle lokale Sichtbarkeit der durchgeführten Änderungen. Erst nach den erfolgten lokalen Datenänderungen sorgt der Client-Prozess für eine Propagierung der Änderungen auf die entfernten Clients. Sobald die Clients die Datenänderungen (entweder in Form von neuen Datenobjekten oder in Form von Änderungsoperationen) erhalten, pflegen sie diese in die lokale Datenbank ein. Bei den Änderungsoperationen kann zwischen problematischen und unproblematischen Änderungen unterschieden werden. Während die unproblematischen Änderungen, wie beispielsweise Einfügen von neuen Datenobjekten, nicht zu Dateninkonsistenzen führen, kann es beim Propagieren der problematischen Änderungen zu Konflikten kommen, falls mehrere Clients die gleichen Daten bearbeitet haben. Zur Minimierung, Vermeidung oder Auflösung von Konflikten muss auch hier auf spezielle Verfahren der Nebenläufigkeitskontrolle zurückgegriffen werden.

Hybride Architektur. Wie es der Name vermuten lässt, vereinigt eine hybride Architektur die Vorteile einer zentralen und einer verteilten Architektur. Hier gibt es einen zentralen Server, der immer eine aktuelle Version jedes Datenobjekts bereithält. Auf die Clients werden Kopien der Daten übertragen. Dabei können auf den Clients entweder alle Daten oder ausschließlich die für die Clients relevanten Datenausschnitte gespeichert werden. Sowohl auf dem Server als auch auf den einzelnen Clients laufen spezielle Prozesse, die für den Datenzugriff und die -synchronisation verantwortlich sind. Die unproblematischen Änderungen können lokal durchgeführt werden und anschließend mit Hilfe von diesen Prozessen an die anderen Clients und den zentralen Server übertragen werden. Die problematischen Änderungen werden hingegen vor lokaler Durchführung zunächst auf den zentralen Server übertragen, dort geprüft und erst dann lokal durchgeführt. Zur Nebenläufigkeitskontrolle können auf dem Server beispielsweise die klassischen Sperrverfahren eingesetzt werden. Alternativ können die problematischen Änderungen sofort auf dem lokalen System durchgeführt werden und erst dann an den Server bzw. die anderen Clients übertragen werden. Dabei müssen ähnlich wie bei der verteilten Architektur entsprechende Konfliktauflösungsmechanismen eingesetzt werden.

[34] Sicherlich sind auch die Performance bzw. die Verfügbarkeit durch den Einsatz eines einzigen zentralen Servers stark beeinträchtigt.

Bei der Konzeption eines Community-Systems zum Austausch von Tuning-Plänen muss berücksichtigt werden, dass der Ansatz von den typischen Mehrbenutzer-Anwendungsszenarien grundsätzlich abweicht. Eine kollaborative, parallele Entwicklung eines einzelnen Tuning-Plans ist wenig sinnvoll, da die DBA typischerweise Tuning-Pläne modellieren, die auf ihren eigenen Erfahrungen basieren. Es findet zunächst also keine Teamarbeit bei der Entwicklung von Tuning-Plänen statt. Die erstellten und auf den Community-Server hochgeladenen Tuning-Pläne können aber anschließend von anderen Community-Mitgliedern weiterentwickelt werden. Aus diesem Grund ist der Einsatz einer hybriden Architektur sinnvoll. Die auf den lokalen Systemen der DBA entwickelten und getesteten Tuning-Pläne werden zum Zweck des Community-Austauschs auf den zentralen Community-Server hochgeladen und mit entsprechenden Metadaten versehen (vgl. *Abschnitt 7.1.6.1*). Anhand dieser Metadaten können die Tuning-Pläne auf dem Server von anderen Nutzern gefunden und auf ihre lokalen Systeme heruntergeladen werden.

7.1.5.4 Evolution von Tuning-Plänen im Mehrbenutzerbetrieb

Speziell im Hinblick auf die Erstellung und Weiterentwicklung von Tuning-Plänen muss untersucht werden, inwieweit die Tuning-Plan-Versionierung den Mehrbenutzerbetrieb beeinträchtigt. Dazu werden im Folgenden zunächst die für die Evolution relevanten Anwendungsszenarien beschrieben. Anschließend wird untersucht, welche der einzelnen Anwendungsszenarien miteinander „verträglich“ sind und somit parallel ausgeführt werden können.

Anwendungsszenario 1: Herunterladen eines Tuning-Plans

1. Suche eines interessierenden Tuning-Plans auf dem Community-Server.
2. Checkout der letzten Revision des ausgewählten Tuning-Plans. Bei dem Checkout erfolgt die Übertragung einer Kopie dieser Revision auf das lokale Arbeitssystem.
3. Lokale Verwendung des Tuning-Plans.

Anwendungsszenario 2: Erstellung eines neuen Tuning-Plans

1. Lokale Erstellung eines Tuning-Plans.
2. Checkin des erstellten Tuning-Plans. Dabei wird zunächst ein neuer Tuning-Plan-Container auf dem Community-Server angelegt. Der eingecheckte Tuning-Plan wird anschließend zur ersten Revision des angelegten Tuning-Plan-Containers.
3. Angabe von Metadaten des Tuning-Plans. Dazu gehört auch eine Einordnung des Tuning-Plans in eine bestimmte Kategorie. Bei einer starken Ähnlichkeit zwischen dem neu erstellten und einem existierenden Tuning-Plan kann (nachträglich) eine Varianten-Beziehung zwischen den beiden Tuning-Plänen hergestellt werden.

Anwendungsszenario 3: Erstellung einer neuen Revision

1. Suche eines entsprechenden Tuning-Plans, der weiterentwickelt werden soll, auf dem Community-Server.

Nutzer 2 Nutzer 1	TP herunterladen	neuen TP erstellen	Revision erstellen	Variante erstellen
TP herunterladen	+	+	+	+
neuen TP erstellen	+	+	+	+
Revision erstellen	+	+	$-_p$ / $(+_o)$	+
Variante erstellen	+	+	+	+

Tabelle 7.2: Evolution von Tuning-Plänen im Mehrbenutzerbetrieb

2. Soll eine ältere Revision des Tuning-Plans weiterentwickelt werden, dann muss zunächst ein Rollback (mittels einer Rollback-Referenz) auf diese erfolgen (vgl. *Abschnitt 7.1.2.5*). Damit wird die zu modifizierende Revision zur letzten Revision und kann weiterentwickelt werden.

3. Checkout der letzten Tuning-Plan-Revision. Dabei erfolgt eine Übertragung einer Kopie dieser Revision auf das lokale Arbeitssystem.

4. Lokale Bearbeitung der Revision.

5. Checkin der neuen Revision. Dabei wird die neue Revision als Nachfolger der zuvor ausgecheckten Revision auf dem Community-Server angelegt.

Anwendungsszenario 4: Erstellung einer neuen Variante

1. Suche eines interessierenden Tuning-Plans und seiner entsprechenden Revision, die die Grundlage für eine neue Tuning-Plan-Variante bilden soll, auf dem Community-Server.

2. Checkout dieser (gegebenenfalls älteren) Tuning-Plan-Revision. Dabei erfolgt eine Übertragung einer Kopie dieser Revision auf das lokale Arbeitssystem.

3. Lokale Bearbeitung der Revision.

4. Checkin der neuen Revision als eine neue Variante des Tuning-Plans. Dabei wird zunächst ein neuer Tuning-Plan-Container als Variante des zuvor ausgecheckten Tuning-Plans angelegt (vgl. *Abschnitt 7.1.2.3*). Anschließend wird die eingecheckte Version des Tuning-Plans zur ersten Revision der neuen Variante.

Die Parallelisierbarkeit der oben genannten Anwendungsszenarien ist in ***Tabelle 7.2*** dargestellt. Aufgrund der Tatsache, dass eine eingecheckte Revision nicht mehr modifiziert werden kann und bei einer Änderung eine neue Nachfolger-Revision (oder eine neue Variante) erzeugt wird, ist die Nebenläufigkeit dabei erfreulich hoch. Einige Erläuterungen zur Nebenläufigkeit der Anwendungsszenarien finden sich nachfolgend:

Tuning-Plan herunterladen. Während die letzte Revision eines Tuning-Plans heruntergeladen wird, kann es zu keinen Konflikten kommen. Wird ein neuer Tuning-Plan bzw. eine neue Tuning-Plan-Variante erstellt, so hat es keinerlei Auswirkungen auf den heruntergeladenen Tuning-Plan. Eine neu erstellte Revision des heruntergeladenen Tuning-Plans ist unter Umständen auf das lokale Arbeitssystem zu übertragen, um die lokal verfügbare Version des Tuning-Plans zu aktualisieren (*Abschnitt 7.1.3*), was jedoch die Nebenläufigkeit nicht einschränkt.

Neuen Tuning-Plan erstellen. Ein auf dem lokalen Arbeitssystem neu erstellter Tuning-Plan kann jederzeit auf den Community-Server übertragen werden. Bis zum Zeitpunkt seiner Speicherung ist dieser für andere Nutzer nicht sichtbar, so dass es auch hier zu keinen Konflikten kommen kann.

Variante erstellen. Die Erstellung einer neuen Tuning-Plan-Variante entspricht im gewissen Umfang der Erstellung eines neuen Tuning-Plans mit dem Unterschied, dass hier ein anderer Tuning-Plan als Grundlage verwendet wird. Genauso wie bei der Erstellung eines neuen Tuning-Plans kommt es auch hier nicht zu Konflikten.

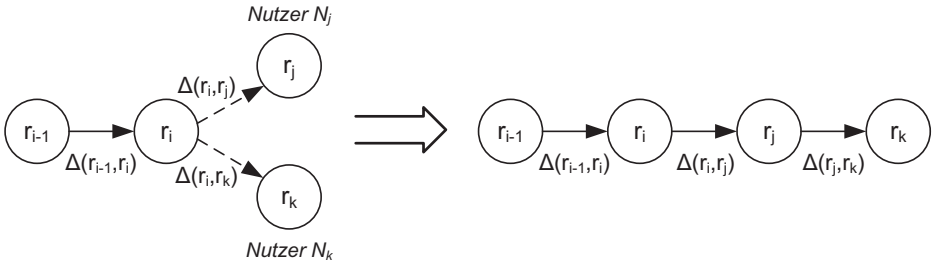
Revision erstellen. Bei der Erstellung von neuen Revisionen hat der Spezialfall einer gleichzeitigen Erstellung mehrerer Revisionen durch verschiedene Nutzer Konfliktpotential. In Abhängigkeit davon, ob pessimistische oder optimistische Nebenläufigkeitskontrolle gewählt wird, lassen sich Konflikte entweder vermeiden oder auflösen.

Pessimistische Nebenläufigkeitskontrolle. Beim Einsatz einer pessimistischen Nebenläufigkeitskontrolle ist ein gleichzeitiges Anlegen mehrerer Revisionen nicht möglich (gekennzeichnet durch $-_p$ in *Tabelle 7.2*). Der erste Nutzer setzt beim Checkout der zu ändernden Revision eine Sperre und gibt diese erst nach der Fertigstellung seiner Änderungen frei. Andere Nutzer müssen auf die Sperrfreigabe warten, bevor sie sich an einer Weiterentwicklung des Tuning-Plans beteiligen können. Der Ablauf des Sperrensetzens und -freigebens sieht wie folgt aus:

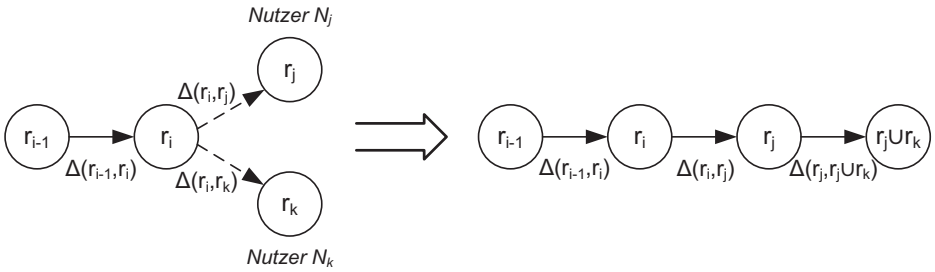
- Beim Checkout der Tuning-Plan-Revision r_i durch den Nutzer N_j Setzen einer Sperre auf r_i
- Der Nutzer N_k kann diese Revision r_i somit nicht auschecken, bis die Sperre freigegeben wurde
- Beim Checkin der neuen Revision r_j als Nachfolger-Revision von r_i durch den Nutzer N_j wird die Sperre von r_i freigegeben
- Der Nutzer N_k kann nun die letzte Tuning-Plan-Revision r_j zwecks Weiterentwicklung auschecken

In *Abschnitt 7.1.2.5* wurde bereits die Einschränkung, dass der Nutzer N_k ausschließlich auf die letzte Tuning-Plan-Revision zwecks Weiterentwicklung zugreifen kann, angesprochen. Die Weiterentwicklung einer älteren Revision kann mittels eines vorhergehenden Rollbacks auf diese Revision umgesetzt werden. Bei Bedarf kann alternativ dazu eine ältere Revision die Grundlage für eine neue Variante bilden.

Der Einsatz einer pessimistischen Nebenläufigkeitskontrolle impliziert eine stark eingeschränkte Nebenläufigkeit (*Abschnitt 7.1.5.1*). Zur Steigerung der Nebenläufigkeit kann eine Timeout-Schranke für eine automatische Sperrfreigabe definiert werden. Sollte die Bearbeitung des Tuning-Plans bis zum Ablauf des zulässigen Zeitfensters



(a) Logisches Überschreiben der Änderungen von N_j durch Änderungen von N_k



(b) Zusammenführung von Änderungen von N_j und N_k

Abbildung 7.8: Revisionsbildung bei optimistischer Nebenläufigkeitskontrolle

nicht abgeschlossen sein, so muss die Sperre erneut angefordert werden. Dies kann entweder durch eine spezielle Sperranforderungsoperation oder – mit im Wesentlichen gleichen Effekt – ein erneutes Checkout umgesetzt werden. Bei einem erneuten Checkout muss jedoch sichergestellt werden, dass die bereits lokal erfolgten Änderungen der Tuning-Plan-Revision nicht verworfen werden.

Optimistische Nebenläufigkeitskontrolle. Beim Einsatz der optimistischen Nebenläufigkeitskontrolle ist eine gleichzeitige Erstellung neuer Revisionen durch verschiedene Nutzer bedingt möglich (gekennzeichnet durch $(+_o)$ in *Tabelle 7.2*). Hier ist ein gleichzeitiger Checkout einer Revision r_i durch mehrere Nutzer (beispielsweise N_j und N_k) zunächst möglich. Erstellen nun die beiden Nutzer zwei neue Revisionen r_j und r_k auf Basis von der ausgecheckten Revision r_i, so kommt es beim abschließenden Checkin beider Revisionen zu einem Konflikt. Die Änderungen der zuerst eingecheckten Revision r_j werden durch die danach eingecheckte Revision r_k logisch überschrieben, da die zuletzt eingecheckte Revision r_k als der Nachfolger der Revision r_j abgespeichert wird. Desweiteren kommt es hier unter Umständen zu Inkonsistenzen in den Delta-Logs[35], da die Änderungsoperationen beider Revisionen r_j und r_k im Normalfall ausgehend von der Basisversion r_i berechnet bzw. protokolliert werden.

Den beiden angesprochenen Problemen kann beispielsweise durch folgende Ansätze begegnet werden (***Abbildung 7.8***):

Ansatz 1: Logisches Überschreiben der Änderungen durch die zuletzt eingecheckte Revision mit automatischer Neubestimmung der Delta-Infor-

[35]Delta-Logs beinhalten die protokollierten Änderungsoperationen.

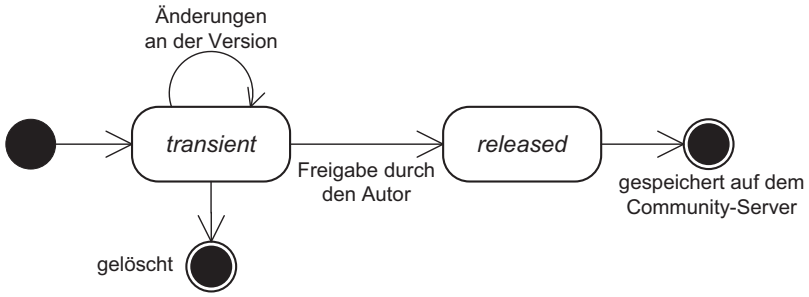


Abbildung 7.9: Zustände einer Tuning-Plan-Version

mationen. Eine graphische Veranschaulichung dieses Lösungsansatzes findet sich in *Abbildung 7.8a*. Der Ablauf ist dabei wie folgt:

- Die zuerst eingecheckte Revision r_j wird zum Nachfolger von r_i.
- Die anschließend eingecheckte Revision r_k wird zum Nachfolger von r_j und das Delta für r_k wird (durch Versionenvergleich) automatisch ausgehend von r_j berechnet.

Ansatz 2: Zusammenführung von Änderungen mit automatischer Neubestimmung der Delta-Informationen. Dieser Lösungsansatz ist in *Abbildung 7.8b* dargestellt und wird nachfolgend kurz erklärt:

- Die zuerst eingecheckte Revision r_j wird zum Nachfolger von r_i.
- Die anschließend eingecheckte Revision r_k wird automatisch oder mit Feedback des Autors von r_k (Nutzer N_k) mit der Revision r_j zusammengeführt und als Nachfolger von r_j gespeichert. Auch hier wird das Delta automatisch berechnet.

Sollte eine automatische Bestimmung der Änderungsoperationen nicht möglich sein (vgl. *Abschnitt 7.1.2.8*), so ist es ratsam, von der pessimistischen Nebenläufigkeitskontrolle Gebrauch zu machen, um die Inkonsistenzen in den Delta-Logs zu vermeiden.

7.1.5.5 Versionszustände

Bei den bisherigen Betrachtungen wurde angenommen, dass Versionen auf zwei Wegen erstellt werden können: Zum einen können (Wurzel-)Versionen explizit erstellt werden. Zum anderen können Versionen implizit, durch Ableiten von existierenden Versionen, erstellt werden. Die Versionen entstehen somit zu bestimmten Zeitpunkten. Viel realistischer ist es jedoch anzunehmen, dass insbesondere bei komplexen Objekten die Versionen *inkrementell* erstellt werden. Um die Entwicklungsstufen der zu versionierenden Objekte besser abbilden zu können, wurde das Konzept der *Versionszustände* eingeführt. Zu jedem Zeitpunkt befindet sich dabei eine Version in einem definierten Zustand. Oft unterscheidet man dabei zwischen drei Versionszuständen: *transient*, *working*, *released* [CK86]. Diese Zustände legen dabei fest, wie auf die Versionen zugegriffen werden kann: Während die *transienten* Versionen sich noch in Entwicklung befinden und vom Autor modifiziert oder gar gelöscht

	Austausch von *freigegebenen* Versionen		**Austausch von *transienten* bzw. *freigegebenen* Versionen**	
	transient	**released**	**transient**	**released**
Sichtbarkeit für andere Nutzer	−	+	+[a]	+
Direkte Änderungen	+	−	+	−
Änderungen durch Bildung neuer Revisionen	−	+	+	+
Bildung von abgeleiteten Varianten	−	+	−	+
Löschbarkeit der Version	+	−	+	−

[a] Die *transienten* Versionen sind nur für bestimmte Nutzergruppen sichtbar.

Tabelle 7.3: Kollaborative Entwicklung und Zustände von Tuning-Plan-Versionen

werden können, werden die *stabilen (stable)* Versionen als lauffähig angesehen und können zwar vom Autor gelöscht, aber nicht mehr verändert werden. Die bereits *veröffentlichten* Versionen (*released*) können dagegen nicht verändert und nicht gelöscht werden.

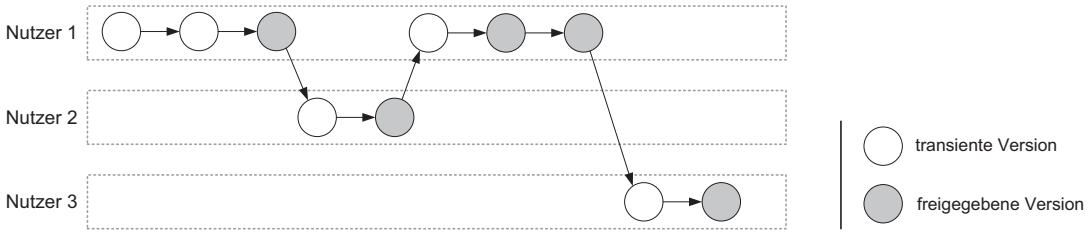
Für die Evolution von Tuning-Plänen im Mehrbenutzerbetrieb erscheint eine Unterscheidung zwischen den Tuning-Plan-Zuständen *transient* und *released* sinnvoll. Das entsprechende Zustandsdiagramm findet sich in ***Abbildung 7.9***.

Transient. Versionen, die sich noch in Entwicklung befinden, werden als *transient* gekennzeichnet. Wird eine neue Version erstellt, so befindet sie sich im Zustand *transient*. Auch beim Ableiten von neuen Revisionen bzw. Varianten von einer existierenden Version werden die neu abgeleiteten Versionen im Zustand *transient* angelegt. Diese können vom Benutzer (Autor) bearbeitet, lokal zwischengespeichert oder gelöscht werden.

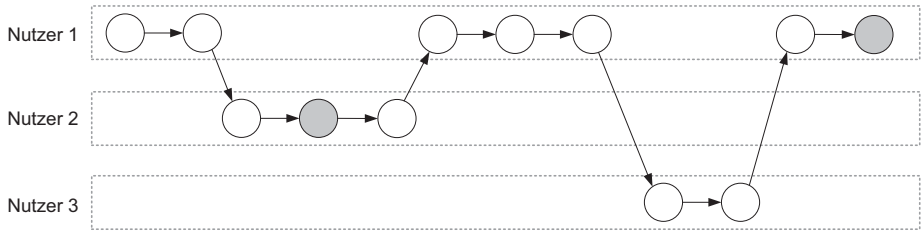
Released. Ist die Bearbeitung und Testen einer *transienten* Version abgeschlossen, so kann diese durch den Nutzer explizit als *released* (*freigegeben*) gekennzeichnet werden. Eine Version, die sich in diesem Zustand befindet, wird „eingefroren“ und kann nicht mehr gelöscht bzw. verändert werden. Auch eine Umwandlung einer Version aus dem Zustand *released* in den Zustand *transient* zur weiteren Bearbeitung ist nicht möglich. Stattdessen kann jedoch von einer solchen eingefrorenen Version eine neue Revision, die beliebig verändert werden kann, angelegt werden.

Speziell im Hinblick auf eine kollaborative Entwicklung von Tuning-Plänen ist weiterhin zwischen zwei Vorgehensweisen zu unterscheiden (***Tabelle 7.3***):

Kollaborative Entwicklung an *freigegebenen* Versionen. Bei dieser Vorgehensweise können ausschließlich die Tuning-Plan-Versionen im Zustand *released* auf den Community-Server hochgeladen werden. Dort stehen sie anderen Community-Mitgliedern



(a) Kollaborative Entwicklung an *freigegebenen* Versionen



(b) Kollaborative Entwicklung an *transienten* bzw. *freigegebenen* Versionen

Abbildung 7.10: Kollaborative Entwicklung von Versionen

zur Verwendung bzw. Weiterentwicklung zur Verfügung. Die transienten Versionen bleiben dagegen auf dem lokalen System gespeichert und sind für andere Community-Mitglieder nicht sichtbar. Dieser Vorgehensweise liegt die Annahme zugrunde, dass eine kollaborative Entwicklung einer Tuning-Plan-Version selten sinnvoll ist, da die DBA die Tuning-Pläne nach ihren eigenen Erfahrungen entwickeln und sich während der Entwicklung mit anderen DBA nicht abstimmen bzw. austauschen wollen (vgl. *Abschnitt 7.1.5.3*).

Auf dem lokalen System können die transienten Versionen von ihren Autoren beliebig oft verändert werden können. Dabei erfolgt keine Revisionsbildung und keine Protokollierung der Änderungsoperationen. Erst bei Veränderungen von freigegebenen Versionen, werden entsprechende Revisionen mit Delta-Informationen angelegt (***Abbildung 7.10a***).

Kollaborative Entwicklung an *transienten* bzw. *freigegebenen* Versionen. Im Gegensatz zur oben genannten Vorgehensweise, wo ausschließlich Versionen im Zustand *released* ausgetauscht werden, könnte man es den DBA dennoch ermöglichen, ihre *transienten* Versionen auf den Community-Server hochzuladen. Damit wäre beispielsweise eine kollaborative Entwicklung einer Tuning-Plan-Version möglich. Mehrere Benutzer könnten somit die *transiente* Tuning-Plan-Version *abwechselnd* auschecken, lokal bearbeiten und anschließend wieder auf den Community-Server hochladen (***Abbildung 7.10b***).

Bei dieser Vorgehensweise ist vom Community-System sicherzustellen, dass diese *transiente* Version entweder nur für eine bestimmte Benutzergruppe (z. B. Kollegen des Tuning-Plan-Autors oder alle Tuning-Plan-Entwickler) sichtbar oder zumindest als *transient* gekennzeichnet und deutlich von Tuning-Plan-Versionen im Zustand *released* unterscheidbar ist, damit die weniger erfahrenen oder außenstehenden Nutzer diese Tuning-Plan-Versionen nicht versehentlich herunterladen. Nachdem die Ent-

wicklung der *transienten* Version abgeschlossen ist und diese für alle Benutzer freigegeben werden kann, ist sie vom letzten Bearbeiter als *released* zu kennzeichnen.

Aufgrund einer möglichen Beteiligung mehrerer Benutzer an der Entwicklung einer *transienten* Version ist eine Protokollierung von erfolgten Änderungen notwendig. Aus diesem Grund werden bei erfolgten Änderungen Revisionen erstellt und mit Informationen über die vorgenommenen Änderungsoperationen angereichert. Somit besteht jederzeit die Möglichkeit, bestimmte Änderungen wieder rückgängig zu machen, indem ein Rollback auf eine ältere Revision einer *transienten* Version erfolgt. Bei möglichen Löschungen von einzelnen Revisionen ist dabei Vorsicht geboten, da es zu Inkonsistenzen (Lücken) in den Delta-Logs kommen kann. Aus diesem Grund sind hier physische Löschungen zu verbieten und durch logische Löschungen (Markierung mit einem Flag „gelöscht“) zu ersetzen (vgl. *Abschnitt 7.1.2.5*).

7.1.6 Auffindbarkeit von Tuning-Plänen

In diesem Abschnitt werden zunächst Konzepte vorgestellt, die die Suche nach den auf dem Community-Server gespeicherten Tuning-Plänen unterstützen. Dazu werden zunächst ausgewählte Konzepte zur Erfassung der Tuning-Plan-Semantik untersucht und bezüglich ihrer Einsetzbarkeit zur Beschreibung der Tuning-Plan-Semantik bewertet (*Abschnitt 7.1.6.1*). Anschließend werden Metadaten definiert, die sowohl für die Suche bzw. Navigation verwendet werden können als auch wichtige Community-Funktionen wie beispielsweise Bewertungs- bzw. Feedback-Vergabe durch die Community-Mitglieder, unterstützen. Diese Metadaten sind als eine Erweiterung der bereits in *Abschnitt 6.1* vorgestellten, auf die Unterstützung der Tuning-Steuerung zugeschnittenen Tuning-Plan-Metadaten, gedacht. Anschließend wird ein Datenmodell vorgestellt, welches zur Verwaltung und Speicherung sämtlicher in dieser Arbeit betrachteter Metadaten eingesetzt werden kann (*Abschnitt 7.1.6.2*). In *Abschnitt 7.1.6.3* werden schließlich mögliche Suchfunktionalitäten vorgestellt, die ein Community-System bereitstellen sollte, um seine Nutzer effektiv zu unterstützen.

7.1.6.1 Metadaten zur Unterstützung der Suchfunktionalitäten

Zur Unterstützung der Suche nach Tuning-Plänen sind insbesondere Konzepte notwendig, die eine Beschreibung der Tuning-Plan-Semantik unterstützen. Zwar spiegeln die in *Abschnitt 6.2* vorgestellten, den Tuning-Plänen zugeordneten Tuning-Absichten die Semantik bzw. die Absicht der Tuning-Pläne wider, jedoch sind diese primär für eine maschinelle Auswertung durch den *Autonomic Tuning Expert* (*Abschnitt 5.2*) zur Laufzeit gedacht. Das zugrunde liegende Modell ist nur beschränkt durch die Benutzer erweiterbar und daher für eine Suche auf dem Community-Server nicht flexibel genug.

Die textuelle Tuning-Plan-Beschreibung enthält eine natürlichsprachige Beschreibung der Tuning-Plan-Absichten sowie der Funktionsweise des Tuning-Plans. Für eine Suche ist ein solches Freitextattribut jedoch nur bedingt ausreichend, da dadurch die bekannten Probleme der Unschärfe bei textueller Beschreibung und Suche in Erscheinung treten.

Zur Definition und Verwaltung von Semantik-beschreibenden Metadaten, die zu einer besseren Auffindbarkeit von Tuning-Plänen auf dem Community-Server beitragen können,

lassen sich verschiedene Technologien wie Ontologien, Taxonomien bzw. Folksonomien einsetzen.

Taxonomien. Eine *Taxonomie* bezeichnet eine abstrakte, hierarchische Struktur. Der Begriff stammt ursprünglich aus der Biologie, hat sich aber in den letzten Jahren nicht nur in den Bibliotheks- und Sprachwissenschaften etabliert, sondern findet auch in der Informationswissenschaft Einzug [Gar04]. Primär werden Taxonomien für Begriffsklassifikationen verwendet. Dabei wird ein *kontrolliertes* (bzw. *geschlossenes*) Vokabular[36], welches aus einer vorgegebenen Menge von Begriffen zu einem bestimmten Thema besteht, in einer Baumstruktur angeordnet. In dieser Struktur gibt es also genau ein Wurzelelement. Zwischen diesem Wurzelelement und seinen Kindelementen ist eine Oberbegriff-Unterbegriff-Beziehung definiert. Diese Oberbegriff-Unterbegriff-Beziehung setzt sich rekursiv bis hin zu den Blättern des Baums fort. Da sich die Taxonomien typischerweise auf bestimmte Themengebiete konzentrieren, ist auch bei den Begriffen eine Kontextabhängigkeit gegeben: Eine Note in einer musikorientierten Taxonomie unterscheidet sich trivialerweise von einer Note in einer Taxonomie aus der Schulwelt.

Eine gängige Erweiterung der Taxonomie bilden Referenzen von den einzelnen Begriffen zu den verwandten Begriffen. In diesem Zusammenhang ist beispielsweise die zu den Taxonomien verwandte Technologie der *Thesauri* zu nennen. Neben der bereits eingeführten Oberbegriff-Unterbegriff-Beziehungen beinhaltet ein *Thesaurus* die Beschreibung der einzelnen Begriffe, die zu jedem Begriff synonymen Begriffe sowie die zu einem Begriff verwandten Begriffe, die keine direkten/indirekten Ober- bzw. Unterbegriffe oder Synonyme des jeweiligen Begriffs sind.

Mittels einer Taxonomie kann eine *Annotation von Tuning-Plänen* folgendermaßen vorgenommen werden: Von Experten wird eine auf das Datenbank-Tuning zugeschnittene Klassenhierarchie vorgegeben, in welche die Nutzer ihre Tuning-Pläne beim Hochladen auf den Community-Server einordnen sollen. Änderungen der Klassenhierarchie durch die Benutzer sind nicht möglich. Die eventuell notwendigen Anpassungen einer vorliegenden Klassenhierarchie sind daher den entsprechenden Experten mitzuteilen. Man könnte es jedoch den Nutzern ermöglichen, Vorschläge bezüglich der Klassenhierarchie-Änderungen zu unterbreiten. Diese könnten anschließend von den Experten freigegeben werden, so dass die vorgenommenen Änderungen für alle Benutzer sichtbar werden.

Es ist zu erwähnen, dass sich die Taxonomien sehr gut für einen navigierenden Zugriff auf die Tuning-Pläne eignen (vgl. *Abschnitt 7.1.6.3*). Dabei kann der Einstieg über das Wurzelelement der Baumstruktur erfolgen. Anschließend ist eine Navigation entlang der die Oberbegriff-Unterbegriff-Beziehungen repräsentierenden Kanten möglich.

So bieten sich beispielsweise die Datenbankproblemkategorisierungsansätze aus *Kapitel 3* für eine Anordnung von Tuning-Plänen an, die einen navigierenden Zugriff darauf unterstützt. Die Tuning-Pläne könnten dabei den Datenbankproblemen zugeordnet werden, die sie adressieren, und diese wiederum nach Problembereichen angeordnet werden (vgl. *Abbildung 3.1*, *Abschnitt 3.1.1*). Nun lassen sich die Tuning-Pläne durch eine Navigation von der Wurzel des Baums zu den Blättern bequem

[36] Bei den später erklärten Ontologien handelt es sich dagegen primär um nicht geschlossene Vokabulare.

erreichen. Zwar bietet dadurch der Einsatz einer Taxonomie eine sehr intuitive Möglichkeit, Tuning-Pläne nach bestimmten Kriterien anzuordnen, jedoch eröffnet sich dabei ein entscheidender Nachteil. Die Hierarchie ist von den gewählten Kriterien abhängig. Eine Berücksichtigung von anderen Kriterien ist in der soeben beschriebenen Hierarchie kaum möglich. Soll eine alternative Navigationsstruktur für den Zugriff auf die Tuning-Pläne bereitgestellt werden, so sind die Tuning-Pläne in beide Hierarchien einzuordnen.

Ein weiterer Nachteil ist eine unzureichende Mächtigkeit bei der Modellierung von äquivalenten Begriffen. Auch die Funktionsweise eines Thesaurus ist hierfür oft nicht ausreichend. Stattdessen kann für eine mächtige und flexible Modellierung des Wissens auf *Ontologien* zurückgegriffen werden.

Folksonomien. Den starren, von Administratoren festgelegten und kontrollierten Hierarchien kann man mit Hilfe einer *Folksonomie* entgegenwirken. Der Begriff *Folksonomie* bildet sich aus den Begriffen *folk* und *taxonomy* und beschreibt die Gesamtheit von Schlagworten (*Tags*), die von einer Community angelegt wurden, um gewisse Inhalte zu erfassen [Sto07, Vos07]. Wichtig ist dabei, dass im Gegensatz zu einer Taxonomie eine Folksonomie keine hierarchische Struktur ist und keine definierten Oberbegriff-Unterbegriff-Beziehungen enthält. Es ist lediglich eine Sammlung von Tags, die von den Nutzern selbst vergeben wurden. Der Kategorisierungsaufwand liegt somit nicht bei einer kleinen Gruppe von Administratoren, sondern bei der Community selbst. Auch die Qualität der Suchergebnisse kann höher sein als bei Taxonomien, da die Nutzer die von ihnen getaggten Informationen nach diesen Tags wieder finden. Man verspricht sich desweiteren, dass eine große Anzahl von Benutzern Informationen und Zusammenhänge sichtbar macht, die ein Einzelner oder eine kleine Gruppe von Administratoren nicht erkannt hätten.

Eine große, ohne Befolgung von Konventionen agierende Community trägt jedoch auch dazu bei, dass Kategorien beispielsweise durch die Verwendung von Synonymen (z. B. *Baby* vs. *Säugling*), verschiedenen Sprachen (z. B. *Foto* vs. *picture*) bzw. einfach nur Begriffen in Singular oder Plural (z. B. *Mensch* vs. *Menschen*) zersplittert werden. So kann ein Teil der Informationen mit einem Begriff kategorisiert werden, während ein anderer Teil der Informationen über einen anderen, im Sinne einer Kategorisierung zum ersten Begriff semantisch äquivalenten Begriff, wiederauffindbar ist.

Ein weiteres Problem ist die Doppeldeutigkeit von einigen Begriffen (Homonyme). Oft ist die Bedeutung eines Begriffs ohne weiteres Kontextwissen nicht erkennbar (z. B. Frucht *Apple* vs. Unternehmen *Apple*).

Den oben genannten Problemen kann durch Einsatz von Technologien, die eine Befolgung von Konventionen und eine Verwendung von kontrollierten Vokabularen erzwingen, zum Teil entgegengewirkt werden.

Zur Annotation von Tuning-Plänen lässt sich eine solche kontrollierte Folksonomie effektiv einsetzen. Die Benutzer können die Tuning-Pläne mit Stichworten markieren. Nach erfolgter Freigabe dieser Stichworte durch einen Administrator (oder einen anderen privilegierten Benutzer) werden diese Stichworte für alle Nutzer sichtbar und können in die Suche einbezogen werden.

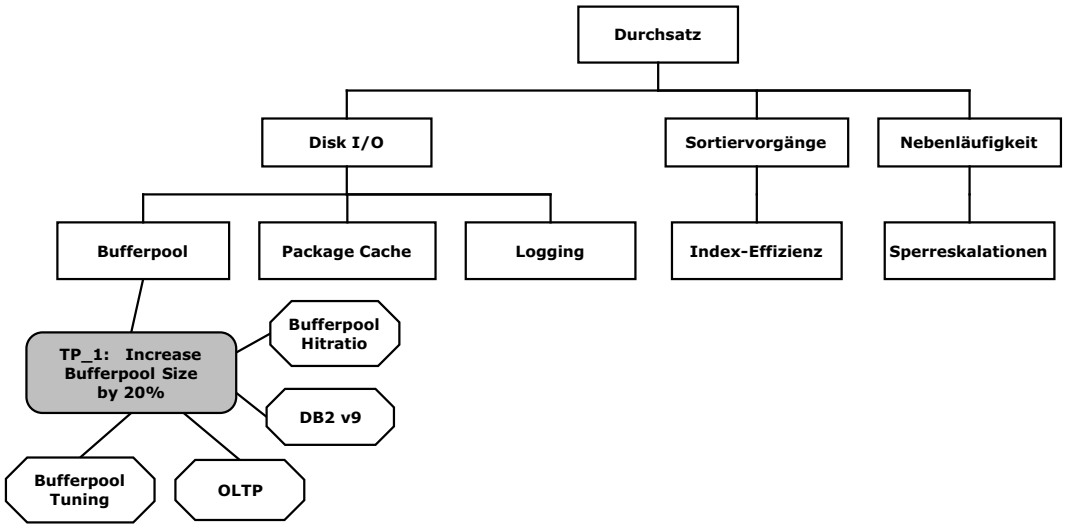


Abbildung 7.11: Kombination einer Taxonomie und Folksonomie

Naheliegenderweise unterstützt eine Folksonomie eine stichwortbasierte Suche von Tuning-Plänen. Es ist dennoch auch möglich, die Folksonomie für den navigierenden Tuning-Plan-Zugriff zu benutzen. Dazu sind jedoch beispielsweise mit Hilfe von semantischen Netzen Beziehungen zwischen den Tags herzustellen. Ein möglicher Ansatz dazu ist in [Cha92] zu finden.

Vielversprechend scheint weiterhin der Einsatz einer Kombination aus einer Taxonomie und einer Folksonomie: Neben einer Einordnung von Tuning-Plänen in vorgegebene Kategorien können die Benutzer die Tuning-Pläne mit eigenen Stichworten versehen. So kann sowohl navigierend als auch stichwortbasiert nach den Tuning-Plänen gesucht werden. In ***Abbildung 7.11*** findet sich ein Beispiel des Zusammenspiels von Tuning-Plan-Kategorien und -Stichworten. Der Bufferpool-Tuning-Plan *TP_1* ist sowohl in die Kategorie *Bufferpool* eingeordnet, als auch mit diversen Stichworten versehen.

Ontologien. Während eines der Hauptnachteile von Folksonomien die Doppeldeutigkeit von Begriffen ist, bieten *Ontologien* die Möglichkeit an, den Kontext der Begriffe zu erfassen und in die Suchanfragen miteinfließen zu lassen. Bei dem Begriff *Ontologie* handelt es sich um eine konzeptuelle Formalisierung eines Wissensbereiches mit Hilfe eines kontrollierten Vokabulars, Beziehungen sowie Ableitungsregeln zwischen den einzelnen Begriffen des Wissensbereichs [HKRS08, UG96]. Da die *Ontologien* einen hohen Aufwand bei ihrer Erstellung erfordern, sollen sie an dieser Stelle nicht näher betrachtet werden.

Es ist zu betonen, dass obwohl sich eine gewisse Ähnlichkeit zwischen den Tuning-Absichten und den in einer Taxonomie angeordneten Begriffen erkennen lässt, ihre Einsatzbereiche verschieden sind. Während die Einordnung der Tuning-Pläne in eine Taxonomie sowie ihre Verschlagwortung von den Community-Benutzern dafür genutzt werden, die Tuning-Pläne semantisch zu beschreiben, werden die Tuning-Absichten eines Tuning-Plans aus-

schließlich von seinem Autor vergeben und vom ATE zur Laufzeit für die Auswahl des das vorlegende Problem am besten auflösenden Tuning-Plans eingesetzt.

Im Gegensatz zu Taxonomien bzw. Folksonomien bieten die Tuning-Absichten eine *maschinell auswertbare* Semantikbeschreibung der Tuning-Pläne. Dabei erfolgt nicht nur eine Zuordnung der Tuning-Pläne zu den entsprechenden Tuning-Absichten, sondern auch eine Verwaltung von Zusammenhängen zwischen den Tuning-Absichten selbst. Diese Zusammenhänge lassen sich dabei durch gerichtete Kanten im Tuning-Absichten-Graphen repräsentieren (vgl. *Abschnitt 6.1.2*). In *Abbildung 6.2* (*Abschnitt 6.1.2*) werden diese Kanten beispielsweise wie folgt interpretiert: Der Durchsatz eines Systems kann also beispielsweise durch eine Erhöhung der Platten-I/O-Geschwindigkeit, eine Beschleunigung der Sortieroperationen bzw. eine Steigerung der Nebenläufigkeit erhöht werden. Die Platten-I/O-Geschwindigkeit wird wiederum durch die Performance der Sortiervorgänge, der Bufferpools, des Package Caches bzw. des Loggings beeinflusst usw.

7.1.6.2 Verwaltung von Tuning-Plan-Metadaten

In *Abschnitt 6.1* wurden bereits ausgewählte Tuning-Plan-Metadaten vorgestellt, die sich in Verwaltungsmetadaten (z. B. Tuning-Plan-Name, Autoren, Erstellungsdatum), deskriptive Metadaten (z. B. Stichwörter, Beschreibung, Tuning-Absichten) und Ausführungsmetadaten (z. B. Effektivität, Einpegelungszeit, Ausführungsdauer, Ausführungskosten) aufteilen lassen. Diese Metadaten waren hauptsächlich auf die Steuerung des Tuning-Prozesses zugeschnitten und berücksichtigten insbesondere keine Community-Aspekte. Der vorliegende Abschnitt erweitert diese Metadaten um weitere Metadaten, die sowohl für die Suche bzw. Navigation verwendet werden können als auch wichtige Community-Funktionen wie beispielsweise Bewertungen oder Feedback der Community-Mitglieder unterstützen.

Eine Verwaltung der Tuning-Plan-Metadaten sowie von weiteren Daten zur Gewährleistung von ausgewählten Repository-spezifischen Community-Funktionen wird durch das in ***Abbildung 7.12*** vorgestellte Datenmodell ermöglicht. In Anbetracht der in diesem Kapitel betrachteten Versionierungsaspekte erfolgt hier eine Aufteilung von Tuning-Plan-spezifischen Metadaten auf Tuning-Plan-Container und Tuning-Plan-Revisionen. Da hier außerdem ein besonderes Augenmerk auf den Community-spezifischen Metadaten liegt, wurde auf eine Berücksichtigung von Ausführungsmetadaten (vgl. *Abschnitt 6.1.3*) verzichtet.

Aus einer im Rahmen der Forschungsarbeiten durchgeführten Studie hat sich ergeben, dass eine Vielzahl von Tuning-Plan-Tags sich auf die Tuning-Plan-spezifischen Hardware- bzw. Software-Parameter bezogen hat. Dazu gehörten beispielsweise die von einem Tuning-Plan optimierten Ressourcen, Angaben zu den für die Funktionsweise des Tuning-Plans notwendigen Hardware- bzw. Software-Komponenten sowie Informationen über die Versionen der eingesetzten Software bzw. Angaben zum Typ der anliegenden Workload, auf welchen der Tuning-Plan zugeschnitten ist. Eine Auswahl solcher Tags findet sich in *Abbildung 7.11*. Um die Semantik dieser Tags zu erfassen und die Freiheitsgrade bei der Stichwort-Gestaltung etwas einzuschränken, wurden spezielle Metadaten eingeführt. Dazu zählen *System-Voraussetzungen*, *betroffene Ressourcen*, *Workload-Klasse* sowie *Tuning-Plan-Klasse* (vgl. *Abschnitt 6.1.2*).

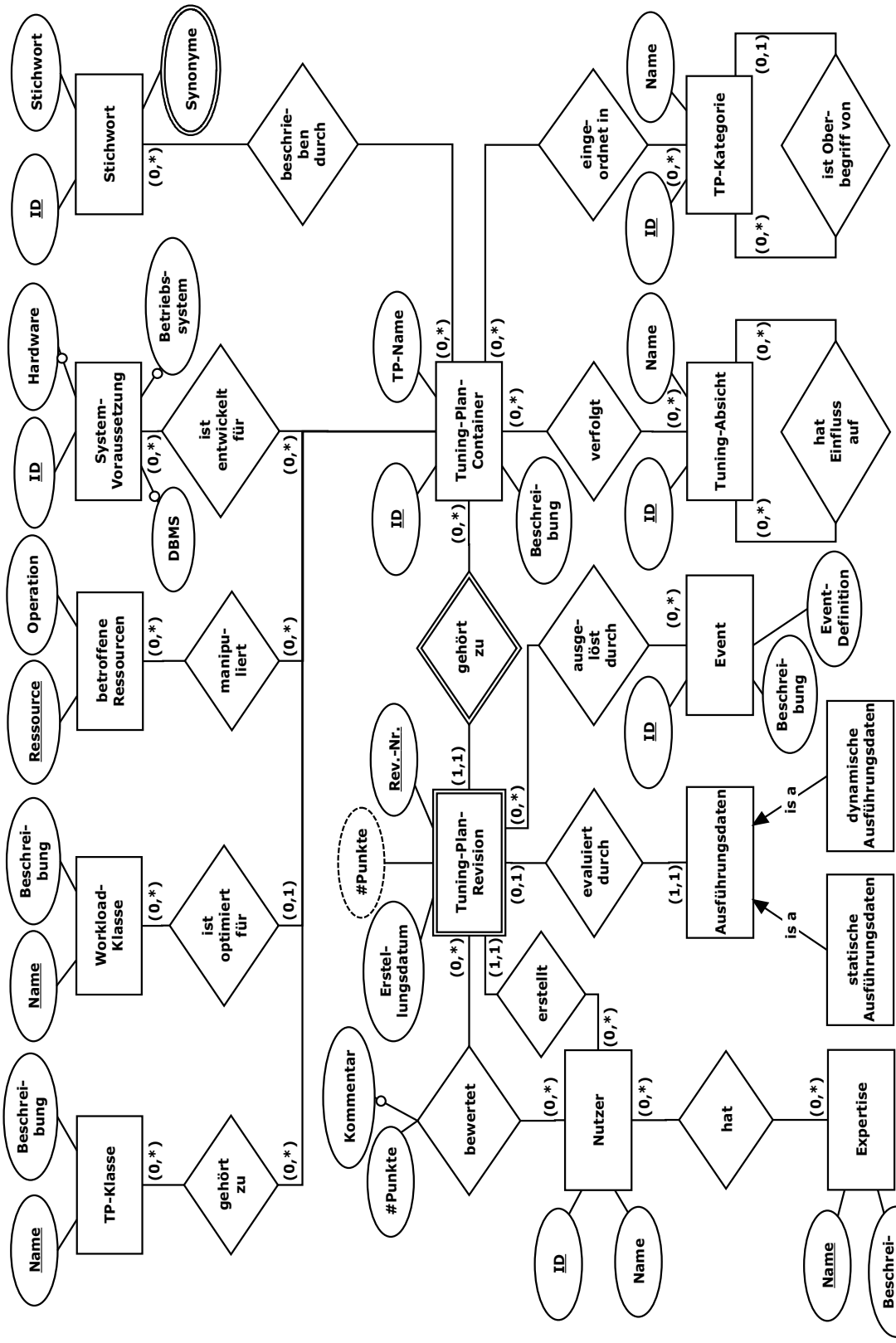


Abbildung 7.12: Datenmodell zur Verwaltung von Tuning-Plan-Metadaten – ein Ausschnitt

Die *System-Voraussetzungen* in *Abbildung 7.12* umfassen aktuell Informationen zu der eingesetzten Hardware, dem Betriebssystem sowie dem Datenbank-Management-System (Entitätstyp *System-Voraussetzung*). Zu den verwalteten Informationen über die *betroffenen Ressourcen* zählt neben einer eindeutigen Ressourcen-ID, die sich beispielsweise aus dem Ressourcennamen und dem Pfad in der Qualifizierungshierarchie (z. B. „*NODE0001.DATA. TBS2*“ – Datenbankinstanz *NODE0001*, Datenbank *DATA*, Tablespace *TBS2*) zusammensetzt, auch die entsprechende durch den Tuning-Plan vorgenommene *Operation* wie *Erstellung*, *Löschung*, *Vergrößerung* oder *Verkleinerung*. Diese Informationen können unter anderem bei der Berücksichtigung von Tuning-Richtlinien ausgewertet werden (vgl. *Abschnitt 6.2*). Die Informationen über die *Workload-Klasse* bestehen aus dem Namen der Workload-Klasse sowie einer Beschreibung. Des Weiteren besteht die Möglichkeit, Tuning-Pläne in Gruppen zusammenzufassen, um sie mit Hilfe von Tuning-Richtlinien zu priorisieren oder bestimmten Ausführungszeitfenstern zuzuordnen. Dies ist mit Hilfe der von Benutzern zu erstellenden *Tuning-Plan-Klassen* möglich (Entitätstyp *TP-Klasse*). So können beispielsweise wartungsspezifische Tuning-Pläne der Tuning-Plan-Klasse *Wartung* zugeordnet und das entsprechende Wartungszeitfenster beispielsweise auf Montag bis Freitag $20 - 23$ Uhr festgelegt werden. Neben einem eindeutigen Namen wird dabei für jede Tuning-Plan-Klasse eine kurze textuelle Beschreibung erfasst.

Die von den Nutzern vergebenen Tags (Entitätstyp *Stichwort*) werden an die Tuning-Plan-Container geknüpft und beziehen sich somit indirekt auf *alle* Tuning-Plan-Revisionen eines Containers. Aus Aufwandsgründen wurde in dieser Arbeit auf die Entwicklung einer Ontologie bzw. eines komplexen Datenmodells zur Verwaltung von Stichwort-Synonymen verzichtet. Das Datenmodell aus der *Abbildung 7.12* beschränkt sich auf eine Auflistung aller synonymen Begriffe zu einem Stichwort im mengenwertigen Attribut *Synonyme*. Des Weiteren lassen sich die Tuning-Pläne in Kategorien einordnen (Entitätstyp *TP-Kategorie*). Dabei handelt es sich um die Begriffe einer Taxonomie (vgl. *Abschnitt 7.1.6.1*). Die hierarchische Beziehung unter den Taxonomie-Begriffen wurde mit Hilfe des rekursiven Beziehungstyps *ist Oberbegriff von* umgesetzt. In Abhängigkeit davon, ob die Tuning-Plan-Container in höchstens eine oder mehrere Kategorien (falls beispielsweise mehrere Taxonomien nebeneinander existieren sollen) eingeordnet werden sollen, ist unter Umständen die Kardinalität vom Beziehungstyp *eingeordnet in* anzupassen.

An dieser Stelle fällt außerdem wieder eine Ähnlichkeit zwischen den Tuning-Plan-Kategorien und den Tuning-Absichten (Entitätstyp *Tuning-Absicht*) auf. Neben den bereits in *Abschnitt 7.1.6.1* erwähnten Unterschieden sind die Tuning-Absichten im Gegensatz zu den Tuning-Plan-Kategorien nicht hierarchisch angeordnet (vgl. die Kardinalitäten der Beziehungstypen *hat Einfluss auf* und *ist Oberbegriff von*).

Im vorliegenden Datenmodell werden außerdem die Community-Mitglieder mit ihren Expertise-Bereichen verwaltet (Entitätstypen *Nutzer* und *Expertise*). Des Weiteren wird erfasst, welche Nutzer welche Tuning-Plan-Revisionen erstellt haben. Eine Erfassung von Änderungsoperationen ist an dieser Stelle nicht nötig. Bei einer Weiterentwicklung eines Tuning-Plans findet, wie in *Abschnitt 7.1.2.3* erläutert, keine Modifikation einer Revision, sondern die Erstellung einer neuen Revision statt. Im Datenmodell in *Abbildung 7.12* wird zunächst von einem einfachen Bewertungssystem ausgegangen. Nutzer können die Tuning-Plan-Revisionen bewerten und zu jeder Bewertung einen Kommentar abgeben (Beziehungstyp *bewertet*). Weitere Techniken zur Einbeziehung der Community-Mitglieder werden in *Abschnitt 7.2* vorgestellt.

Auch Events, die die Tuning-Pläne auslösen, gehören zu wichtigen zu erfassenden Metadaten. In *Abbildung 7.12* wird von einer vereinfachten Verwaltung der Event-Definition im XML-Attribut *Event-Definition* ausgegangen (Entitätstyp *Event*). Eine textuelle Beschreibung des Events wird im Attribut *Beschreibung* abgelegt. Die Events werden den Tuning-Plan-Revisionen und nicht den Tuning-Plan-Containern zugeordnet, da es vorkommen kann, dass eine Tuning-Plan-Revision von einem Event ausgelöst wird, welches von den Events anderer Revisionen des gleichen Tuning-Plans leicht abweicht. Für weitere Informationen zum Thema Events sei hier auf *Kapitel 3* verwiesen.

Wie eingangs erwähnt, wurde in dem vorgestellten Datenmodell auf eine ausführliche Darstellung von Ausführungsdaten verzichtet. Anhand des Entitätstyps *Ausführungsdaten* sowie der abgeleiteten Entitätstypen *statische Ausführungsdaten* und *dynamische Ausführungsdaten* soll lediglich angedeutet werden, dass eine Verwaltung der Ausführungsdaten pro Tuning-Plan-Revision erfolgt, da die einzelnen Revisionen eines Tuning-Plans ein unterschiedliches Ausführungsverhalten aufweisen können.

7.1.6.3 Suchfunktionalitäten

Das Durchsuchen von Repository-Inhalten ist eine der grundlegendsten Repository-Funktionen. Die Suchfunktionalitäten, die für ein Tuning-Plan-Repository in Frage kommen, lassen sich in Anlehnung an die Anforderungen für die Suchfunktionalitäten in Wissensportalen [Weg02] wie folgt unterscheiden:

Suche nach Tuning-Plänen. Die Suche nach bestimmten Tuning-Plänen ist eine klassische Suche nach gespeicherten Objekten, deren Eigenschaften den Suchkriterien entsprechen. Für die Suche eignen sich insbesondere die Metadaten der Tuning-Pläne (*Abschnitte 6.1, 7.1.6.2*). Besonders erwähnenswert sind dabei folgende Suchfunktionalitäten:

Volltextsuche. Es muss nach beliebigen Wörtern und Wortkombinationen in Freitext-Attributen von Tuning-Plänen gesucht werden können. Insbesondere ist diese Art von Suche zum Durchsuchen der Tuning-Plan-Beschreibungstexte geeignet. Eine gewichtete Anzeige der Treffermenge nach relativer Häufigkeit der gesuchten Worte ist dabei besonders hilfreich.

Stichwortsuche. Dabei kann unterschieden werden, ob eine freie Angabe von Stichworten möglich ist oder ob ausschließlich auf eine Auswahl von Stichworten aus einem vorliegenden Stichwortvokabular zurückgegriffen wird. Während die erste Variante auch für eine Suche nach Synonymen eingesetzt werden kann, indem die eingegebenen Stichworte zunächst ausgewertet und auf entsprechende Stichwörter des Vokabulars abgebildet (also „normalisiert“) werden, sorgt die zweite Variante dafür, dass unmittelbar bei der Angabe von Stichworten eine Abbildung auf entsprechende Elemente der zugrunde liegenden Ontologie bzw. Taxonomie (vgl. *Abschnitt 7.1.6.1*) erfolgt.

Suche über Tuning-Absichten. Obwohl die Tuning-Absichten maschinell ausgewertet und vom ATE zur Laufzeit für die Auswahl des das vorliegende Problem am besten auflösenden Tuning-Plans eingesetzt werden, können sie dennoch neben anderen Tuning-Plan-spezifischen Metadaten in die Suche miteinbezogen werden. So

kann beispielsweise nach allen Tuning-Plänen, die für die Steigerung der Bufferpool Hitratio verantwortlich sind, gesucht werden.

Kontextsensitive Suche. Bei Tuning-Plänen kann beispielsweise der Kontext von gesuchten Begriffen berücksichtigt werden, indem neben dem Suchwort angegeben wird, in welchen Tuning-Plan-Elementen (Namen von Tuning-Schritten bzw. Kanten, Werte von bestimmten Parametern etc.) gesucht werden soll.

Suche über Ausführungsmetadaten. Nach einer durchgeführten Suche nach den Tuning-Plänen kann eine Verfeinerung des Suchergebnisses durch die Berücksichtigung von Ausführungsmetadaten wie höchste Effektivität bzw. minimale Ausführungskosten (vgl. *Abschnitt 6.1.3*) vorgenommen werden.

Suche nach Experten (und ihren Beiträgen). Eine Expertensuche, die es ermöglicht, Personen mit Kenntnissen und Fähigkeiten zu bestimmten Themen zu finden, ist von großer Wichtigkeit in einer Community. Die Suche kann beispielsweise nach Stichworten erfolgen, die den gesuchten Expertisebereich beschreiben. Anschließend können alle Beiträge, sprich Tuning-Pläne, der gefundenen Experten angezeigt oder nach bestimmten Kriterien durchsucht werden. Alternativ dazu können die Experten indirekt über ihre Beiträge gefunden werden, indem die Benutzer zu den interessierenden Beiträgen die Autoren angezeigt bekommen. Wurden einmal die entsprechenden Experten gefunden, so können beispielsweise ihre Beiträge abonniert werden. Beim Erstellen neuer bzw. Modifizieren alter Beiträge erfolgt in diesem Fall eine entsprechende Benachrichtigung der Abonnenten (vgl. dazu die Notifikation weiter unten).

Navigierende Suche. Neben einer expliziten Suche finden die Navigationsansätze eine hohe Akzeptanz. Die Navigation bzw. Exploration erfolgt dabei durch Verfolgen von Assoziationen unter den Elementen. Dafür können Verknüpfungen aus einer Autorenschaft bzw. einer Klassifikation (*Abschnitt 7.1.6.1*) oder sogar Bewertungen von Tuning-Plänen (*Abschnitt 7.2*) herangezogen werden.

Speicherung der Suchanfragen. Mit diesem Mechanismus können mehrere Suchanfragen zu einer virtuellen Suchanfrage kombiniert, abgespeichert und somit jeder Zeit wiederverwendet werden. Ein solcher Satz an Suchkriterien ist mit einer *View* aus der relationalen Datenbankwelt vergleichbar und kann als eine dynamische Kollektion von Informationsobjekten angesehen werden, deren Inhalt bei jedem Öffnen der Kollektion durch ein wiederholtes Ausführen der Anfrage ermittelt wird.

Notifikation. Auf dem Konzept der Speicherung der Suchkriterien kann der Notifikationsmechanismus aufbauen, der den Benutzer über Veränderungen in den Suchergebnissen seiner gespeicherten Suchanfragen informiert. Dieser Mechanismus kann außerdem zur Benachrichtigung der Benutzer über neue Beiträge bestimmter Experten zu abonnierten Themen verwendet werden. Dieser Notifikationsmechanismus ist als orthogonal zu den in *Abschnitt 7.1.3* vorgestellten Benachrichtigungsdienst, der für eine Benachrichtigung von Nutzern über die Entstehung von neuen Revisionen der von Ihnen verwendeten Tuning-Pläne sorgt, anzusehen.

Personalisierung. Die Benutzeroberfläche des Community-Servers sollte die Möglichkeit bereitstellen, eine Personalisierung des Portals vorzunehmen. Somit kann beispielsweise konfiguriert werden, dass die gespeicherten Suchanfragen bei jedem Einloggen

des Benutzers automatisch ausgewertet und die Suchergebnisse dem Benutzer präsentiert werden.

Die Suchalgorithmen können um weitere Konzepte angereichert werden, um die Suchqualität zu erhöhen. Mit Hilfe eines Thesaurus können die Synonyme zu den gesuchten Begriffen ermittelt und die Suchanfrage entsprechend erweitert werden, um diese Synonyme zu berücksichtigen. Neben der klassischen booleschen Suche könnte eine unscharfe Suche beispielsweise dazu verwendet werden, Tippfehler in den Suchbegriffen zu erkennen und auszugleichen.

7.2 Community-Funktionen zur Einbeziehung der Nutzer

Online-Communities sind heutzutage aus unserem Leben nicht mehr wegzudenken. Zahlreiche Interessensgemeinschaften nutzen das Internet, um Informationen auszutauschen. Jedoch verspüren viele der Online-Communities das Problem einer ungenügenden bzw. stark differierenden Beteiligung der Community-Mitglieder. Sehr weit verbreitet ist der Effekt, dass nur ein kleiner Teil der Community sich an dem Informationsaustausch aktiv beteiligt (also Informationen bereitstellt), während ein Großteil der Community sich lediglich passiv beteiligt, indem er diese Informationen konsumiert. So sorgen beispielsweise in Open-Source-Entwicklungscommunities 4% der Mitglieder für eine Beantwortung von 50% der Nutzerfragen [Lv00]. 4% der Entwickler produzieren wiederum 88% von neuem Code und 66% von Code-Fixes [MFH02].

Es ist daher sehr wichtig, gute Bedingungen für die Nutzerbeteiligung an der Bereitstellung und Weiterentwicklung von Inhalten zu schaffen. Community-Mitglieder sollen einerseits bei der Benutzung des Community-Systems unterstützt werden, indem ihnen beispielsweise mächtige Suchfunktionalitäten zu Verfügung gestellt werden (vgl. *Abschnitt 7.1.6.3*). Andererseits sollen die Mitglieder zu einer regen Beteiligung motiviert und für ihre Beiträge belohnt werden. Durch diese Anforderungen stellen sich unter anderem Fragen bezüglich der Qualitätsermittlung der Beiträge, der Möglichkeiten zur Motivation und Belohnung von Nutzern und der Funktionalitäten, die eine Benutzbarkeit des Community-Systems verbessern können. Dabei wird die Übertragbarkeit der diskutierten Ansätze auf das Community-System zum Austausch von Tuning-Plänen geprüft, da es im Hinblick auf die kollaborative Entwicklung bzw. Austausch von Tuning-Plänen notwendig ist, die Funktionalitäten des Community-Servers um entsprechende Funktionen zur Einbeziehung der Nutzer zu erweitern.

7.2.1 Arten von Communities

Bevor die einzelnen Community-Funktionen beschrieben und hinsichtlich ihrer Übertragbarkeit auf das Community-System zum Austausch von Tuning-Plänen untersucht werden können, ist es hilfreich festzustellen, um welche Art von Community es sich bei der in diesem Kapitel betrachteten Community von Datenbankadministratoren handelt. Ein Überblick über mögliche Community-Arten sowie einige Vorschläge zur Kategorisierung von Communities finden sich beispielsweise in [Hei04, BMHK99]. Im Kontext einer DBA-Community spielen jedoch primär die inhaltliche Fokussierung der Community sowie ihre Zielgruppe und die Motive der Community-Mitglieder eine wichtige Rolle.

Inhaltliche Fokussierung der Community. Typischerweise hängt das Interesse einzelner Nutzer an einer Community von ihrer thematischen Fokussierung ab. Dabei spielt die Schärfe der Fokussierung eine sehr wichtige Rolle. Bei den Themengebieten einer Community kann es sich einerseits um allgemeine und weit verbreitete Themen wie beispielsweise Informationstechnologie handeln. Andererseits lassen sich Communities auf bestimmte Themen wie Programmiersprachen, Datenbanken oder gar Datenbank-Tuning spezialisieren.

Es ist jedoch eine noch stärkere Spezialisierung einer Community denkbar. So kann sich eine DBA-Community beispielsweise auf die Betrachtung eines bestimmten Datenbank-Systems wie *IBM DB2*, *Oracle Database* oder *Microsoft SQL Server* beschränken.

Grundsätzlich ist eine weitere Einschränkung bezüglich der Produktplattform (beispielsweise *IBM DB2 for Linux, Unix und Windows* vs. *IBM DB2 for z/OS*) sinnvoll. Aufgrund starker Unterschiede in Systemarchitekturen von *IBM DB2 for LUW* und *IBM DB2 for z/OS* unterscheiden sich auch die entsprechenden Tuning-Konzepte und somit auch die Tuning-Pläne gravierend, weshalb eine Trennung beider Communities als sinnvoll erscheint. Sollte jedoch eine solche Trennung nicht erwünscht sein, so sollte das Community-System wenigstens das Bilden von Interessensgruppen innerhalb der Community sowie die Angabe der entsprechenden Produktplattform bei jedem verwalteten Tuning-Plan zulassen und eine strikte Trennung zwischen den beiden Interessensgruppen und ihren Beiträgen sicherstellen.

Eine noch stärkere Spezialisierung auf einzelne Produktversionen ist dagegen in der Regel wenig sinnvoll, da die Versionen sich typischerweise nur geringfügig bezüglich ihrer Architektur voneinander unterscheiden und relativ schnell durch Nachfolgeversionen abgelöst werden.

Zielgruppe der Community. Besonders wichtig ist es, weiterhin zu berücksichtigen, an welche Zielgruppe sich eine Community orientiert. Im Kontext einer DBA-Community ist es beispielsweise vorstellbar, dass ein Datenbank-System-Hersteller ein Community-System ausschließlich für seine Großkunden betreibt und sich verstärkt um die innerhalb dieser Community diskutierten Probleme kümmert, was ein klassisches Beispiel für eine kommerzielle Business-to-Consumer-Community (B2C-Community) ist. Ein anderes Community-System könnte für die Datenbankadministratoren kleinerer Unternehmen eingerichtet werden. In Abhängigkeit von der Zielgruppe einer Community sind auch die bereitzustellenden Funktionalitäten des Community-Systems entsprechend zu konzipieren.

Motive der Community-Mitglieder. Es stellt sich des Weiteren die Frage nach den Motiven der Community-Mitglieder. Dabei sind nach [Bei04] Communities mit *sozialer* (Knüpfen und Unterhalten sozialer Kontakte und Informationsaustausch zu interessensspezifischen Themen), *professioneller* (Austausch des Fachwissens) und *kommerzieller Orientierung* (Erzielung eines geldwerten Vorteils) zu unterscheiden.

Offensichtlich handelt es sich bei dem in diesem Kapitel beschriebenen DBA-Community-System um ein Community-System mit professioneller Orientierung. Dabei bilden die Tuning-Pläne das im Community-System zu verwaltende und in der Community auszutauschende Wissen. Mit speziellen Community-Mechanismen wie Diskussionsforen bzw. Chats wird den Community-Mitgliedern weiterhin eine Kommunika-

tion untereinander ermöglicht. Bei Bedarf lässt sich dieses Community-System um Eigenschaften eines kommerziellen Community-Systems erweitern, bei dem einerseits die Unternehmen und andererseits die professionellen Community-Mitglieder ihre Tuning-Pläne vertreiben können.

Grundsätzlich lässt sich weiterhin zwischen den *öffentlichen* und *privaten* (*geschlossenen*) Communities unterscheiden. Während öffentliche Communities oft breitere Themenbereiche abdecken, stets nach neuen Mitgliedern streben, welche ihre eigenen Ideen und Inhalte einbringen sollen und es für neue Mitglieder relativ einfach ist, den Communities beizutreten, konzentrieren sich geschlossene Communities auf spezifische Themen und setzen für eine Anmeldung am System eine Benutzerverifikation und oft sogar eine Einladung eines Community-Mitglieds voraus.

Bei der hier diskutierten DBA-Community handelt es sich um eine geschlossene Community, die eine strenge Benutzerverifikation verlangt. Insbesondere bei der Einbringung bzw. Veränderung von Tuning-Plänen ist die Vertrauenswürdigkeit der Community-Mitglieder ein wichtiges Thema.

7.2.2 Unterstützung der Nutzer bei der Informationsbereitstellung und -benutzung

Im vorliegenden Abschnitt werden zum einen Techniken zur Motivation und Unterstützung der Nutzer bei der Informationsbereitstellung beschrieben. Diese animieren die Community-Mitglieder dazu, sich am Einpflegen von Informationen in die Community-Datenbank sowie an den Diskussionen bzw. Bewertungsvorgängen aktiv zu beteiligen und belohnen sie für ihren Einsatz. Zum anderen werden hier Ansätze zur Unterstützung der Nutzer bei der Benutzung der im Community-System gespeicherten Information vorgestellt. Als klassische Beispiele für solche Mechanismen sind Suchfunktionalitäten bzw. Empfehlungssysteme zu nennen.

Grundsätzlich existieren verschiedene Mechanismen zur Motivation der Nutzer sich am Informationsaustausch in einer Community zu beteiligen und ihrer Unterstützung dabei. Zu den bekanntesten zählen dabei folgende:

Top-Listen lassen sich nach verschiedenen quantitativen Kriterien (beispielsweise Anzahl der Beiträge) aufstellen. Anhand dieser Kriterien lässt sich anschließend die Platzierung der Community-Mitglieder in diesen Top-Listen in Abhängigkeit von ihren Aktivitäten in der Community ermitteln.

Bewertungssysteme ermöglichen die Abgabe von Bewertungen in Form von Punkten, Noten oder textuellem Feedback. Dadurch lässt sich nicht nur feststellen, ob die Beiträge der Community-Mitglieder als positiv oder negativ wahrgenommen werden, sondern oft auch die Gründe dafür erfahren.

Punktesysteme belohnen Nutzer für ihre Beiträge oder andere Aktivitäten in der Community mit Punkten. Je nach Wichtigkeit der Aktivitäten können dabei die vergebenen Punktezahlen variieren. Die gesammelten Punkte können anschließend in Karriereresystemen Anwendung finden oder in Form einer virtuellen Community-Währung beispielsweise zum Erwerb von Tuning-Plänen eingesetzt werden.

Karrieresysteme definieren eine Ranghierarchie und lassen die Nutzer je nach ihrer Aktivität in dieser Hierarchie aufsteigen. Mit steigendem Rang können dabei entsprechende Rechte und manchmal sogar Pflichten in der Community verbunden sein.

Auszeichnungen lassen sich an Community-Mitglieder für bestimmte Verdienste in der Community vergeben. Die Ermittlung der auszuzeichnenden Nutzer kann dabei entweder durch einen hoch-privilegierten Nutzer wie den Administrator oder aber auch im Rahmen einer Umfrage erfolgen. In bestimmten Fällen kann die Ermittlung der auszuzeichnenden Nutzer auch automatisch beispielsweise anhand des Punktestands erfolgen. Beispiele für Auszeichnungen sind *„Nutzer der Woche“* bzw. *„Autor des besten Tuning-Plans des Monats“*.

Bestrafung der Community-Mitglieder ist für die Ausführung unerwünschter oder verbotener Aktionen in einer Community in Erwägung zu ziehen. Die Bestrafung äußert sich typischerweise im Abzug von Punkten bzw. in einer Verschlechterung des Nutzerstatus (Herunterstufen in der Ranghierarchie bzw. Wegnehmen von Auszeichnungen).

Suchfunktionalitäten für eine komfortable Auffindbarkeit von gespeicherten Informationen bilden nicht nur eine der grundlegendsten Repository-Funktionen, dabei handelt es sich zugleich auch um eines der effektivsten Mechanismen zur Unterstützung der Community-Mitglieder bei der Benutzung des Community-Systems. Unabhängig davon, welche Inhalte in einem Community-System ausgetauscht werden, spielt die Auffindbarkeit dieser eine entscheidende Rolle. Da speziell auf die Suche von Tuning-Plänen zugeschnittenen Suchfunktionalitäten bereits in *Abschnitt 7.1.6.3* vorgestellt wurden, soll auf eine weitere Betrachtung dieses Mechanismus an dieser Stelle verzichtet werden.

Empfehlungssysteme ermöglichen eine Erfassung und Auswertung des Nutzerverhaltens in Community-Systemen. Basierend darauf lassen sich Vorhersagen treffen bezüglich dessen, was die Nutzer interessieren könnte. Diese Informationen können anschließend beispielsweise für eine gezielte Informationsbereitstellung bzw. Abgabe von Empfehlungen verwendet werden.

In nachfolgenden Abschnitten werden einige der weit verbreitetsten Mechanismen exemplarisch vorgestellt und hinsichtlich ihrer Anwendbarkeit im Kontext des Tuning-Plan-Austauschs bewertet.

7.2.2.1 Top-Listen

Bei den *Top-Listen* (auch als *Bestenlisten* bekannt) handelt es sich um ein Verfahren zur Darstellung von Informationen in Form einer Auflistung, bei welcher die Informationen nach einem quantitativen Kriterium automatisch durch das System bewertet und anschließend in die Ergebnisliste aufgenommen werden. Dabei erfolgt eine absteigende Sortierung der Liste anhand des Bewertungsergebnisses (bestes Ergebnis zuerst, schlechtestes Ergebnis zuletzt). Die Anzahl der Elemente einer solchen Liste beschränkt sich oft auf einen bestimmten Wert. Gängig sind dabei Listen mit 5, 10 bzw. 100 Elementen (Top-5-, Top-10- bzw. Top-100-Listen).

Besonders motivierend wirken nach [Hoi07, HAM07] die *Mitglieder-Top-Listen*. Als Kriterien zur Aufnahme in die Listen sind in einer DBA-Community beispielsweise die Anzahl neuer Beiträge, Anzahl vorgenommener Editierungen sowie Anzahl abgegebener Bewertungen vorstellbar. Somit enthalten die Mitglieder-Top-Listen jeweils die Namen der aktivsten Mitglieder bezüglich des für die Erstellung der entsprechenden Liste gewählten Kriteriums. Dabei ist es wichtig zu erwähnen, dass bei der Erstellung von Top-Listen keine Berücksichtigung der Qualität des Tuning-Plans bzw. der abgegebenen Bewertung stattfindet. Einen hohen Rang in einer solchen Top-Liste erreicht ein Benutzer durch eine hohe Quantität der eingestellten oder modifizierten Tuning-Pläne bzw. abgegebener Bewertungen.

Anzahl neuer Beiträge. Bei der Erstellung einer Mitglieder-Bestenliste bezüglich der Anzahl neuer Beiträge ist zu berücksichtigen, dass ausschließlich das Einstellen von getesteten und funktionierenden Tuning-Plänen einen Aufstieg in der Bestenliste nach sich ziehen darf. Um dies sicherzustellen, gibt es mehrere Möglichkeiten. Das Ziel erreicht man beispielsweise durch die Auswertung des Zustands der hochgeladenen Tuning-Pläne (*Abschnitt 7.1.5.5*). Die eventuell hochgeladenen transienten Tuning-Pläne dürfen entweder gar nicht gezählt werden oder müssen weniger Punkte bringen als freigegebene Tuning-Pläne.

Anzahl vorgenommener Editierungen. Bei der hier vorgestellten DBA-Community muss insbesondere die Evolution der Tuning-Pläne gefördert werden. Die vorgenommenen Änderungen an den Tuning-Plänen sind daher genauso wichtig wie die neu eingestellten Tuning-Pläne, da sie eine Beseitigung der Fehler (neue Revisionen) oder Erweiterung der Funktionalität der Tuning-Pläne (neue Varianten) ermöglichen.

Anzahl abgegebener Bewertungen. Communities leben von einem Meinungsaustausch. Die Möglichkeit, Beiträge anderer Mitglieder zu bewerten, ist dabei von großer Bedeutung (*Abschnitt 7.2.2.2*). Durch Einrichten einer Top-Liste, die die aktivsten Mitglieder in Bezug auf die Bewertungen der Beiträge anderer Mitglieder enthält, lassen sich die Mitglieder zum Bewerten animieren. Es muss dabei jedoch sichergestellt werden, dass jedes Mitglied eine bestimmte Tuning-Plan-Revision nur einmal bewerten darf. Das in *Abschnitt 7.1.6.2* vorgestellte Datenmodell ermöglicht bereits das Abspeichern einer einzigen Bewertung eines Nutzers zu einer Tuning-Plan-Revision. Da jedes Mitglied in der Community authentifiziert wird und eine exakte Zuordnung der abgegeben Bewertung dem Mitglied ermöglicht wird, sollte es jedem Mitglied sinnvollerweise auch ermöglicht werden, seine Bewertung bzw. seinen Kommentar zu ändern.

Bei der Erstellung von Top-Listen muss berücksichtigt werden, dass ein soziales Belohnungssystem die Mitglieder nicht ausschließlich für ihre Aktivitäten in der Vergangenheit belohnen darf. Vielmehr sind die Mitglieder insbesondere für ihre aktuellen Aktivitäten zu belohnen. Sollte beispielsweise ein Mitglied in der Vergangenheit besonders viele Beiträge verfasst und sich auf den ersten Platz der Top-Liste hochgearbeitet haben, so haben jüngere Mitglieder kaum Chancen, ihn von seinem ersten Platz zu verdrängen. Eine Einbeziehung des Zeitfaktors ist daher von einer großen Bedeutung. Die Aktivität der Autoren lässt sich beispielsweise unter Berücksichtigung der Zeit wöchentlich oder monatlich ermitteln.

Wie bereits erwähnt, handelt es sich bei der Aktivität der Autoren um ein *quantitatives* Bewertungskriterium. Geht man von der Annahme aus, dass die von vielen Benutzern

heruntergeladenen Tuning-Pläne eine höhere Qualität haben oder eine größere Nachfrage genießen, so lassen sich diese Informationen in die Berechnung der Produktivität der Autoren einbinden. Die Produktivität unterscheidet sich von der Aktivität der Mitglieder durch eine Berücksichtigung des Nutzens der geleisteten Beiträge. Zwar existieren aussagekräftigere Mechanismen zur Ermittlung des Nutzens der Beiträge, jedoch setzen diese ein gewisses Maß an Feedback bezüglich der Qualität der Beiträge durch andere Community-Mitglieder voraus. Dazu zählen beispielsweise die Bewertungsmechanismen (*Abschnitt 7.2.2.2*) bzw. die Berücksichtigung von Ausführungsmetadaten, die die Feststellung der Tuning-Plan-Qualität beispielsweise anhand seiner Effektivität ermöglichen. Die Ermittlung und Berücksichtigung der Download-Raten bzw. der Verwendung der Tuning-Pläne lässt sich hingegen automatisch durchführen und sollte daher bei der Bestimmung der Produktivität neben den anderen Verfahren ebenfalls Anwendung finden.

Ferner lässt sich bei der Ermittlung des Nutzens der Beiträge die Art der Nutzer, die die Tuning-Pläne einsetzen, berücksichtigen. Dieser Überlegung liegt die Annahme zugrunde, dass wenn ein Tuning-Plan von Mitgliedern eingesetzt wird, die entweder selbst regelmäßig Tuning-Pläne einstellen, als Experten gelten oder als Moderatoren bzw. Administratoren in der Community tätig sind, dann hat dieser Tuning-Plan eine höhere Qualität als ein Tuning-Plan, der primär von Community-Neulingen und nicht von den Experten eingesetzt wird. Andererseits ist zu beachten, dass die Experten unter Umständen zwar in der Community aktiv sind und Tuning-Pläne einstellen, selbst jedoch keine Tuning-Pläne von anderen Mitgliedern einsetzen. Das Verhalten der Experten in der Community muss daher am System beobachtet und analysiert werden, was jedoch nicht Gegenstand der vorliegenden Arbeit ist.

Die Idee, den Nutzen der Beiträge zu berücksichtigen, lässt sich auch bei der Erstellung der Top-Listen verwirklichen. So lassen sich die Beiträge zu bestimmten Themen stärker gewichten. Damit bekommen die Administratoren der Online-Community ein Mittel zur Einflussnahme auf die Diskussionsthemen. Im Fall des Tuning-Plan-Austauschs lässt sich damit beispielsweise erreichen, dass von den Mitgliedern vorwiegend Tuning-Pläne zur Lösung bestimmter Problembereiche eingestellt werden. Bei einer Einführung neuer Anwendungen oder neuer Versionen der Community bereits bekannter Anwendungen kann durch die Bekanntgabe einer solchen Gewichtung erreicht werden, dass innerhalb kurzer Zeit ein großer Pool an entsprechenden Tuning-Plänen entsteht.

Auf eine ähnliche Weise können die Beiträge in Abhängigkeit von ihrer Länge (anwendbar bei Texten) bzw. ihrer Komplexität (anwendbar bei strukturierten Dokumenten wie beispielsweise Tuning-Plänen) gewichtet werden. Während die Bestimmung der Textlänge relativ einfach ist, ist es zur zur Bestimmung der Komplexität der Tuning-Pläne kaum ausreichend, die Größe der entsprechenden Dateien auszuwerten. Stattdessen ist auf die Auswertung der verwendeten Basisaktivitäten und der Kontrollstrukturen zurückzugreifen. Es muss jedoch berücksichtigt werden, dass ein Tuning-Plan auch *ungenutzten* (oder *toten*) *Code* [ALSU08, AL00] beinhalten kann, der während einer Tuning-Plan-Ausführung nicht erreicht werden kann. Auf die Forschungsansätze, die sich mit der Identifizierung vom ungenutzten Code beschäftigen, soll hier jedoch nicht eingegangen werden. Weiterhin darf nicht vergessen werden, dass die Länge des Texts bzw. die Komplexität eines Tuning-Plans keinesfalls eine Aussage bezüglich der Qualität dieses Beitrags zulässt. Andererseits kann es jedoch ein Indikator dafür sein, dass der Nutzer bei der Erstellung dieses Beitrags viel Aufwand betrieben hat, den es zur weiteren Motivation der Nutzer zu belohnen gilt.

Durch die Darstellung von Informationen mittels Top-Listen lassen sich die Nutzer nicht nur bei der Informationsbereitstellung, sondern auch bei der Informationsbenutzung unterstützen. Hier erweisen sich beispielsweise Top-Listen mit den meist angesehenen Beiträgen bzw. mit den letzten Beiträgen als sehr hilfreich. In einer DBA-Community würden die entsprechenden Tuning-Plan-Top-Listen beispielsweise nach der Anzahl der Downloads der Tuning-Pläne, nach ihrer Gesamtbewertung bzw. nach dem Datum ihres Hochladens erstellt werden.

Anzahl der Downloads der Tuning-Pläne. Durch die Erstellung einer Top-Liste mit den meist heruntergeladenen Tuning-Plänen lassen sich die Tuning-Pläne identifizieren, die die größte Nachfrage in der Community genießen. Das kann verschiedene Gründe haben: Die Tuning-Pläne können sich einerseits mit Aspekten beschäftigen, die für einen Großteil der Community von Interesse sind. Andererseits kann es sich bei diesen Tuning-Plänen aber auch um die ersten Tuning-Pläne zu einer neu erschienenen Version eines Software-Produkts handeln. Oder es können lediglich Tuning-Pläne sein, die aufgrund ihrer guten Bewertung und einer entsprechenden Platzierung in der Liste der best bewerteten Tuning-Pläne von einer großen Anzahl von Community-Mitgliedern heruntergeladen wurden.

Wie man sieht, bewegt auch diese Top-Liste die Mitglieder indirekt dazu, qualitativ hochwertige Beiträge zu verfassen. Primär geht es hier aber um die Unterstützung der Nutzer bei der Entscheidung, welche Tuning-Pläne es eventuell anzuschauen gilt in der Annahme, dass Tuning-Pläne, die für einen großen Teil der Community interessant sind, auch die meisten einzelnen Mitglieder interessieren.

Anzahl der Bewertungspunkte der Tuning-Pläne. Eine nach diesem Kriterium erstellte Top-Liste enthält die best bewerteten Tuning-Pläne und unterstützt die Nutzer bei der Auswahl der Tuning-Pläne auf dem Community-Server. Im Gegensatz zu den Mitglieder-Top-Listen erfolgt hier eine direkte Berücksichtigung von Qualitätsmerkmalen der Tuning-Pläne durch die Auswertung ihrer Gesamtbewertungen.

Datum des Hochladens der Tuning-Pläne. Fasst man in einer Liste die zuletzt *hochgeladenen* Tuning-Pläne zusammen, so bekommen die Mitglieder die Möglichkeit sich die aktuellen Trends in der Community anzuschauen. Dadurch werden die Mitglieder dazu animiert, diese neu eingestellten Tuning-Pläne auszuprobieren und gegebenenfalls zu kommentieren, was wiederum unter Umständen dazu führt, dass andere Mitglieder diese Tuning-Pläne aufgrund von ihrer Bewertung herunterladen.

In einer leicht modifizierten Form kann eine Liste mit den zuletzt *aktualisierten* Tuning-Plänen erstellt werden. Im Gegensatz zur oben beschriebenen Liste, die nach dem Datum des Hochladens der *ersten* Version eines Tuning-Plans erstellt wird, müsste diese Liste nach dem Datum des Hochladens einer beliebigen Nachfolgeversion erstellt werden.

Nicht nur die Tuning-Plan-Top-Listen unterstützen die Community-Mitglieder bei der Benutzung des Community-Wissens. Durch eine Liste mit den meist verwendeten Suchworten lassen sich beispielsweise, ähnlich wie bei der Liste mit den meist heruntergeladenen Tuning-Plänen, die Trends in der Community erkennen. Dadurch bekommen die Nutzer die Chance, sich rechtzeitig mit einem neuen Thema auseinanderzusetzen.

7.2.2.2 Bewertungen

Ob ein Beitrag bei einer Online-Community als positiv wahrgenommen wird oder nicht, lässt sich manchmal schwer bewerten. Durch eine einfache Auswertung der Anzahl der Zugriffe auf den Beitrag kann eine Aussage bezüglich des allgemeinen Interesses der Community für das entsprechende Thema getroffen werden. Anhand einiger Metadaten kann weiterhin versucht werden, eine Aussage bezüglich der Qualität des Beitrags zu treffen. Eine sehr effektive Methode zur Einschätzung der Qualität der Beiträge besteht jedoch darin, die Community-Mitglieder zu befragen. Dazu lassen sich *Bewertungsmechanismen* (auch bekannt als *Rating*) einsetzen [Hoi07]. Jeder autorisierte Nutzer kann dabei jeden Beitrag bewerten, indem er angibt, ob er den Beitrag nützlich gefunden hat oder für den Beitrag Punkte bzw. eine Note vergibt. So kann ein Benutzer, der einen bestimmten Tuning-Plan heruntergeladen hat, bei seiner nächsten Anmeldung im Community-System bezüglich dieses Tuning-Plans befragt werden.

Zu beachten ist dabei, dass geschlossene Fragen mit vorgegebenen Antwortmöglichkeiten (z. B. *„Fanden Sie diesen Tuning-Plan hilfreich? [Ja/Nein]“* bzw. *„Wie hilfreich fanden Sie diesen Tuning-Plan? [0 – überhaupt nicht hilfreich ... 5 – sehr hilfreich]“* meistens automatisch ausgewertet werden können, während die von Nutzern in Textform abgegebene Kommentare (beispielsweise als Antwort auf die Frage *„Was hat Ihnen an diesem Tuning-Plan besonders gut gefallen?“*) sich kaum auswerten lassen und primär als Hilfestellung für die Community bei der Weiterentwicklung von Tuning-Plänen anzusehen sind. Bei der automatischen Auswertung der Antworten muss berücksichtigt werden, dass die Bewertungsskalen nicht nur bezüglich der möglichen Wertebereiche, sondern auch bezüglich der Bedeutung (z. B. höhere Punktezahlen sind besser oder kleinere Noten sind besser) stark variieren können. Auch die Berechnungsvorschriften zur Bestimmung der Gesamtbewertung können sich von System zu System unterscheiden. Eine typische Berechnungsvorschrift ist dabei die Summe der Bewertungen geteilt durch die Anzahl der Bewertungen (durchschnittliche Bewertung).

Mit Hilfe der Bewertungsmechanismen kann nicht nur die Gesamtqualität der Beiträge eingeschätzt werden. Es besteht weiterhin die Möglichkeit, die Bewertung auf bestimmte Teilaspekte der Beiträge zu konzentrieren. So lassen sich die Tuning-Pläne beispielsweise bezüglich ihrer Nachvollziehbarkeit, Dokumentation oder Effektivität bewerten. Anhand der Bewertungsergebnisse kann anschließend dem Tuning-Plan-Autor aufgezeigt werden, welche Stärken bzw. Schwächen sein Tuning-Plan hat.

Ähnlich wie bei den Top-Listen (*Abschnitt 7.2.2.1*) kann auch hier eine Gewichtung von abgegebenen Bewertungen erfolgen. So lassen sich beispielsweise die Bewertungen von besonderen Nutzergruppen stärker gewichten. Zu solchen Nutzergruppen können Administratoren, Moderatoren, Experten auf entsprechenden Gebieten oder Nutzer mit einer guten Platzierung in den Top-Listen gehören.

7.2.2.3 Punktesystem

Bei dieser Motivationstechnik werden Aktivitäten der Community-Mitglieder mit einem entsprechenden, gegebenenfalls variierenden Punkteguthaben belohnt. Eine Auswahl an möglichen zu belohnenden Nutzeraktivitäten findet sich nachfolgend:

Einstellen von Tuning-Plänen. Für das Einstellen von Tuning-Plänen sind die Community-Mitglieder mit einem entsprechenden Punkteguthaben zu entlohnen. Die Höhe dieses Guthabens kann dabei entweder fix sein oder, ähnlich wie bei den gewichteten Bestenlisten, in Abhängigkeit vom Thema oder Umfang der Beiträge ermittelt werden (vgl. *Abschnitt 7.2.2.1*). Weiterhin können in die Berechnung des Guthabens die Bewertungen anderer Nutzer einbezogen werden. Das Einstellen eines Tuning-Plans, der von vielen Nutzern als gut bzw. nützlich bewertet wurde, würde damit mehr Punkte bringen als das Einstellen eines Tuning-Plans, der überwiegend schlechte Bewertungen erhalten hat.

Weiterentwickeln von Tuning-Plänen. Genauso wie für das Einstellen von neuen Tuning-Plänen sollten die Nutzer auch für die Weiterentwicklung von Tuning-Plänen mit einem Punkteguthaben belohnt werden. Auch hier sind diverse Gewichtungsalgorithmen bei der Ermittlung der Guthabenhöhe vorstellbar.

Herunterladen von Tuning-Plänen. Nicht nur das Einpflegen von neuen Informationen in die Community-Datenbank, sondern auch die Verwendung bereits abgespeicherter Informationen ist in einer Community von Bedeutung. Die Nutzer sollten dementsprechend auch für das Herunterladen von Tuning-Plänen belohnt werden. Wurden die Tuning-Pläne einmal heruntergeladen und ausprobiert, so sind die Nutzer dazu zu veranlassen, eine Bewertung abzugeben.

Abgeben von Bewertungen und Kommentaren. Auch das Bewerten bzw. Kommentieren von Tuning-Plänen kann mit einem entsprechenden Punkteguthaben entlohnt werden. Dabei kann beispielsweise die Länge der abgegebenen Textkommentare bei der Berechnung der Punkteguthabenhöhe ausschlaggebend sein. Die abgegebenen Bewertungen lassen sich durch andere Benutzer bezüglich ihrer Qualität beurteilen. Diese Bewertungen der Bewertung bzw. des Kommentars können ebenfalls in die Formel zur Berechnung des Guthabens miteinbezogen werden. Durch andere Community-Mitglieder positiv bewerteten Kommentare würden somit ein höheres Punkteguthaben implizieren und die Nutzer damit zu begründeten Kommentaren motivieren.

Verfassen von Beiträgen in den Foren. Die meisten Community-Systeme stellen auch Foren zur Ermöglichung von Diskussionen unter den Mitgliedern bereit. Das Verfassen von Beiträgen in solchen Foren ist ebenfalls zu belohnen. Das Punkteguthaben kann hier, wie bei den abgegebenen Kommentaren, in Abhängigkeit von der Länge der verfassten Beiträge errechnet werden.

Lesen von Foren. Eine passive Beteiligung an den Diskussionen lässt sich ähnlich wie die aktive mit Punkteguthaben belohnen. Naheliegenderweise sollte ein Benutzer für das Lesen von Beiträgen in den Foren weniger Guthaben erhalten als für das Verfassen, sein Interesse an den Diskussionen in der Community, sollte aber dennoch belohnt werden.

Wie man sehen konnte, werden bei diesem Ansatz die Community-Mitglieder für ihre Aktivitäten in der Community mit einem entsprechenden Punkteguthaben belohnt. Die Mitglieder müssen aber daran interessiert sein, ihr Guthaben zu erhöhen. Die Möglichkeiten, die den Community-Betreibern in diesem Zusammenhang zur Verfügung stehen, sind sehr vielfältig und sind im Folgenden exemplarisch veranschaulicht:

Erstellung von Top-Listen bezüglich des erreichten Punktestands. Anhand vom erzielten Punktestand lassen sich beispielsweise Top-Listen erstellen, die die Namen der aktivsten Community-Mitglieder enthalten. Alternativ oder ergänzend dazu lassen sich Top-Listen mit den Mitgliedern erstellen, die höchstes Punkteguthaben in einem bestimmten Zeitraum (z. B. Tag, Woche, Monat) angesammelt haben.

Einsatz des Punktestands in Karrieresystemen. Eine weitere Einsatzmöglichkeit findet das Punkteguthaben in Karrieresystemen. Anhand des Punktestands einzelner Mitglieder kann beispielsweise eine Rangzuweisung erfolgen. Mit steigendem Guthaben erarbeitet sich ein Mitglied einen höheren Rang und bekommt somit mehr Rechte in der Community.

Vergleichbarkeit zwischen Mitgliedern. Der Punktstand der Community-Mitglieder ist ein Indikator für ihre Aktivität in der Community und kann daher eine Grundlage für die Vergleichbarkeit der Nutzer untereinander liefern [Hoi07].

Bestrafung der Mitglieder. Des Weiteren liefert ein Punktesystem eine Grundlage für die Bestrafung von Community-Mitgliedern bei Verstößen gegen Community-interne Regeln. Für jede verbotene Aktivität kann ein bestimmter Betrag vom Guthaben des Nutzers abgezogen werden. Sollte sein Guthaben aufgrund von solchen Strafsanktionen unter eine vordefinierte Grenze fallen, so kann dieser Nutzer für einen gewissen Zeitraum gesperrt werden.

7.2.2.4 Empfehlungssysteme

Bei Empfehlungssystemen handelt es sich um ein wichtiges Werkzeug zur Unterstützung der Community-Mitglieder bei der Benutzung der auf dem Community-System gespeicherten Information. Empfehlungssysteme werten Nutzerprofile aus und versuchen anhand von gewonnenen Informationen vorherzusagen, was die Nutzer interessieren könnte [TH01, HK07]. Typischerweise unterscheidet man dabei zwischen den *inhaltsbasierten*, den *kooperativen* und den *hybriden Empfehlungssystemen*.

Inhaltsbasierte Empfehlungssysteme Bei den *inhaltsbasierten Empfehlungssystemen* wird anhand des Feedbacks einer Person ein Profil erstellt, welches bei der Ermittlung von diese Person interessierenden Beiträgen angewendet wird. Bei der Erhebung des Feedbacks wird ein Nutzer entweder vom System explizit zu seinen Interessen befragt (explizite Profilerstellung) oder sein Verhalten wird von System beobachtet und ausgewertet (implizite Profilerstellung) [SKR01]. Der Ansatz basiert also auf einem Vergleich der Nutzerpräferenzen mit den Eigenschaften der zu empfehlenden Beiträge. Insbesondere in einer geschlossenen Community hat dieser Ansatz einen entscheidenden Vorteil, da der Nutzer beim Erstellen seines Community-Profils seine Präferenzen selbst angeben kann und somit primär Diskussionsbeiträge bzw. Tuning-Pläne empfohlen bekommen kann, die seinen Interessen auch tatsächlich entsprechen.

Der Ansatz hat jedoch eine Reihe von entscheidenden Nachteilen. So ist hier beispielsweise das sogenannte *Überspezialisierungsproblem* zu nennen [BS97]. So könnten dem Nutzer aufgrund von seinen Präferenzen sowie seinem Verhalten wiederholt

Tuning-Pläne empfohlen werden, die sich nur minimal von den bereits heruntergeladenen unterscheiden. Andere Tuning-Pläne, die den Präferenzen des Nutzerprofils nicht genau entsprechen, würden stattdessen unberücksichtigt bleiben.

Auch das sogenannte *Kaltstartproblem* [BS97], bei dem einem neuen Benutzer keine Empfehlungen gegeben werden können, da sein Profil noch unbekannt ist, zählt zu den Nachteilen dieses Ansatzes. Dieser Nachteil kann jedoch bei einer geschlossenen Community vernachlässigt werden, da die Benutzer typischerweise nach der Anmeldung und der Erstellung ihres Profils das Community-System oft aufsuchen. Somit werden sowohl die expliziten als auch die impliziten Informationen über die Nutzerpräferenzen relativ schnell gesammelt. Bei den herkömmlichen E-Commerce-Anwendungen ist das Problem jedoch meistens nicht zu vernachlässigen, da viele Nutzer die entsprechenden Verkaufsportale nur selten oder sogar einmalig besuchen und kaum dazu bereit sind, ihre Präferenzen in einem Profil abzuspeichern.

Kooperative Empfehlungssysteme Im Gegensatz zu den inhaltsbasierten Empfehlungssystemen, basieren die *kooperativen Empfehlungssysteme* auf Bewertungen anderer Nutzer [BS97]. Aus den abgegebenen Bewertungen jedes Nutzers lassen sich Bewertungsprofile erstellen. Diese Bewertungsprofile repräsentieren die Interessen der Nutzer, da sie die Informationen dazu enthalten, welche Beiträge durch die einzelnen Nutzer als hilfreich und welche als nicht hilfreich eingestuft wurden. Nun können den Nutzern Beiträge bzw. Tuning-Pläne empfohlen werden, die bereits von anderen Nutzern mit einem ähnlichen Bewertungsprofil positiv bewertet wurden. Neben den Bewertungsprofilen lassen sich hier weitere Informationen wie beispielsweise die Verweilzeiten auf bestimmten Forenseiten in den Vergleichsprozess miteinbeziehen.

Auch bei diesem Ansatz finden sich einige Probleme. Ähnlich wie bei den inhaltsbasierten Empfehlungssystemen kann es hier zum Kaltstartproblem kommen. Einerseits benötigt das System initial eine sehr breite Datenbasis mit Bewertungsprofilen. Andererseits können einem Nutzer, der noch neu im System ist und keine Bewertungen abgegeben hat, keine Empfehlungen unterbreitet werden.

Bei Systemen mit vielen Beiträgen, aber wenigen Nutzern mit Bewertungsprofilen kann es passieren, dass es kaum Ähnlichkeiten in den Bewertungsprofilen gibt, da die Nutzer beispielsweise kaum die gleichen Beiträge beurteilt haben. Dieses Phänomen wird aufgrund von einer leeren Nutzer-Objekt-Matrix (*sparsity*) als das *Sparsity Problem* [MMN02] bezeichnet.

Hybride Empfehlungssysteme Als *hybride Empfehlungssysteme* werden Empfehlungssysteme bezeichnet, die die beiden vorhergehenden Ansätze miteinander kombinieren. Durch eine solche Kombination lassen sich die Probleme wie das Kaltstartproblem bzw. das *Sparsity Problem* vermeiden [HK07]. Im Gegensatz zu einem kooperativen Empfehlungssystem ermöglicht somit ein hybrides System das Unterbreiten von Empfehlungen an Nutzer, deren Interessen sich stark von den Interessen anderer Nutzer unterscheiden.

Ein Beispiel für eine Kombination von inhaltsbasierten und kooperativen Empfehlungsmechanismen bietet das sogenannte *kooperative Objekt-zu-Objekt-Filtern*, wie es von *Amazon* mit den Empfehlungen vom Typ *„Nutzer, die das Produkt X gekauft haben, interessierten sich auch für das Produkt Y“* betrieben wird [HK07, LSY03].

Weiterhin lässt sich durch eine solche Kombination auch das Problem der Überspezialisierung verringern, da hier aufgrund der Betrachtung von Bewertungsprofilen anderer Nutzer die Empfehlungen nicht ausschließlich auf den Eigenschaften der Objekte basieren.

Aufgrund der offensichtlichen Vorteile von hybriden Empfehlungssystemen ist auch im Kontext eines DBA-Community-Systems auf ein hybrides Empfehlungsmechanismus zurückzugreifen. Mit Hilfe dieses Empfehlungsmechanismus können den Community-Mitgliedern Vorschläge bezüglich interessanter Diskussionsthemen bzw. Tuning-Pläne effektiv unterbreitet werden.

7.3 Erweiterung der ATE-Architektur um eine zentrale Community-Plattform zur Verwaltung und zum Austausch von Tuning-Plänen

In diesem Abschnitt wird ein Vorschlag zur Integration von Konzepten, die eine kollaborative Entwicklung und den Austausch von Datenbank-Tuning-Praktiken in einer Community von Datenbankadministratoren ermöglichen, in die bestehende Infrastruktur des *Autonomic Tuning Expert (ATE)* unterbreitet. Eine entsprechende Grobarchitektur des um einige Community-Plattform-Komponenten erweiterten ATE findet sich in ***Abbildung 7.13***. Dabei konzentriert sich die Darstellung auf eine Auswahl von im Kontext dieser Arbeit wichtigsten Komponenten. Auf die Darstellung und Beschreibung weiterer Komponenten, die beispielsweise eine Authentifizierung der Benutzer am Community-Server bzw. die Definition der die Grundlage für den Empfehlungsmechanismus (*Abschnitt 7.2.2.4*) bildenden Benutzerprofile, wurde hier verzichtet. Weiterführende Details zu diesen und weiteren umzusetzenden Komponenten finden sich in [Hof09].

Die Abbildung veranschaulicht außerdem die Arbeitsabläufe zur Erstellung bzw. Weiterentwicklung (blau), Ausführung (rot) und Austausch (grün) von Tuning-Plänen. Nachfolgend findet sich eine kurze Beschreibung entsprechender Systemkomponenten, die sich ebenfalls in drei Kategorien (Tuning-Plan-Erstellung/-Weiterentwicklung, -Ausführung und -Austausch) unterteilen lassen.

Tuning-Plan-Ausführung

Da der MAPE-Kreislauf zur probleminitiierten Ausführung von Tuning-Plänen bereits aus *Abschnitt 5.2* weitgehend bekannt ist, wird auf seine erneute Beschreibung in diesem Abschnitt verzichtet. Es ist lediglich zu erwähnen, dass hier die bereits bekannte ATE-Architektur (in *Abbildung 7.13* gestrichelt umrahmt) um den *Tuning Plan Evaluator* erweitert wurde. Dabei handelt es sich um eine Systemkomponente, die für das Sammeln von dynamischen Ausführungsmetadaten verantwortlich ist, die anschließend im lokalen Tuning-Plan-Repository gespeichert und bei Bedarf auf den Community-Server hochgeladen werden, um beispielsweise die Bestimmung von globalen Ausführungsmetadaten, wie die bisherige Anzahl der Ausführungen einzelner Tuning-Pläne bzw. ihre durchschnittliche Effektivität, zu ermöglichen.

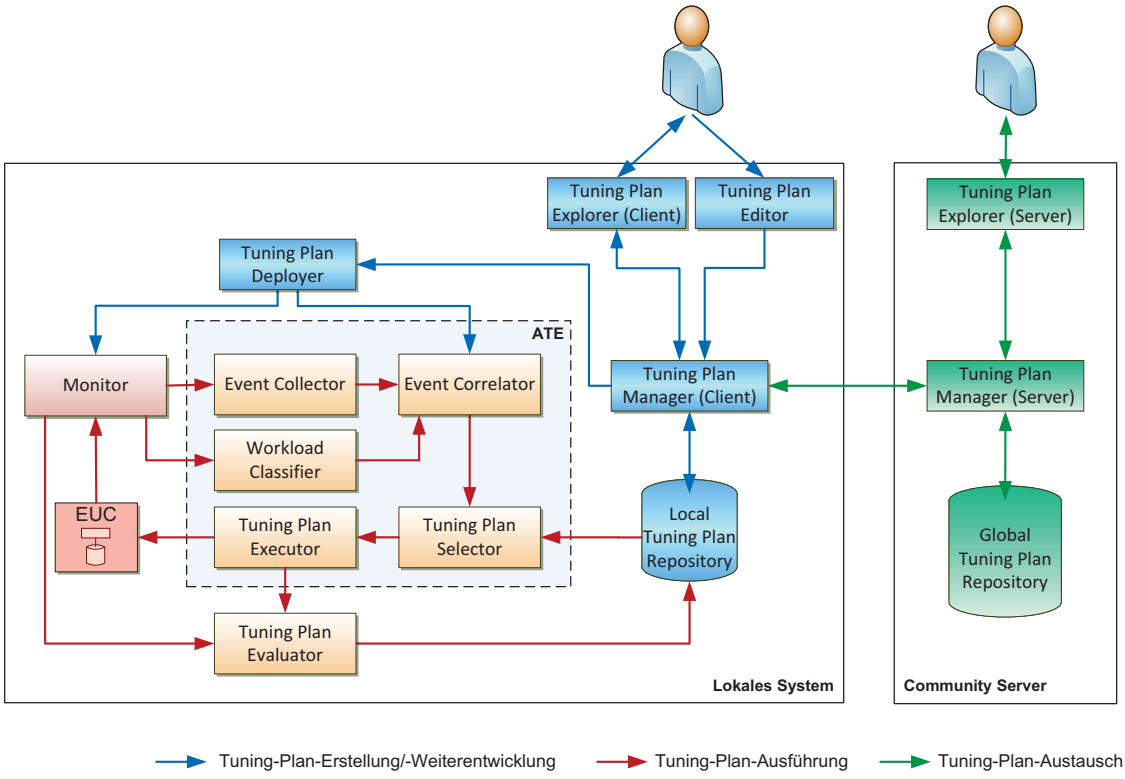


Abbildung 7.13: Grobarchitektur der Community-Plattform – Erstellung, Ausführung und Austausch von Tuning-Plänen

Tuning-Plan-Erstellung/-Weiterentwicklung

Tuning Plan Editor. Der *Tuning Plan Editor* bietet eine graphische Oberfläche zur Erstellung von Tuning-Plänen, zur Definition von entsprechenden, diese Tuning-Pläne auslösenden Events[37] sowie zur Angabe von Tuning-Plan-Metadaten. Bei der Tuning-Plan-Erstellung spielen dabei Aspekte wie die Versionierung bzw. eine kollaborative Entwicklung von Tuning-Plänen eine wichtige Rolle.

Tuning Plan Explorer (Client). Mit Hilfe des *Tuning Plan Explorer Clients* bekommt der Administrator einen benutzerfreundlichen Zugriff auf die im lokalen *Tuning Plan Repository* gespeicherten Tuning-Pläne. Die Unterstützung des Zugriffs erfolgt dabei unter anderem durch die Bereitstellung von entsprechenden Suchfunktionalitäten (*Abschnitt 7.1.6.3*). Weiterhin bietet der *Tuning Plan Explorer Client* die Möglichkeit, die Tuning-Plan-Suche auf dem Community-Server durchzuführen. Dabei lassen sich sowohl die lokal abgelegten Tuning-Pläne auf den Community-Server hochladen, als auch die sich im Community-Repository befindlichen Tuning-Pläne in das lokale *Tuning Plan Repository* herunterladen. Heruntergeladene Tuning-Pläne können nun beispielsweise mit Hilfe des *Tuning Plan Editors* weiterentwickelt oder mit dem *Tuning Plan Deployer* auf dem lokalen System installiert werden. Des Weite-

[37] Die Definition von Events wird in der aktuellen prototypischen Implementierung des Tuning-Plan-Editors nicht unterstützt.

ren ermöglicht der Client-seitige *Tuning Plan Explorer* unter anderem das Bewerten bzw. Kommentieren von lokal getesteten Tuning-Plänen. Schließlich unterstützt diese Komponente die Visualisierung der Evolutionshistorie einzelner Tuning-Pläne und ermöglicht sowohl den Zugriff auf beliebige Revisionen bzw. Varianten der Tuning-Pläne als auch einen (lokalen) Rollback auf ältere Revisionen.

Tuning Plan Manager (Client). Der lokale, Client-seitige *Tuning Plan Manager* bietet eine Schnittstelle zum Zugriff auf das Tuning-Plan-Repository. Neben einer lokalen Versionskontrolle ist diese Komponente für die Kommunikation mit dem Community-Server-seitigen *Tuning Plan Manager* beim Herunterladen sowie Hochladen von Tuning-Plänen bzw. ihren Metadaten verantwortlich.

Local Tuning Plan Repository. Das lokale *Tuning Plan Repository* beinhaltet alle (sowohl aktiven als auch inaktiven) auf dem lokalen System vorhandenen Tuning-Pläne mit ihrer lokalen Evolutionshistorie sowie die Definitionen entsprechender, diese Tuning-Pläne auslösender Ereignisse. Zu den einzelnen Tuning-Plänen werden im Repository weiterhin entsprechende Metadaten (*Abschnitte 6.1, 7.1.6.2*) verwaltet.

Tuning Plan Deployer. Der *Tuning Plan Deployer* ist für die Installation von Tuning-Plänen auf dem lokalen System zuständig. Bei der Tuning-Plan-Installation werden Informationen über die, den Tuning-Plan auslösenden, atomaren sowie komplexen Events aus dem lokalen Tuning-Plan-Repository ausgelesen und entsprechende Anpassungen in den betroffenen Komponenten vorgenommen. Die installierten Tuning-Pläne werden anschließend im Repository als aktiv vermerkt, damit für sie beispielsweise die Sammlung von Ausführungsmetadaten erfolgen kann.

Tuning-Plan-Austausch

Tuning Plan Explorer (Server). Der Server-seitige *Tuning Plan Explorer* bietet Funktionalitäten für einen webbasierten Zugriff auf die im globalen Tuning-Plan-Repository gespeicherten Tuning-Pläne und ihre Metadaten. Insbesondere wird dabei die Darstellung der Evolutionshistorie der Tuning-Pläne (Tuning-Plan-Revisionen bzw. abgeleitete Tuning-Plan-Varianten) ermöglicht. Hat ein Community-Plattform-Nutzer einen ihn interessierenden Tuning-Plan identifiziert, so kann er den Download dieses Tuning-Plans auf sein lokales System anstoßen. Durch die Kommunikation zwischen dem Server- und dem Client-seitigen *Tuning Plan Explorer* lässt sich von hier aus auch die Installation des heruntergeladenen Tuning-Plans auf dem lokalen System initiieren. Ähnlich wie beim Client-seitigen *Tuning Plan Explorer* werden auch hier Community-unterstützende Funktionalitäten bereitgestellt. Neben den vom *Tuning Plan Explorer Client* angebotenen Bewertungs- und Kommentierfunktionen finden hier die insbesondere für einen webbasierten Zugriff geeigneten Mechanismen wie Diskussionsforen, Empfehlungen sowie Darstellung von Informationen in entsprechenden Top-Listen (vgl. *Abschnitt 7.2*) Anwendung.

Tuning Plan Manager (Server). Der *Tuning Plan Manager* des Community-Servers ist das Gegenstück zum Client-seitigen *Tuning Plan Manager* und ist somit für die Übertragung von Daten zwischen den lokalen Systemen und dem Community-Server verantwortlich. Des Weiteren sorgt er für die serverseitige Versionierung von Tuning-Plänen.

Global Tuning Plan Repository. Im globalen *Tuning Plan Repository* werden Tuning-Pläne, Events sowie weitere Tuning -Plan-Metadaten abgelegt. Zur Unterstützung der Evolution von Tuning-Plänen erfolgt hier die Verwaltung ihrer sämtlichen Revisionen und Varianten.

7.4 Zusammenfassung

Heutzutage stehen den Administratoren zahlreiche Informationsquellen (wie Produkthandbücher, Foren oder IT-Blogs) für Datenbank-Tuning-Praktiken zur Verfügung. Bei den darin beschriebenen Tuning-Vorgehensweisen handelt es sich jedoch typischerweise um unstrukturierte Informationen, deren Überführung in formale, maschinell abarbeitbare Tuning-Abläufe sich als ein äußerst zeitraubender und mühsamer Prozess gestaltet.

Zur Unterstützung dieses Prozesses lässt sich eine webbasierte Community-Plattform konzipieren, die eine kollaborative Entwicklung und den Austausch von Datenbank-Tuning-Praktiken in einer Community von Datenbankadministratoren ermöglicht. Die zur Umsetzung einer solchen Community-Plattform notwendigen Konzepte wurden im vorliegenden Kapitel untersucht. Ein Schwerpunkt wurde dabei auf eine effiziente, semantikreiche Speicherung von Tuning-Plänen im Community-Repository gelegt. Da durch die Art der hier beschriebenen Community-Plattform zu erwarten ist, dass eine hohe Anzahl an ähnlichen Tuning-Plänen verwaltet wird, wurde von einer dokumentenbasierten Speicherung von Tuning-Plänen beispielsweise in einem XML-wertigen Attribut abgesehen. Stattdessen wird vorgeschlagen, Tuning-Pläne in ihre Basiskonstrukte wie Tuning-Schritte bzw. Kontrollstrukturen aufzuteilen und anschließend in einer eigens dafür entwickelten Datenstruktur abzuspeichern[38]. Dadurch kann nicht nur eine effiziente Speicherung von Tuning-Plänen unter Ausnutzung der Informationen über ihre Ähnlichkeiten umgesetzt, sondern insbesondere auch ein Einblick in die Tuning-Plan-Semantik durch die Erfassung einzelner Tuning-Plan-Basiskonstrukte im Datenmodell geboten werden.

Diese Vorgehensweise gewährleistet außerdem eine hohe Flexibilität bezüglich der eingesetzten Formate der Tuning-Pläne. Eine Speicherung von Tuning-Plänen in einem Werkzeug-unabhängigen Format und eine Bereitstellung verschiedener Zerlegungs- bzw. Zusammensetzungsalgorithmen ermöglichen nicht nur die Verwaltung von in beliebigen Quellformaten vorliegenden Tuning-Plänen, sondern auch ihre Ausführung mit beliebigen Ausführungswerkzeugen. Es ist also potenziell jederzeit möglich, die Formate des Modellierungs- bzw. des Ausführungswerkzeugs unabhängig voneinander zu verändern. So kann beispielsweise unabhängig vom Format des Tuning-Plan-Modellierungswerkzeugs die Verwendung des jPDL-Formats für die anschließende Ausführung von Tuning-Plänen mit dem *Java Business Process Management Framework* der *JBoss Gruppe* gewählt werden. Die Flexibilität bezüglich der verwendeten Formate ist insbesondere durch die erwartete Langlebigkeit der Community-Plattform und die damit verbundene, potenzielle Evolution der eingesetzten Tuning-Plan-Modellierungs- bzw. -Ausführungswerkzeuge und somit auch ihrer Dateiformate von großer Bedeutung. Aber auch in einer Community, deren Mitglieder verschiedene Tuning-Plan-Modellierungs- und -Ausführungswerkzeuge einsetzen und dennoch einen werkzeugübergreifenden Tuning-Plan-Austausch anstreben, kann von dieser Flexibilität profitiert werden.

[38] Dieses Vorgehen ist auch als *XML-Shredding* bekannt.

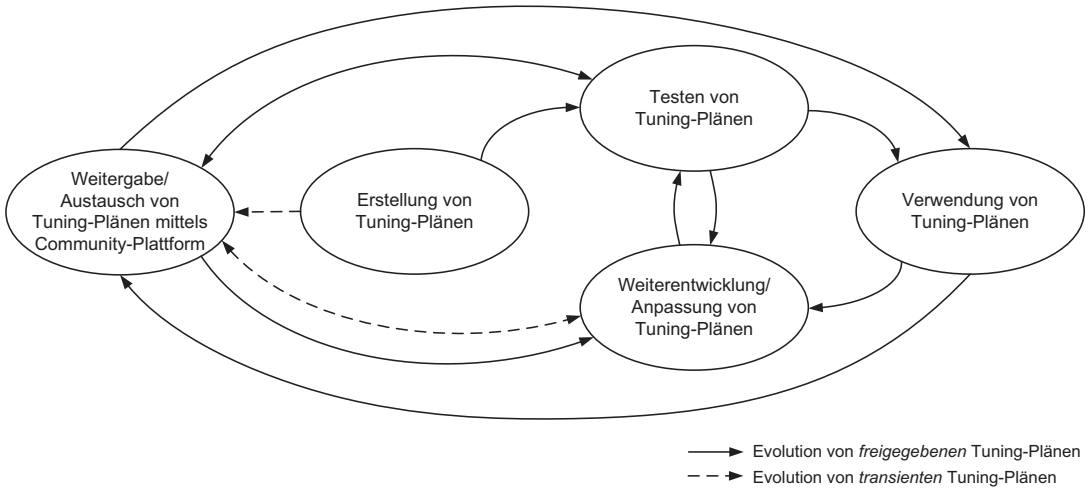


Abbildung 7.14: Entstehung und Evolution von Tuning-Plänen im Mehrbenutzerbetrieb

Dieses Datenmodell ermöglicht nicht nur eine semantikreiche, feingranulare und speicherplatzsparende Speicherung von Tuning-Plänen, sondern bildet auch die Grundlage für das Versionierungsmodell, welches auf eine ähnliche Weise Informationen über die Gemeinsamkeiten zwischen einzelnen Versionen der verwalteten Tuning-Pläne berücksichtigt und lediglich die in den Versionen vorgenommenen Änderungen semantisch auf der Ebene von durch die Änderungen betroffenen Tuning-Plan-Basiskonstrukten verwaltet. Dadurch wird einerseits die Nachvollziehbarkeit der erfolgten Änderungen für die Nutzer gewährleistet. Andererseits hat diese Art der Versionierung Vorteile bei der Zusammenführung von Versionen bzw. bei der Benachrichtigung der Community über die erfolgten Änderungen und ermöglicht unter anderem auch das Nachfahren von Änderungen auf parallelen Entwicklungszweigen (also anderen Varianten des Tuning-Plans), um damit beispielsweise die auf einem Entwicklungszweig stattgefundenen Fehlerkorrekturen auf andere Entwicklungszweige zu übertragen.

Zur Unterstützung der Tuning-Plan-Evolution im Mehrbenutzerbetrieb wurde in diesem Kapitel untersucht, welche Aktionen einzelner Benutzer miteinander „verträglich“ und somit parallel ausführbar sind, für welche Szenarien eine Nebenläufigkeitskontrolle notwendig ist und wie sich diese umsetzen lässt. In diesem Kontext ist zur Steigerung der Nebenläufigkeit die Verwendung des Konzepts der Versionszustände sinnvoll. Dabei lässt sich zwischen *freigegebenen* und somit anderen Nutzern zur Verfügung stehenden Tuning-Plan-Versionen und *transienten* Tuning-Plan-Versionen, die sich noch in Entwicklung befinden und daher lediglich von einer Gruppe von Entwicklern zugreifbar sind, unterscheiden. Die unter Berücksichtigung dieser Konzepte möglichen Szenarien bei der Evolution von Tuning-Plänen im Mehrbenutzerbetrieb sind in ***Abbildung 7.14*** skizziert.

Ein weiterer Schwerpunkt dieses Kapitels wurde auf die Ansätze zur Unterstützung der Nutzer bei der Informationsbereitstellung und -benutzung gelegt. Dazu wurden unter anderem Suchfunktionalitäten beschrieben, die eine intuitive Durchsuchbarkeit des Repositorys ermöglichen sowie Konzepte vorgestellt, die dazu verwendet werden können, Nutzer zu einer regen Beteiligung am Tuning-Plan-Austausch zu motivieren und für ihre Beteiligung

zu belohnen. Schließlich wurde hier ein Vorschlag zur Erweiterung der bestehenden ATE-Architektur zur Ermöglichung einer kollaborativen Entwicklung und eines Austauschs von Tuning-Plänen in einer DBA-Community unterbreitet.

Kapitel 8

Zusammenfassung und Ausblick

Nachfolgend wird zunächst im *Abschnitt 8.1* eine kompakte Zusammenfassung der Ergebnisse der vorliegenden Arbeit gegeben. Anschließend gibt *Abschnitt 8.2* einen Ausblick auf mögliche weiterführende Arbeiten.

8.1 Ergebnisse der Arbeit

Durch die in den letzten Jahrzehnten gestiegene Komplexität von IT-Systemen wurde auch ihre Administration deutlich aufwendiger und kostspieliger. Dieses Problem lässt sich einerseits durch die Entwicklung intelligenter Werkzeuge zur Unterstützung der Administratoren adressieren, wodurch das Problem jedoch lediglich entschärft, aber nicht vollständig gelöst werden kann. Zum anderen wird versucht, die Administrationsvielfalt der Systeme zu reduzieren, indem intelligente Mechanismen, die Teilaspekte der Administration autonom umsetzen, entwickelt und in die Systeme integriert werden.

Diesem Ziel haben sich Initiativen wie beispielsweise das *Autonomic Computing* verschrieben. Sie ermöglichen die Übertragung der komplexen Systemverwaltungs- und -konfigurationsaufgaben an die zu verwaltenden Systeme selbst. Typischerweise wird dabei auf das aus der Kontrolltheorie bekannte Konzept der rückgekoppelten Kontrollschleife zurückgegriffen, wodurch die Systeme ihren eigenen Zustand kontinuierlich überwachen, problematische Abweichungen vom „gesunden“ Zustand erkennen und entsprechende Aktionen einleiten können, die das System in den „gesunden“ Zustand zurück versetzen.

Die vorliegende Arbeit entstand im Rahmen eines Kooperationsprojekts zwischen *IBM* und dem Lehrstuhl, welches das Ziel verfolgte, die Übertragbarkeit der Konzepte des *Autonomic Computing* auf das Datenbank-Tuning zu untersuchen und eine Infrastruktur zur Automatisierung typischer Datenbank-Tuning-Aufgaben unter Minimierung menschlicher Interaktion zu konzipieren. Die wesentlichen Ergebnisse dieser Arbeit werden nachfolgend noch einmal kurz aufgeschlüsselt.

Formalisierung und Modellierung bewährter Tuning-Vorgehensweisen der Datenbankadministratoren

Eine der Grundvoraussetzungen für die Automatisierung der Datenbank-Tuning-Aufgaben bilden die Beschreibung und Modellierung des Tuning-Wissens. Daher wurde in dieser Arbeit zunächst aufgezeigt, wie sich die Problemdiagnose sowie die typischerweise darauf folgenden bewährten Tuning-Vorgehensweisen der Datenbankadministratoren formalisieren

und modellieren lassen. Weiterhin wurde untersucht, welchen Nutzen eine solche Formalisierung bzw. Modellierung bei der Umsetzung des autonomen Tuning datenbankbasierter Software-Systeme und somit bei der Entlastung der Administratoren bei ihren alltäglichen Routinetätigkeiten mit sich bringt.

Beschreibung von Problemen. Zur Ermöglichung einer automatischen Problemerkennung bzw. -diagnose wurde hier vorgeschlagen, auf das Konzept der Ereignisse und ihrer Erkennung sowie Verarbeitung zurückzugreifen. Insbesondere aufgrund der angestrebten Integration von typischerweise in komplexen IT-Infrastrukturen bereits vorhandenen, sich über mehrere Schichten der IT-Infrastruktur erstreckenden Überwachungswerkzeugen hat sich ein zweistufiger Ansatz als effektiv erwiesen. Im ersten Schritt erfolgt dabei die Definition von problematische Situationen repräsentierenden, „atomaren“ Ereignissen mittels dieser Überwachungswerkzeuge. Anschließend erfolgen im zweiten Schritt eine Konsolidierung dieser Ereignisse aus sämtlichen aktiven Überwachungswerkzeugen und eine darauf durchgeführte Erkennung von vorab definierten Ereignismustern, wodurch sich die aufgetretenen Probleme genau eingrenzen lassen.

Beschreibung von Tuning-Vorgehensweisen. In Reaktion auf erkannte Probleme lassen sich entsprechende, diese Probleme lösende Tuning-Aktionen initiieren. Um Administratoren zu unterstützen und eine benutzerfreundliche Modellierung von ihren bewährten Tuning-Vorgehensweisen zu ermöglichen, wurde in dieser Arbeit eine erweiterbare Sprache entworfen, die einen Satz von vordefinierten semantisch reichen, Tuning-spezifischen Basisaktivitäten bereitstellt und mittels Kontrollstrukturen ihre Verknüpfung zu speziellen Tuning-Workflows (*Tuning-Plänen*) ermöglicht.

Anschließend wurde untersucht, welche Werkzeuge sich zur Modellierung von Tuning-Plänen eignen und wie sie sich effektiv einsetzen lassen. Ein besonderes Augenmerk wurde dabei auf Workflow-Management-Systeme und ihre Funktionalitäten bei der Modellierung von Arbeitsabläufen gelegt. Nach einigen mit *IBM WebSphere Integration Developer* und *IBM WebSphere sMash (WsM)* durchgeführten Tests fiel die Wahl auf den WsM, der sich aufgrund seiner Leichtgewichtigkeit und seiner leichten Erweiterbarkeit für die prototypische Umsetzung eines Tuning-Plan-Editors und einer entsprechenden -Ausführungskomponente am besten eignet.

Vorstellung einer Infrastruktur zur probleminitiierten Ausführung von Tuning-Plänen

Des Weiteren wurde in dieser Arbeit ein Architekturvorschlag zum best-practices-orientierten Datenbank-Tuning vorgestellt. Die Architektur des *Autonomic Tuning Expert (ATE)* baut dabei auf den Konzepten zur Formalisierung des Tuning-Wissens auf und ermöglicht eine kontinuierliche Überwachung eines zu optimierenden Systems durch die Einbindung beliebiger, bereits vorhandener oder neu eingerichteter Überwachungswerkzeuge und eine darauf aufbauende Erkennung von aufgetretenen Problemen. In Reaktion auf erkannte Probleme oder möglicherweise problematische Trends wird anschließend die Ausführung von Tuning-Plänen eingeleitet, die im Vorfeld von Administratoren zur Behebung von diesen Problemen erstellt wurden.

Basierend auf einer Auswahl Workload-relevanter Systemmetriken ermöglicht dabei eine Workload-Erkennungskomponente eine System- und Tuning-unabhängige Erkennung des Typs der anliegenden Workload, der sowohl den Überwachungs- als auch den Tuning-Prozess entscheidend beeinflusst. Im Rahmen einer Machbarkeitsstudie wurde anschließend ein Satz von auf das Tuning eines DB2-Systems zugeschnittenen Tuning-Plänen entwickelt und zur Validierung der vorgestellten Architektur unter einer wechselnden Workload eingesetzt. Durch eine Auswertung ausgewählter Performance-relevanter Metriken konnte dabei die Effektivität des Tuning nachgewiesen werden.

Konzepte zur Steigerung der Planungsqualität im autonomen Datenbank-Tuning

Der in dieser Arbeit vorgestellte Ansatz zeichnet sich durch seine Universalität, Modularität und seine leichte Erweiterbarkeit auf beliebige Hard- und Software-Architekturen aus. Der Zugriff auf die zu tunenden Komponenten erfolgt dabei ausschließlich über die nach Außen sichtbaren Schnittstellen und es wird kein Einblick in die Systeminterna benötigt. Dennoch kann von Informationen, wie die Abhängigkeiten unter den einzelnen Sensoren und Effektoren, die mit der Änderung der Effektoren verbundenen Kosten bzw. die Historie der Tuning-Plan-Ausführungen sowie ihrer damaliger Effektivität bei der Planung profitiert werden.

Metadaten und Wissensmodell. Aus diesem Grund wurden in dieser Arbeit Metadaten identifiziert, die zur Unterstützung der Tuning-Steuerung eingesetzt werden können und beispielsweise die Auswahl von zur Auflösung eines erkannten Problems unter gegebenen Nebenbedingungen geeigneten Tuning-Plänen unterstützen oder sogar einen Einfluss auf die Ausführung dieser Tuning-Pläne haben. Diese Metadaten wurden als Grundlage für die Konstruktion eines Wissensmodells für das autonome Datenbank-Tuning verwendet. Dieses Wissensmodell stellt dabei nicht nur das für die einzelnen Phasen des Tuning-Prozesses notwendige Wissen bereit, sondern ermöglicht auch die Weitergabe des in diesen Phasen gesammelten bzw. generierten Wissens an entsprechende Komponenten des Tuning-Systems.

Tuning-Richtlinien. In der Regel kann das Verhalten autonom agierender Systeme indirekt durch die Definition von entsprechenden Zielvorgaben beeinflusst werden. So wurde in dieser Arbeit ein Vorschlag für eine XML-basierte Sprache zur Definition von Tuning-Richtlinien unterbreitet und das dieser Sprache zugrunde liegende Modell vorgestellt. Die Tuning-Richtlinien ermöglichen es den Administratoren einerseits, Zielvorgaben an das zu tunende System zu definieren, Restriktionen für bestimmte Ressourcen anzugeben bzw. einzelne Datenbankobjekte und Anwendungen zu priorisieren und entsprechend im Tuning-Prozess, beispielsweise bei der Speicherverteilung, zu berücksichtigen. Andererseits können mit Hilfe von Tuning-Richtlinien weitere beim Tuning zu berücksichtigende Nebenbedingungen, wie beispielsweise das in bestimmten Zeitabschnitten zu erwartende Nutzerverhalten, angegeben werden, deren Kenntnis sowohl beim reaktiven als auch beim proaktiven Tuning Anwendung finden kann.

Hypothesenbäume. Während sowohl die Metadaten als auch die Tuning-Richtlinien unter anderem dazu verwendet werden, die Planungsphase des Tuning-Prozesses indirekt

zu beeinflussen, wurden in dieser Arbeit weiterhin Konzepte erarbeitet, die eine unmittelbare Einflussnahme auf die Planung durch die Modellierung des gesamten Planungsablaufs ermöglichen. Mit Hilfe von sogenannten Hypothesenbäumen wird den Administratoren die Möglichkeit geboten, die gesamte Planungslogik für das Tuning einzelner Problembereiche manuell zu erfassen und dem System zur Verfügung zu stellen, wodurch in den meisten Fällen eine deutliche Beschleunigung des Problemdiagnose- und -behebungsprozesses zu erwarten ist.

Konzepte zur zentralisierten Verwaltung und Austausch von Tuning-Plänen mittels einer Community-Plattform

Heutzutage stehen den Administratoren zahlreiche Informationsquellen für Datenbank-Tuning-Praktiken zur Verfügung. Bei den darin beschriebenen Tuning-Vorgehensweisen handelt es sich jedoch typischerweise um unstrukturierte Informationen, deren Überführung in formale, maschinell abarbeitbare Tuning-Pläne sich als ein äußerst zeitraubender und mühsamer Prozess gestaltet.

Zur Unterstützung dieses Prozesses wurde in dieser Arbeit eine webbasierte Community-Plattform konzipiert, die eine kollaborative Entwicklung und einen Austausch von Datenbank-Tuning-Praktiken in einer Community von Datenbankadministratoren ermöglicht. Ein Schwerpunkt der Arbeit wurde dabei auf eine effiziente, feingranulare, semantikreiche Speicherung von Tuning-Plänen im Community-Repository gelegt. Durch die gewählte Art der Tuning-Plan-Speicherung wird eine hohe Flexibilität bezüglich der verwendeten Tuning-Plan-Formate gewährleistet. So werden einerseits die Verwaltung von in beliebigen Quellformaten vorliegenden Tuning-Plänen und andererseits auch ihre Ausführung mit beliebigen Ausführungswerkzeugen ermöglicht, was insbesondere durch die erwartete Langlebigkeit der Community-Plattform und die damit verbundene, potenzielle Evolution der eingesetzten Tuning-Plan-Modellierungs- bzw. -Ausführungswerkzeuge und somit auch ihrer Dateiformate von großer Bedeutung ist.

Das für die Speicherung von Tuning-Plänen erstellte Datenmodell bildet weiterhin die Grundlage für das Versionierungsmodell, welches auf eine ähnliche Weise Informationen über die Gemeinsamkeiten zwischen einzelnen Versionen der verwalteten Tuning-Pläne berücksichtigt und lediglich die in den Versionen vorgenommenen Änderungen semantisch auf der Ebene von durch die Änderungen betroffenen Tuning-Plan-Basiskonstrukten verwaltet. Dadurch wird einerseits die Nachvollziehbarkeit der erfolgten Änderungen für die Nutzer gewährleistet. Andererseits hat diese Art der Versionierung Vorteile bei der Zusammenführung von Versionen bzw. bei der Benachrichtigung der Community über die erfolgten Änderungen und ermöglicht unter anderem auch das Nachfahren von Änderungen auf parallelen Entwicklungszweigen.

Zur Unterstützung der Tuning-Plan-Evolution im Mehrbenutzerbetrieb wurden einige Konzepte zur Nebenläufigkeitskontrolle untersucht und speziell im Hinblick auf eine kollaborative Tuning-Plan-Entwicklung angepasst. Einen weiteren Schwerpunkt dieser Arbeit bilden die Ansätze zur Unterstützung der Nutzer bei der Informationsbereitstellung und -benutzung. Zum einen zählen dazu Suchfunktionalitäten, die eine intuitive Durchsuchbarkeit des Repositorys ermöglichen. Zum anderen sind es Konzepte, die dazu verwendet werden können, Nutzer zu einer regen Beteiligung am Tuning-Plan-Austausch zu motivieren und für ihre Beteiligung zu belohnen. Schließlich wurde hier ein Vorschlag zur

Erweiterung der bestehenden ATE-Architektur um Komponenten, die eine kollaborative Entwicklung und einen Austausch von Datenbank-Tuning-Praktiken in einer Community von Datenbankadministratoren ermöglichen, unterbreitet.

8.2 Weiterführende Arbeiten

Mit der vorliegenden Arbeit ist die Entwicklung an den vorgestellten Konzepten nicht als abgeschlossen anzusehen, vielmehr bietet die Arbeit verschiedene Ansatzpunkte für Erweiterungen und Verbesserungen der Konzepte und der hier vorgestellten Implementierungen. Im Folgenden wird eine Auswahl der wesentlichen Ansatzpunkte vorgestellt.

Ereignis-Definition und -Auswertung. In dieser Arbeit wurde gezeigt, wie die Problemdiagnose auf die Definition und automatische Erkennung von Ereignissen zurückgeführt werden kann. In weiteren Arbeiten sollte eine intuitive Sprache zur Definition von solchen (komplexen) Ereignissen entwickelt werden. Aufbauend darauf sind unter anderem Konzepte notwendig, die eine Interpretation von solchen Ereignis-Definitionen und eine automatische Aufteilung der definierten Ereignisse auf Teil-Ereignisse, die sich anschließend an entsprechende Überwachungswerkzeuge verteilen lassen, ermöglichen.

Datensammlung und -austausch durch Tuning-Pläne. Im Rahmen des Tuning kann es vorkommen, dass die verwendeten Überwachungswerkzeuge nicht ausreichend Daten für eine Problemdiagnose und -auflösung bereitstellen können, weil beispielsweise die Intervalle zwischen den Datensammlungszeitpunkten zu groß sind oder bestimmte Informationen durch die Überwachungswerkzeuge einfach nicht erfasst werden. In solchen Fällen kann es förderlich sein, wenn die Tuning-Pläne dahingehend erweitert werden, dass sie selbst Daten sammeln und diese anschließend selbst verwerten bzw. anderen Tuning-Plänen zur Verfügung stellen können. Dazu sind Konzepte zur Ermöglichung einer Zwischenspeicherung der gesammelten Daten zu entwickeln. Die Verwendung einer speziellen Datenstruktur, die nicht nur das Speichern von gesammelten Informationen, sondern auch ihr Annotieren mit entsprechenden Metadaten unterstützt, ist ratsam. Dabei ist neben dem Ablegen von Informationen durch die Daten sammelnden Tuning-Pläne insbesondere auch das Abrufen dieser Daten durch andere Tuning-Pläne zu ermöglichen.

Des Weiteren ist zu berücksichtigen, dass während einige Tuning-Pläne sich auf das Sammeln von Informationen konzentrieren, andere Tuning-Pläne den Zugriff auf diese Informationen für ihre Funktionsfähigkeit voraussetzen können. Anhand von solchen Abhängigkeiten unter den Tuning-Plänen lassen sich Tuning-Plan-Pakete bilden, die die jeweils aktuellen Versionen einzelner Tuning-Pläne beinhalten und als Austauschgranulat innerhalb der Community verwendet werden können.

Parametrisierung von Tuning-Plänen. Während die Konzepte zur Bildung von Tuning-Plan-Revisionen bzw. -Varianten bereits einen wesentlichen Beitrag zur Reduktion der Flut von ähnlichen Tuning-Plänen auf dem Community-Server leisten, kann die Vielfalt von Tuning-Plan-Varianten durch eine Parametrisierung von Tuning-Plänen

weiter eingeschränkt werden. Liegen mehrere Tuning-Pläne vor, die sich beispielsweise lediglich anhand von verwendeten Schrittweiten einzelner Tuning-Schritte unterscheiden (z. B. Erhöhung der Bufferpool-Größe um 10% in einem Tuning-Plan und um 30% in einem anderen Tuning-Plan), so ist eine Zusammenfassung dieser Tuning-Pläne zu einem neuen parametrisierten Tuning-Plan möglich, der zur Laufzeit die passenden Werte für die parametrisierten Tuning-Schritte ermittelt. Dieses Konzept kann noch weiter ausgebaut werden, um beispielsweise die Parametrisierung anhand ganzer Tuning-Pfade zu ermöglichen.

Automatische Tuning-Plan-Komposition. Eine der möglichen Erweiterungen des im Rahmen dieser Arbeit vorgestellten best-practices-orientierten Ansatzes ist eine automatische Erzeugung von Tuning-Plänen durch Komposition von vordefinierten, im Repository abgelegten Tuning-Schritten oder Tuning-Schritt-Blöcken und eine Anpassung dieser Tuning-Pläne zur Laufzeit. Zur Ermöglichung einer automatischen Komposition von Tuning-Plänen sind insbesondere Konzepte zur Erfassung der Semantik der einzelnen Tuning-Schritte bzw. ihrer Blöcke unabdingbar. Eine solche Semantikerfassung ist beispielsweise mit Hilfe von Ontologien realisierbar.

Steigerung der Planungsqualität. Im Rahmen weiterer Arbeiten ist unter anderem die Planungsqualität des Prototyps zu verbessern. Das in der Arbeit zur Unterstützung des autonomen Tuning und insbesondere der Planung vorgestellte Wissensmodell bietet dazu zahlreiche Ansatzpunkte. So sollten beispielsweise die Auswahl und anschließende Ausführung von Tuning-Plänen unter Berücksichtigung ihrer Kosten und des durch ihre Ausführung erzielten Nutzens durchgeführt werden. Durch die Entwicklung eines Kosten-Nutzen-Modells, welches diese Informationen für die einzelnen Tuning-Aktivitäten oder ganze Tuning-Pläne erfasst, und durch die Auswertung dieses Modells zur Laufzeit kann die Planungsqualität verbessert werden.

Einsatz des Prototyps in komplexen IT-Umgebungen. Im Rahmen weiterführender Arbeiten sollte die Funktionsweise und Effizienz des in dieser Arbeit vorgestellten Prototyps in einer komplexen IT-Umgebung mit mehreren interagierenden Komponenten, die im Rahmen des Tuning aufeinander abgestimmt werden müssen, getestet werden. Dazu sollen einerseits die Sensor- und Effektorschnittstellen dieser Komponenten untersucht, entsprechende Adapter des *Event Collectors* konfiguriert und auf den durch diese Komponenten bereitgestellten Effektoren aufbauende Tuning-Schritte erstellt werden. Andererseits sind für Testzwecke Tuning-Pläne zu erstellen, die das komponentenübergreifende Tuning umsetzen.

Weiterentwicklung von Tuning-Richtlinien. Aus dem hier eingeführten Konzept der Tuning-Richtlinien ergeben sich einige Herausforderungen für weitere Forschungsarbeiten. Durch die Entwicklung einer Komponente zur Auswertung des aktuellen bzw. des nach Vornehmen bestimmter Konfigurationsänderungen erwarteten Systemzustands und eines Kosten-Nutzen-Modells lassen sich *Goal Policies* umsetzen, die eine Vorgabe des Zielzustands und seine Erreichung im Tuning-Prozess unter Einhaltung bestimmter Nebenbedingungen, wie maximal akzeptable Kosten, ermöglichen.

Grundsätzlich entspricht der Zielzustand eines Systems einer Menge von Performance-Metriken mit entsprechenden durch ihre Werte einzuhaltenden Wertebereichen. Es ist daher weiterhin zu untersuchen, ob aus einer (abstrakten) Vorgabe des Zielsystemzustands eine automatische Ableitung von (weniger abstrakten) zu

erreichenden Zwischenzielen bzw. unter Umständen sogar Events und der damit verknüpften Tuning-Aktionen möglich ist. Sämtliche am Policy-Modell vorgenommenen Erweiterungen sind anschließend in den *Tuning Coordinator* zu integrieren.

Weiterentwicklung des Hypothesenbaum-Ansatzes. Auch der Hypothesenbaum-basierte Ansatz zur nutzergetriebenen Planung bietet eine Grundlage für weitere Arbeiten. So ist zum einen der Austausch von Hypothesenbäumen mit zugehörigen Tuning-Plänen in der Community zu ermöglichen. Dazu sind sowohl entsprechende Speicherungs- und Versionierungsmodelle als auch Konzepte zum Mehrbenutzerbetrieb zu entwickeln. Konsequenterweise sind auch Konzepte notwendig, um eine automatische Zusammenführung mehrerer Hypothesenbäume, die einem *Top Event* zugeordnet sind, zu unterstützen.

Prototypische Umsetzung der Community-Plattform. Die in dieser Arbeit konzipierte Community-Plattform sollte zukünftig prototypisch umgesetzt werden. Auf der Basis des Prototyps können anschließend Messungen bezüglich der Effektivität der hier vorgeschlagenen Speicherungs- und Versionierungsmodelle durchgeführt werden. Dabei könnte dieser Speicherungs- und Versionierungsansatz mit einer dokumentenbasierten Speicherung unter Zuhilfenahme von DBMS-internen Mechanismen (wie beispielsweise *DB2 pureXML*) zur Speicherung von XML-wertigen Attributen verglichen werden. Bei einem solchen Vergleich sind die mit einer Speicherung von Tuning-Plänen in XML-wertigen Attributen verbundenen Transformationen der gespeicherten Tuning-Pläne in andere Formate zu berücksichtigen. Anhand des Prototyps könnte weiterhin auch die Wirkungsweise der in dieser Arbeit vorgestellten Community-Funktionen untersucht werden.

Literaturverzeichnis

[ACD+07] Andrieux, Alain ; Czajkowski, Karl ; Dan, Asit ; Keahey, Kate ; Ludwig, Heiko ; Nakata, Toshiyuki ; Pruyne, Jim ; Rofrano, John ; Tuecke, Steve ; Xu, Ming ; Open Grid Forum (OGF) (Hrsg.): *Web Service Agreement (WS-Agreement) Specification. GFD-R-P.107.* Open Grid Forum (OGF), März 2007

[ACN00] Agrawal, Sanjay ; Chaudhuri, Surajit ; Narasayya, Vivek R.: Automated Selection of Materialized Views and Indexes in SQL Databases. In: *Proceedings of the 26th International Conference on Very Large Data Bases.* San Francisco, CA, USA : Morgan Kaufmann Publishers Inc., 2000 (VLDB '00), 496–505

[AL00] Appelrath, Hans-Jürgen ; Ludewig, Jochen: *Skriptum Informatik: Eine konventionelle Einführung.* 5. Auflage. Teubner Verlag, 2000

[ALSU08] Aho, Alfred V. ; Lam, Monica S. ; Sethi, Ravi ; Ullman, Jeffrey D.: *Compiler: Prinzipien, Techniken und Werkzeuge.* 2. Auflage. Pearson Studium, 2008

[Are08] Arenswald, Stephan: *Einsatz von Workflow-Technologien im autonomen Datenbanktuning*, Institut für Informatik, Friedrich-Schiller-Universität Jena, Studienarbeit, Februar 2008

[Are09] Arenswald, Stephan: *Concept, Design and Implementation of a System for Graphical Modeling and Processing of Hypothesis Trees for Database Tuning Problems*, Institut für Informatik, Friedrich-Schiller-Universität Jena / IBM Deutschland Research & Development GmbH, Diplomarbeit, Januar 2009

[Aut07] Auth, Lars: *Evaluierung autonomer Mechanismen von Oracle Database 10g*, Institut für Informatik, Friedrich-Schiller-Universität Jena, Diplomarbeit, Mai 2007

[AWH92] Aiken, Alexander ; Widom, Jennifer ; Hellerstein, Joseph M.: Behavior of Database Production Rules: Termination, Confluence, and Observable Determinism. In: *Proceedings of the 1992 ACM SIGMOD International Conference on Management of data.* New York, NY, USA : ACM, 1992 (SIGMOD '92), S. 59–68

[BBC+03] Bantz, David F. ; Bisdikian, Chatschik ; Challener, David ; Karidis, John P. ; Mastrianni, Steve J. ; Mohindra, Ajay ; Shea, Dennis G. ; Vanover, Michael: Autonomic Personal Computing. In: *IBM Systems Journal* 42 (2003), Januar, S. 165–176

[BBKZ93] Branding, Holger ; Buchmann, Alejandro P. ; Kudrass, Thomas ; Zimmermann, Jürgen: Rules in an Open System: The REACH Rule System. In: *Rules in Database Systems*, 1993, S. 111–126

[BC06] Bruno, Nicolas ; Chaudhuri, Surajit: To tune or not to tune? A lightweight physical design alerter. In: *Proceedings of the 32nd International Conference on Very Large Data Bases*, VLDB Endowment, 2006 (VLDB '06), 499–510

[BCD95] Barnard, David. T. ; Clarke, Gwen ; Duncan, Nicholas: Tree-to-tree Correction for Document Trees / Department of Computing and Information Science, Queen's University. Kingston Ontario, Canada, Januar 1995. – Forschungsbericht

[BCL96] Brown, Kurt P. ; Carey, Michael J. ; Livny, Miron: Goal-oriented Buffer Management Revisited. In: *SIGMOD Record* 25 (1996), Juni, S. 353–364

[BD94] Bernstein, Philip A. ; Dayal, Umeshwar: An Overview of Repository Technology. In: *Proceedings of the 20th International Conference on Very Large Data Bases*. San Francisco, CA, USA : Morgan Kaufmann Publishers Inc., 1994 (VLDB '94), S. 705–713

[BDR98] Bailey, James ; Dong, Guozhu ; Ramamohanarao, Kotagiri: Decidability and Undecidability Results for the Termination Problem of Active Database Rules. In: *Proceedings of the 17th ACM SIGMOD-SIGACT-SIGART Symposium on Principles of Database Systems*, 1998, S. 264–273

[BDR04] Bailey, James ; Dong, Guozhu ; Ramamohanarao, Kotagiri: On the decidability of the termination problem of active database systems. In: *Theoretical Computer Science* 311 (2004), Januar, 389–437. `http://portal.acm.org/citation.cfm?id=979891.979902`

[Bei04] Beinhauer, Malte ; Scheer, A.-W. (Hrsg.): *Knowledge Communities*. 1. Auflage. Köln : EUL Verlag, 2004

[Ben05] Benoit, Darcy G.: Automatic Diagnosis of Performance Problems in Database Management Systems. In: *Proceedings of the Second International Conference on Automatic Computing*. Washington, DC, USA : IEEE Computer Society, 2005, 326–327

[Bez10] Bezroukov, Nikolai ; Softpanorama (Hrsg.): *Event Correlation Technologies*. Softpanorama, Dezember 2010. `http://www.softpanorama.org/Admin/Event_correlation/`

[BG05] Biazetti, Ana ; Gajda, Kim: Achieving Complex Event Processing with Active Correlation Technology. In: *IBM developerWorks* (2005). `http://www.ibm.com/developerworks/autonomic/library/ac-acact/`

[BHJS00] Bigus, Joe P. ; Hellerstein, Joseph L. ; Jayram, T. S. ; Squillante, Mark S.: AutoTune: A Generic Agent for Automated Performance Tuning. In: *Proceedings of the International Conference on Practical Application of Intelligent Agents and Multi-Agents*, 2000

[BLNP97] Bendix, Lars ; Larsen, Per N. ; Nielsen, Anders I. ; Petersen, Jesper Lai S.: CoEd - A Tool for Cooperative Development of Hierarchical Documents / Aalborg University. Dänemark, 1997. – Forschungsbericht

[BM93] Bertino, Elisa ; Martino, Lorenzo: *Object-Oriented Database Systems: Concepts and Architectures*. Boston, MA, USA : Addison-Wesley Longman Publishing Co., Inc., 1993

[BM05] Bellagio, David ; Milligan, Tom: *Software Configuration Management Strategies and IBM®Rational®Clearcase®: A Practical Introduction*. 2. Auflage. IBM Press, 2005

[BMCL94] Brown, Kurt P. ; Mehta, Manish ; Carey, Michael J. ; Livny, Miron: Towards Automated Performance Tuning for Complex Workloads. In: *Proceedings of the 20th International Conference on Very Large Data Bases*. San Francisco, CA, USA : Morgan Kaufmann Publishers Inc., 1994 (VLDB '94), 72–84

[BMHK99] Beinhauer, Malte ; Markus, Ursula ; Hess, Helge ; Kronz, Andreas: Virtual Community – Kollektives Wissensmanagement im Internet. In: Scheer, A.-W. (Hrsg.): *Electronic Business und Knowledge Management – Neue Dimensionen für den Unternehmenserfolg, Tagungsband zur 20. Saarbrücker Arbeitstagung*. Heidelberg : Physica Verlag, 1999, S. 403–431

[BPT+04] Boxwala, Aziz A. ; Peleg, Mor ; Tu, Samson ; Ogunyemi, Omolola ; Zeng, Qing T. ; Wang, Dongwen ; Patel, Vimla L. ; Greenes, Robert A. ; Shortliffe, Edward H.: GLIF3: A Representation Format for Sharable Computer-Interpretable Clinical Practice Guidelines. In: *Journal of Biomedical Informatics* 37 (2004), Juni, 147–161. `http://portal.acm.org/citation.cfm?id=1022378.1022380`

[Bre94] Brewer, Eric A.: *Portable High-Performance Supercomputing: High-Level Platform-Dependent Optimization*, Dissertationsschrift, September 1994

[Bre95] Brewer, Eric A.: High-Level Optimization via Automated Statistical Modeling. In: *Proceedings of the fifth ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming*. New York, NY, USA : ACM, 1995 (PPOPP '95), S. 80–91

[Bro95] Brown, Kurt P.: *Goal-Oriented Memory Allocation In Database Management Systems*, University of Wisconsin – Madison, Ph.D. Thesis, 1995

[BS97] Balabanović, Marko ; Shoham, Yoav: Fab: Content-based, Collaborative Recommendation. In: *Communications of the ACM* 40 (1997), Nr. 3, S. 66–72

[BW00] Baralis, Elena ; Widom, Jennifer: An Algebraic Approach to Static Analysis of Active Database Rules. In: *ACM Transactions on Database Systems* 25 (2000), September, S. 269–332

[CAM02] Cobena, Gregory ; Abiteboul, Serge ; Marian, Amelie: Detecting Changes in XML Documents. In: *ICDE '02: Proceedings of the 18th International*

Conference on Data Engineering. Washington, DC, USA : IEEE Computer Society, 2002

[Cat94] Cattell, Rick G.: *Object Data Management: Object-Oriented and Extended Relational Database Systems*. Boston, MA, USA : Addison-Wesley Longman Publishing Co., Inc., 1994

[CBMW06] Chen, Whei-Jen ; Baumbach, Ute ; Miskimen, Marcia ; Wicke, Werner ; IBM Corp. (Hrsg.): *DB2 Performance Expert for Multiplatforms V2.2. IBM Redbooks.* IBM Corp., März 2006

[CCW00] Ceri, Stefano ; Cochrane, Roberta ; Widom, Jennifer: Practical Applications of Triggers and Constraints: Success and Lingering Issues (10-Year Award). In: Abbadi, Amr E. (Hrsg.) ; Brodie, Michael L. (Hrsg.) ; Chakravarthy, Sharma (Hrsg.) ; Dayal, Umeshwar (Hrsg.) ; Kamel, Nabil (Hrsg.) ; Schlageter, Gunter (Hrsg.) ; Whang, Kyu-Young (Hrsg.): *Proceedings of 26th International Conference on Very Large Data Bases, September 10-14, 2000, Kairo, Ägypten*, Morgan Kaufmann, 2000 (VLDB 2000), S. 254–262

[CGM97] Chawathe, Sudarshan S. ; Garcia-Molina, Hector: Meaningful Change Detection in Structured Data. In: *SIGMOD '97: Proceedings of the 1997 ACM SIGMOD International Conference on Management of Data*. New York, NY, USA : ACM, 1997, S. 26–37

[CGNZ99] Chaudhuri, Surajit ; Graefe, Eric Christensenand G. ; Narasayya, Vivek R. ; Zwilling, Michael J.: Self-Tuning Technology in Microsoft SQL Server. In: *Data Engineering Journal* 22 (1999), Juni, S. 20–26

[Cha92] Chaffin, Roger: The Concept of a Semantic Relation. In: Lehrer, Adrienne (Hrsg.) ; Kittay, Eva F. (Hrsg.): *Frames, Fields and Contrasts. New Essays in Semantic and Lexical Organisation*. Lawrence Erlbaum Associates Inc., März 1992, S. 253–288

[CK86] Chou, Hong-Tai ; Kim, Won: A Unifying Framework for Version Control in a CAD Environment. In: *Proceedings of the 12th International Conference on Very Large Data Bases*. San Francisco, CA, USA : Morgan Kaufmann Publishers Inc., 1986 (VLDB '86), S. 336–344

[CKAK94] Chakravarthy, Sharma ; Krishnaprasad, V. ; Anwar, Eman ; Kim, Seung-Kyum: Composite Events for Active Databases: Semantics, Contexts and Detection. In: *Proceedings of the 20th International Conference on Very Large Data Bases*. San Francisco, CA, USA : Morgan Kaufmann Publishers Inc., 1994 (VLDB '94), 606–617

[CLP05] Cyran, Michele ; Lane, Paul ; Polk, Jennifer P. ; Oracle (Hrsg.): *Oracle® Database – Concepts (10g Release 2). Part Number B14220-02*. Oracle, Oktober 2005. `http://download.oracle.com/docs/cd/B19306_01/server.102/b14220/memory.htm`

[CMDD62] Corbató, Fernando J. ; Merwin-Daggett, Marjorie ; Daley, Robert C.: An Experimental Time-Sharing System. In: *Proceedings of the Spring Joint*

Computer Conference, 1.-3. Mai, 1962. New York, NY, USA : ACM, 1962 (AIEE-IRE '62 (Spring)), S. 335–344

[CN97] Chaudhuri, Surajit ; Narasayya, Vivek R.: An Efficient Cost-Driven Index Selection Tool for Microsoft SQL Server. In: *Proceedings of the 23rd International Conference on Very Large Data Bases*. San Francisco, CA, USA : Morgan Kaufmann Publishers Inc., 1997 (VLDB '97), 146–155

[CRGMW96] Chawathe, Sudarshan S. ; Rajaraman, Anand ; Garcia-Molina, Hector ; Widom, Jennifer: Change Detection in Hierarchically Structured Information. In: *SIGMOD '96: Proceedings of the 1996 ACM SIGMOD International Conference on Management of Data*. New York, NY, USA : ACM, 1996, S. 493–504

[CSFP09] Collins-Sussman, Ben ; Fitzpatrick, Brian W. ; Pilato, C. M.: *Subversion 1.6 Official Guide: Version Control with Subversion*. Fultus Corporation, 2009

[CT03] Comai, Sara ; Tanca, Letizia: Termination and Confluence by Rule Prioritization. In: *IEEE Transactions on Knowledge and Data Engineering* 15 (2003), März–April, Nr. 2, S. 257 – 270

[CW97] Conradi, Reidar ; Westfechtel, Bernhard: Towards a Uniform Version Model for Software Configuration Management. In: *ICSE '97: Proceedings of the SCM-7 Workshop on System Configuration Management*. London, UK : Springer-Verlag, 1997, S. 1–17

[CW98] Conradi, Reidar ; Westfechtel, Bernhard: Version Models for Software Configuration Management. In: *ACM Computing Surveys* 30 (1998), Nr. 2, S. 232–282

[DBB+88] Dayal, U. ; Blaustein, B. ; Buchmann, A. ; Chakravarthy, U. ; Hsu, M. ; Ledin, R. ; McCarthy, D. ; Rosenthal, A. ; Sarin, S. ; Carey, M. J. ; Livny, M. ; Jauhari, R.: The HiPAC Project: Combining Active Databases and Timing Constraints. In: *SIGMOD Record* 17 (1988), März, S. 51–70

[DD06] Dageville, Benoit ; Dias, Karl: Oracle's Self-Tuning Architecture and Solutions. In: *Bulletin of the Technical Committee on Data Engineering* Bd. 29 IEEE Computer Society, 2006

[DDF+06] Dobson, Simon ; Denazis, Spyros ; Fernández, Antonio ; Gaïti, Dominique ; Gelenbe, Erol ; Massacci, Fabio ; Nixon, Paddy ; Saffre, Fabrice ; Schmidt, Nikita ; Zambonelli, Franco: A Survey of Autonomic Communications. In: *ACM Transactions on Autonomous and Adaptive Systems (TAAS)* 1 (2006), Dezember, S. 223–259

[DDLS01] Damianou, Nicodemos ; Dulay, Naranker ; Lupu, Emil ; Sloman, Morris: The Ponder Policy Specification Language. In: *Proceedings of the International Workshop on Policies for Distributed Systems and Networks*. London, UK : Springer-Verlag, 2001 (POLICY '01), 18–38

[DG00] Dittrich, Klaus R. ; Gatziu, Stella: *Aktive Datenbanksysteme – Konzepte und Mechanismen*. 2. Auflage. Heidelberg : dpunkt.verlag, 2000

[DH72] Dijkstra, Edsger W. ; Hoare, C. A. R. ; Dahl, O. J. (Hrsg.) ; Dijkstra, E. W. (Hrsg.) ; Hoare, C. A. R. (Hrsg.): *Structured Programming*. London, UK : Academic Press Ltd., 1972

[DHP+05] Diao, Yixin ; Hellerstein, Joseph L. ; Parekh, Sujay ; Griffith, Rean ; Kaiser, Gail ; Phung, Dan: Self-Managing Systems: A Control Theory Foundation. In: *Proceedings of the 12th IEEE International Conference and Workshops on Engineering of Computer-Based Systems*. Washington, DC, USA : IEEE Computer Society, 2005, 441–448

[DMT07] DMTF (Hrsg.): *Common Information Model Simplified Policy Language (CIM-SPL) – Version 1.0.0a*. DMTF, Januar 2007

[DRS+05] Dias, Karl ; Ramacher, Mark ; Shaft, Uri ; Venkataramani, Venkateshwaran ; Wood, Graham: Automatic Performance Diagnosis and Tuning in Oracle. In: *Proceedings of Conference on Innovative Data Systems Research (CIDR)*, 2005, S. 84–94

[Eat07a] Eaton, Chris: *DB2PD – Great Diagnostic Tool*, Januar 2007. `http://blogs.ittoolbox.com/database/technology/archives/db2pd-great-diagnostic-tool-14089`

[Eat07b] Eaton, Chris: *DB2PD to Monitor Buffer Pools (Part 1)*, Januar 2007. `http://blogs.ittoolbox.com/database/technology/archives/db2pd-to-monitor-bufferpools-14219`

[Eat07c] Eaton, Chris: *DB2PD to Monitor Buffer Pools (Part 2)*, April 2007. `http://blogs.ittoolbox.com/database/technology/archives/db2pd-to-monitor-bufferpools-part-2-15464`

[Eat07d] Eaton, Chris: *DB2PD to Monitor Locks*, März 2007. `http://blogs.ittoolbox.com/database/technology/archives/db2pd-to-monitor-locks-15228`

[Eat07e] Eaton, Chris: *DB2PD to Monitor Log Utilization*, Februar 2007. `http://blogs.ittoolbox.com/database/technology/archives/db2pd-to-monitor-log-utilization-14318`

[Eat07f] Eaton, Chris: *DB2PD to Monitor Table Access*, März 2007. `http://blogs.ittoolbox.com/database/technology/archives/db2pd-to-monitor-table-access-14967`

[Eck08] Eckert, Michael: *Complex Event Processing with XChangeEQ: Language Design, Formal Semantics and Incremental Evaluation for Querying Events*, Fakultät für Mathematik, Informatik und Statistik, Ludwig-Maximilians-Universität München, Ph.D. Thesis, Oktober 2008

[Ehl07] Ehlert, Martin: *Beschreibung und Verwaltung von Tuning-Strategien für ein autonomes Datenbanksystem*, Institut für Informatik, Friedrich-Schiller-Universität Jena, Studienarbeit, Januar 2007

[Eln04] Elnaffar, Said S.: *Towards Workload-aware DBMSs: Identifying Workload Type and Predicting its Change*. Kingston, Ontario, Canada, Queen's University, Ph.D. Thesis, 2004

[EN10] Etzion, Opher ; Niblett, Peter: *Event Processing in Action*. Manning Publications Co., 2010

[EPBM03] Elnaffar, Said ; Powley, Wendy ; Benoit, Darcy ; Martin, Pat: Today's DBMSs: How autonomic are they? In: *Proceedings of the 14th International Workshop on Database and Expert Systems Applications*. Washington, DC, USA : IEEE Computer Society, 2003 (DEXA '03), 651–655

[EPT10] *Event Processing Technical Society (EPTS)*. `http://www.ep-ts.com`. Version: 2010, Abruf: November 2010

[Eri99] Ericson, Clifton A.: Fault Tree Analysis – A History. In: *Proceedings of 17th International System Safety Conference*, 1999

[Exn07] Exner, Tobias: *Techniken zur Approximation und Optimierung von Performancefunktionen in Datenbankmanagementsystemen*, Institut für Informatik, Friedrich-Schiller-Universität Jena, Diplomarbeit, April 2007

[FB04] Fogel, Karl ; Bar, Moshe: *Open Source-Projekte mit CVS*. 3. Auflage. Bonn : mitp, 2004

[FN99] Feitelson, Dror G. ; Naaman, Michael: Self-Tuning Systems. In: *IEEE Software* 16 (1999), März, S. 52–60

[Gan06] Ganskow, Kai: *Autonomic Database Performance Tuning: Formale Systemmodellierung am Beispiel von IBM DB2*, Institut für Informatik, Friedrich-Schiller-Universität Jena, Diplomarbeit, September 2006

[Gar04] Garshol, Lars M.: Metadata? Thesauri? Taxonomies? Topic maps! Making sense of it all. In: *Journal of Information Science* 30 (2004), Nr. 4, S. 378–391

[Göb08] Göbel, Andreas: *Konzeptuelle und prototypische Evaluierung der Möglichkeiten des Intelligent Miner im Autonomen Datenbank-Tuning*, Institut für Informatik, Friedrich-Schiller-Universität Jena, Studienarbeit, 2008

[GBS+05] Gucer, Vasfi ; Beganskas, Charles ; Salustri, Fabrizio ; Saulman, Jerry ; Toure, Mamadou ; Willis, John ; Yamazi, Brett ; Scandolara, André L. ; Shah, Ghufran ; Wallace, Scott ; Wouters, Dirk ; IBM Corp. (Hrsg.): *Getting Started with IBM Tivoli Monitoring 6.1 on Distributed Environments. IBM Redbooks*. IBM Corp., Dezember 2005

[GC03] Ganek, Alan G. ; Corbi, T. A.: The Dawning of the Autonomic Computing Era. In: *IBM Systems Journal* 42 (2003), Januar, S. 5–18

[GD93] Gatziu, Stella ; Dittrich, Klaus R.: Events in an Active Object-Oriented Database System. University of Zurich, 1993. – Forschungsbericht

[Ger07] Gerlicher, Ansgar: Computer-Supported Cooperative Work (CSCW) – kollaborative Systeme und Anwendungen. In: Schmitz, Roland (Hrsg.): *Kompendium Medieninformatik – Medienpraxis*. Springer-Verlag, 2007, S. 143–195

[GG99] Gehring, Hermann ; Gadatsch, Andreas: *Eine Rahmenarchitektur für Workflow-Management-Systeme. Fachbereichsbericht Nr. 275*. Hagen : Fernuniversität Hagen, 1999 (Diskussionsbeiträge / Fachbereich Wirtschaftswissenschaft 275). http://gso.gbv.de/DB=2.1/CMD?ACT=SRCHA&SRT=YOP&IKT=1016&TRM=ppn+304319783&sourceid=fbw_bibsonomy

[GSPW99] Goel, Ashvin ; Steere, David ; Pu, Calton ; Walpole, Jonathan: Adaptive Resource Management via Modular Feedback Control / Oregon Graduate Institute. 1999. – Technischer Bericht

[Haa96] Haake, Anja: *Versionenunterstützung für strukturierte Hyperdokumente im elektronischen Publizieren*. Darmstadt, Technische Universität Darmstadt, Dissertationsschrift, November 1996

[HAM07] Hoisl, Bernhard ; Aigner, Wolfgang ; Miksch, Silvia: Soziale Belohnung in Wiki Systemen. In: *Wikis im Social Web Wikiposium 2005/06*, OCG Austrian Computer Society, 2007, S. 60–72

[Hay05] Hayes, Scott: In 8 Years of UDB Performance Solutions: Life Lessons from the Field. In: *IDUG 2005 – Europe*, 2005

[HCP+93] Hripcsak, George ; Clayton, Paul D. ; Pryor, T. Allan ; Haug, Peter ; Wigertz, Ove B. ; Van Der Lei, Johan: The Arden Syntax for Medical Logic Modules. In: *Proceedings of the Annual Symposium on Computer Application in Medical Care* 10 (1993), Nr. 4, 215–224. http://www.ncbi.nlm.nih.gov/pubmed/8270835

[Hei04] Heiss, Silke F.: Personale und interpersonale Faktoren für die Wissenskommunikation in Communities of Practice. In: Reinhardt, Rüdiger (Hrsg.) ; Eppler, Martin. J. (Hrsg.): *Wissenskommunikation in Organisationen – Methoden, Instrumente, Theorien*. Berlin : Springer Verlag, 2004

[Hen07] Hennig, Christoph: *Klassifikation und Formalisierung von DB-Performance-Problemen, deren Diagnose und Auflösung anhand von Best-Practice-Methoden am Beispiel von IBM DB2 UDB*, Institut für Informatik, Friedrich-Schiller-Universität Jena, Studienarbeit, August 2007

[Hew07] Hewlett-Packard (Hrsg.): *HP Adaptive Infrastructure – Das Rechenzentrum der Zukunft. Hintergrundinformation*. Hewlett-Packard, Februar 2007. www.hp.com/go/adaptiveinfrastructure

[HK07] Höhfeld, Stefanie ; Kwiatkowski, Melanie: Empfehlungssysteme aus informationswissenschaftlicher Sicht – State of the Art. In: *IWP-Information Wissenschaft & Praxis* 58 (2007), Nr. 5, 265-276. http://wwwalt.phil-fak.uni-duesseldorf.de/infowiss/admin/public_dateien/files/58/1189509550empfehlung.pdf

[HKRS08] Hitzler, Pascal ; Krötzsch, Markus ; Rudolph, Sebastian ; Sure, York: *Semantic Web – Grundlagen*. Springer Verlag, 2008

[HLD+05] Hepp, Martin ; Leymann, Frank ; Domingue, John ; Wahler, Alexander ; Fensel, Dieter: Semantic Business Process Management: A Vision Towards Using Semantic Web Services for Business Process Management. In: *Proceedings of the IEEE ICEBE 2005*, IEEE Computer Society, 2005, S. 535–540

[HN01] Hansen, Hans R. ; Neumann, Gustaf: *Wirtschaftsinformatik 1. Grundlagen betrieblicher Informationsverarbeitung*. 8. Auflage. Lucius & Lucius Verlagsgesellschaft, 2001

[Hof09] Hofmeister, Alexander: *Konzeptionierung und prototypische Entwicklung eines Systems zur Community-orientierten Verwaltung von Tuning-Plänen*, Institut für Informatik, Friedrich-Schiller-Universität Jena, Diplomarbeit, Februar 2009

[Hoi07] Hoisl, Bernhard: *Motivate Online Community Contributions Using Social Rewarding Techniques – A Focus on Wiki Systems*, Softwaretechnik und interaktive systeme. Technische Universität Wien, Magisterarbeit, September 2007. `http://ieg.ifs.tuwien.ac.at/projects/SocialRewarding/MasterThesisSocialRewarding.pdf`

[HR01] Härder, Theo ; Rahm, Erhard: *Datenbanksysteme: Konzepte und Techniken der Implementierung*. 2. Auflage. Springer Verlag, 2001

[IBM01] IBM Corp. (Hrsg.): *Autonomic Computing: IBM's Perspective on the State of Information Technology*. IBM Corp., 2001. `http://www.research.ibm.com/autonomic/manifesto/autonomic_computing.pdf`

[IBM03a] IBM Corp. (Hrsg.): *An Architectural Blueprint for Autonomic Computing*. 1. Auflage. IBM Corp., April 2003

[IBM03b] IBM Corp. (Hrsg.): *IBM Tivoli Enterprise Console 3.9. Rule Developer's Guide. SC32-1234-00*. IBM Corp., August 2003. `http://publib.boulder.ibm.com/tividd/td/tec/SC32-1234-00/en_US/PDF/ecodmst.pdf`

[IBM04a] IBM Corp. (Hrsg.): *Autonomic Computing Toolkit: Developer's Guide. SC30-4083-02*. IBM Corp., August 2004

[IBM04b] IBM Corp. (Hrsg.): *IBM DB2 Universal Database, Version 8 – Application Development Guide: Building and Running Applications. SC09-4825-01*. IBM Corp., 2004

[IBM06a] IBM Corp. (Hrsg.): *An Architectural Blueprint for Autonomic Computing*. 4. Auflage. IBM Corp., 2006. `http://www-03.ibm.com/autonomic/pdfs/AC_Blueprint_White_Paper_4th.pdf`

[IBM06b] IBM Corp. (Hrsg.): *DB2 Connect Version 9. User's Guide. SC10-4229-00*. IBM Corp., 2006

[IBM06c] IBM Corp. (Hrsg.): *DB2 Data Warehouse Edition Version 9.1.1 – Intelligent Miner Modeling: Administration and Programming Guide.* IBM Corp., 2006

[IBM06d] IBM Corp. (Hrsg.): *DB2 Version 9 for Linux, UNIX, and Windows – Administration Guide: Planning. SC10-4223-00.* IBM Corp., 2006

[IBM06e] IBM Corp. (Hrsg.): *DB2 Version 9 for Linux, UNIX, and Windows – Command Reference. SC10-4226-00.* IBM Corp., 2006

[IBM06f] IBM Corp. (Hrsg.): *DB2 Version 9 for Linux, UNIX, and Windows – Performance Guide. SC10-4222-00.* IBM Corp., 2006

[IBM06g] IBM Corp. (Hrsg.): *DB2 Version 9 for Linux, UNIX, and Windows – SQL Reference Volume 1. SC10-4249-00.* IBM Corp., 2006

[IBM06h] IBM Corp. (Hrsg.): *DB2 Version 9 for Linux, UNIX, and Windows – System Monitor Guide and Reference. SC10-4251-00.* IBM Corp., 2006

[IBM06i] IBM Corp. (Hrsg.): *DB2 Version 9 for Linux, UNIX, and Windows – Troubleshooting Guide. GC10-4240-00.* IBM Corp., 2006

[IBM07] IBM Corp. (Hrsg.): *IBM Tivoli Monitoring for Databases: DB2 Agent User's Guide Version 6.2.0. SC12-3588-01.* IBM Corp., 2007

[IBM08a] IBM Corp. (Hrsg.): *DB2 Version 9.5 for Linux, UNIX, and Windows – Administrative Routines and Views. SC23-5843-01.* IBM Corp., 2008

[IBM08b] IBM Corp. (Hrsg.): *IBM Tivoli Monitoring User's Guide. SC32-9409-01.* IBM Corp., 2008

[IBM08c] IBM Corp. (Hrsg.): *IBM Tivoli Monitoring. Version 6.2 Fix Pack 1 (Revised May 2008) User's Guide. SC32-9409-02.* IBM Corp., 2008

[IBM10] IBM Corp. (Hrsg.): *IBM Tivoli Directory Integrator 7.1 Reference Guide. SC27-2707-00.* IBM Corp., 2010. `http://publib.boulder.ibm.com/infocenter/tivihelp/v2r1/topic/com.ibm.IBMDI.doc_7.1/referenceguide.htm`

[IBM11] IBM Corp. (Hrsg.): *IBM WebSphere sMash Developer's Guide. Version 1.1.31566.* IBM Corp., Juni 2011. `http://www.projectzero.org/sMash/1.1.x/docs/zero.devguide.doc/index.html`

[Jac95] Jacobson, Van: Congestion Avoidance and Control. In: *ACM SIGCOMM Computer Communication Review* 25 (1995), Januar, S. 157–187

[Jäg08] Jäger, Anna: *Evaluierung von Konzepten, Problemen und Lösungsansätzen aktiver Datenbanksysteme und Untersuchung der Übertragbarkeit auf autonome Datenbanksysteme*, Institut für Informatik, Friedrich-Schiller-Universität Jena, Studienarbeit, Februar 2008

[Kar07] Karabel, Tunca: *Entwurf und Implementierung eines web-basierten Assistenten zur Erstellung, Ausführung und Verwaltung von Datenbank-Tuning-Strategien*, Hochschule für Technik, Stuttgart / IBM Deutschland Research & Development GmbH, Böblingen, Bachelorarbeit, Dezember 2007

[KC03] Kephart, Jeffrey O. ; Chess, David M.: The Vision of Autonomic Computing. In: *Computer* 36 (2003), Januar, S. 41–50

[Kes95] Keshav, Srinivasan: A Control-Theoretic Approach to Flow Control. In: *ACM SIGCOMM Computer Communication Review* 25 (1995), Januar, S. 188–201

[KN09] Krumke, Sven O. ; Noltemeier, Hartmut: *Graphentheoretische Konzepte und Algorithmen*. Vieweg+Teubner Verlag / GWV Fachverlage GmbH, Wiesbaden, 2009

[Krü09] Krüger, Jan: *Verwaltung und Versionierung von WebSphere sMash basierten Tuningplänen*, Institut für Informatik, Friedrich-Schiller-Universität Jena, Studienarbeit, Dezember 2009

[KRW+07] Küspert, Klaus ; Rabinovitch, Gennadi ; Wiese, David ; Stumm, Rüdiger ; Reichert, Michael: *Autonomic Database Performance Tuning*. Vortrag anlässlich der Verleihung des IBM Shared University Research Grant 2007 an den Lehrstuhl für Datenbanken und Informationssysteme, Friedrich-Schiller-Universität Jena, November 2007

[KW04] Kephart, Jeffrey O. ; Walsh, William E.: An Artificial Intelligence Perspective on Autonomic Computing Policies. In: *Proceedings of the Fifth IEEE International Workshop on Policies for Distributed Systems and Networks*. Washington, DC, USA : IEEE Computer Society, 2004, 3–12

[KWR05] Küspert, Klaus ; Wiese, David ; Rabinovitch, Gennadi: *Autonome Datenbank-Administration*. Poster zum Tag der Forschung der Friedrich-Schiller-Universität Jena, Oktober 2005

[Leb95] Leblang, David B.: The CM Challenge: Configuration Management that works. In: *Configuration Management*. New York, NY, USA : John Wiley & Sons, Inc., 1995, S. 1–37

[Len00] Lenk, Torsten: *Flexible Versionierung semistrukturierter Dokumente im Kontext von Autonomie und Kooperation*, Arbeitsbereich Softwaresysteme, Technische Universität Hamburg-Harburg, Diplomarbeit, Februar 2000

[Lin01] Lindholm, Tancred: *A 3-way Merging Algorithm for Synchronizing Ordered Trees – The 3DM Merging and Differencing Tool for XML*. Helsinki, Helsinki University of Technology, Department of Computer Science, Laboratory of Information Processing Science, Master's thesis, 2001

[LKD+03] Ludwig, Heiko ; Keller, Alexander ; Dan, Asit ; King, Richard P. ; Franck, Richard ; IBM Corp. (Hrsg.): *Web Service Level Agreement (WSLA) Language Specification – Version 1.0*. IBM Corp., Januar 2003

[LL95] Lang, Stefan ; Lockemann, Peter C.: *Datenbankeinsatz*. Springer Verlag, 1995

[LR97] Leymann, Frank ; Roller, Dieter: Workflow-based Applications. In: *IBM Systems Journal* 36 (1997), Januar, S. 102–123

[LSY03] Linden, Greg ; Smith, Brent ; York, Jeremy: Amazon.com Recommendations: Item-to-Item Collaborative Filtering. In: *IEEE Internet Computing* 7 (2003), S. 76–80

[Luc07] Luckham, David C.: *A Short History of Complex Event Processing. Part 1: Beginnings*, 2007. `http://complexevents.com/?p=321`

[Lv00] Lakhani, Karim ; von Hippel, Eric: How Open Source software works: „Free“ user-to-user assistance / MIT Sloan School of Management. 2000. – Working Paper

[ME07] Mihaeli, Joris ; Etzion, Opher: Detecting Event Processing Patterns in Event Databases. In: *Proceedings of the Second International Workshop on Event-driven Architecture, Processing and Systems (EDA-PS'07)* (2007)

[MESW01] Moore, B. ; Ellesson, E. ; Strassner, J. ; Westerinen, A.: *Policy Core Information Model – Version 1 Specification*, 2001

[MFH02] Mockus, Audris ; Fielding, Roy T. ; Herbsleb, James D.: Two Case Studies of Open Source Software Development: Apache andMozilla. In: *ACM Transactions on Software Engineering and Methodology* 11 (2002), Nr. 3, S. 309–346

[MH03] Millsap, Cary ; Holt, Jeff: *Optimizing Oracle Performance*. Sebastopol, CA, USA : O'Reilly & Associates, Inc., 2003

[Mic11a] Microsoft (Hrsg.): *Microsoft Research: AutoAdmin Project Summary*. Microsoft, 2011. `http://research.microsoft.com/en-us/projects/autoadmin/`, Abruf: September 2011

[Mic11b] Microsoft (Hrsg.): *Microsoft SQL Server 2008 R2: Online Product Documentation*. Microsoft, 2011. `http://msdn.microsoft.com/en-us/library/ms130214.aspx`, Abruf: September 2011

[Mic11c] Micrososft (Hrsg.): *Microsoft Dynamic Systems Initiative: Der Weg zu sich selbst verwaltenden dynamischen Systemen*. Micrososft, 2011. `http://www.microsoft.com/germany/server/dsi/default.mspx`, Abruf: September 2011

[Mil05] Miller, Brent: The Autonomic Computing Edge: The Role of Knowledge in Autonomic Systems. In: *IBM developerWorks* (2005), September. `http://www.ibm.com/developerworks/autonomic/library/ac-edge6`

[MMN02] Melville, Prem ; Mooney, Raymod J. ; Nagarajan, Ramadass: Content-Boosted Collaborative Filtering for Improved Recommendations. In: *Eighteenth National Conference on Artificial Intelligence*. Menlo Park, CA, USA : American Association for Artificial Intelligence, 2002, S. 187–192

[MP90] Massalin, Henry ; Pu, Calton: Fine-Grain Adaptive Scheduling using Feedback. In: *Computing Systems* 3 (1990), S. 139–173

[MS93] Moffett, Jonathan D. ; Sloman, Morris S.: Policy Hierarchies for Distributed Systems Management. In: *IEEE Journal on Selected Areas in Communications* 11 (1993), S. 1404–1414

[MSZ01] McIlraith, Sheila A. ; Son, Tran C. ; Zeng, Honglei: Semantic Web Services. In: *IEEE Intelligent Systems* 16 (2001), Nr. 2, S. 46–53

[NCK+92] Neuwirth, Christine M. ; Chandhok, Ravinder ; Kaufer, David S. ; Erion, Paul ; Morris, James ; Miller, Dale: Flexible Diff-ing in a Collaborative Writing System. In: *CSCW '92: Proceedings of the 1992 ACM Conference on Computer-Supported Cooperative Work*. New York, NY, USA : ACM, 1992, S. 147–154

[NFS00] Narayanan, Dushyanth ; Flinn, Jason Nelson ; Satyanarayanan, Mahadev: Using History to Improve Mobile Application Adaptation. In: *Proceedings of the Third IEEE Workshop on Mobile Computing Systems and Applications (WMCSA'00)*. Washington, DC, USA : IEEE Computer Society, 2000, 31–40

[NSN+97] Noble, Brian D. ; Satyanarayanan, Mahadev ; Narayanan, Dushyanth ; Tilton, James E. ; Flinn, Jason ; Walker, Kevin R.: Agile Application-Aware Adaptation for Mobility. In: *Proceedings of the sixteenth ACM Symposium on Operating Systems Principles*. New York, NY, USA : ACM, 1997 (SOSP '97), S. 276–287

[OAS05] OASIS (Hrsg.): *eXtensible Access Control Markup Language (XACML) – Version 2.0*. OASIS, Februar 2005

[OAS07a] OASIS (Hrsg.): *Web Services Business Process Execution Language Version 2.0*. OASIS, April 2007

[OAS07b] OASIS (Hrsg.): *Web Services Business Process Execution Language Version 2.0. Primer*. OASIS, Mai 2007

[OGT04] Orhanovic, Jens ; Grodtke, Ivo ; Tiefenbacher, Michael: *DB2 Administration – Einführung, Handbuch und Referenz, Version 8*. 1. Auflage. Addison-Wesley Verlag, 2004

[Ohs04] Ohst, Dirk: *Versionierungskonzepte mit Unterstützung für Differenz- und Mischwerkzeuge*, Fachbereich Elektrotechnik und Informatik. Universität Siegen, Dissertationsschrift, 2004

[OMG11] OMG (Hrsg.): *Business Process Model and Notation (BPMN) – Version 2.0*. OMG, Januar 2011. `http://www.omg.org/spec/BPMN/2.0`

[OMGM+98] Ohno-Machado, Lucila ; Gennari, John H. ; Murphy, Shawn N. ; Jain, Nilesh L. ; Tu, Samson W. ; Oliver, Diane E. ; Pattison-Gordon, Edward ; Greenes, Robert A. ; Shortliffe, Edward H. ; Barnett, G. Octo: The GuideLine Interchange Format: A Model for Representing Guidelines. In: *Journal of the American Medical Informatics Association* 5 (1998), Juli, Nr. 4, 357–372. `http://www.jamia.org/cgi/content/abstract/5/4/357`

[Ora01] Oracle Corporation (Hrsg.): *Oracle Enterprise Manager. Database Tuning with the Oracle Tuning Pack Release 9.0.1. Part No. A86647-01*. Oracle Corporation, Juni 2001. `http://download.oracle.com/docs/cd/B10501_01/em.920/a86647.pdf`

[Ora07] Oracle (Hrsg.): *Oracle Database Rules Manager and Expression Filter Developer's Guide 11g Release 1 (11.1). B31088-0.* Oracle, Juli 2007. http://download.oracle.com/docs/cd/B28359_01/appdev.111/b31088/toc.htm

[Ora08] Oracle Corporation (Hrsg.): *Oracle® Database Performance Tuning Guide 10g Release 2 (10.2). B14211-03.* Oracle Corporation, März 2008. http://download.oracle.com/docs/cd/B19306_01/server.102/b14211.pdf

[Pat99] Paton, Norman W. (Hrsg.): *Active Rules in Database Systems.* Springer, New York, 1999

[PBO+00] Peleg, Mor ; Boxwala, Aziz A. ; Ogunyemi, Omolola ; Zeng, Qing ; Tu, Samson ; Lacson, Ronilda ; Bernstam, Elmer ; Ash, Nachman ; Mork, Peter ; Ohno-Machado, Lucila ; Shortliffe, Edward H. ; Greenes, Robert A.: GLIF3: The Evolution of a Guideline Representation Format. In: *Proceedings of the AMIA Symposium*, 2000, S. 645–649

[PD99] Paton, Norman W. ; Díaz, Oscar: Active Database Systems. In: *ACM Computing Surveys* 31 (1999), März, S. 63–103

[PM06] Powley, Wendy ; Martin, Pat: A Reflective Database-Oriented Framework for Autonomic Managers. In: *Proceedings of the International Conference on Autonomic and Autonomous Systems.* Washington, DC, USA : IEEE Computer Society, 2006 (ICAS '06)

[PTB+03] Peleg, Mor ; Tu, Samson ; Bury, Jonathan ; Ciccarese, Paolo ; Fox, John ; Greenes, Robert A. ; Hall, Richard ; Johnson, Peter D. ; Jones, Neill ; Kumar, Anand ; Miksch, Silvia ; Quaglini, Silvana ; Seyfang, Andreas ; Shortliffe, Edward H. ; Stefanelli, Mario: Comparing Computer-Interpretable Guideline Models: A Case-Study Approach. In: *Journal of American Medical Informatics Association* 10 (2003), S. 52–68

[Rab06] Rabinovitch, Gennadi: Technologien und Konzepte zur autonomen Verwaltung von IT-Systemen. In: *Tagungsband zum 18. Workshop über Grundlagen von Datenbanken, Institut für Informatik, Martin-Luther-Universität Halle-Wittenberg.* Wittenberg, Juni 2006, S. 120–124

[Rab09] Rabinovitch, Gennadi: Policy-Based Coordination of Best-Practice Oriented Autonomic Database Tuning. In: *Proceedings of the 2009 Computation World: Future Computing, Service Computation, Cognitive, Adaptive, Content, Patterns.* Washington, DC, USA : IEEE Computer Society, 2009, S. 55–60

[RAWR08] Reichert, Michael ; Arenswald, Stephan ; Wiese, David ; Rabinovitch, Gennadi: *Workload Aware Exception Detection.* Patent Proposal, September 2008

[RCR94] Roddick, John F. ; Craske, Noel G. ; Richards, Thomas J.: A Taxonomy for Schema Versioning Based on the Relational and Entity Relationship Models. In: *ER '93: Proceedings of the 12th International Conference on*

the Entity-Relationship Approach. London, UK : Springer-Verlag, 1994, S. 137–148

[Ree05] Rees, Steve: *Advanced Performance Diagnostics in DB2 UDB for LUW v8.2.x*. Vortrag. IDUG 2005 – Europe, Oktober 2005

[Ree07] Rees, Steve: *Zen and the Art of Database Performance. DB2 Chat with the Lab. IBM Software Group, DB2 Information Management Software*, Februar 2007. `ftp://ftp.software.ibm.com/software/data/db2/9/labchats/20070221-slides.pdf`

[Rei08] Reichenbach, Michael: *Entwicklung eines Modells zur Definition von Tuning-Richtlinien zur Steuerung des autonomen Datenbank-Tunings*, Institut für Informatik, Friedrich-Schiller-Universität Jena, Diplomarbeit, Juli 2008

[RMHA09] Raza, Basit ; Mateen, Abdul ; Hussain, Tauqeer ; Awais, Mian M.: Autonomic Success in Database Management Systems. In: Miao, Huaikou (Hrsg.) ; Hu, Gongzhu (Hrsg.): *8th IEEE/ACIS International Conference on Computer and Information Science, IEEE/ACIS ICIS 2009, 1.-3. Juni, 2009, Shanghai, China*, IEEE Computer Society, 2009, S. 439–444

[Roe08] Roeder, Katharina: *Konzipierung und prototypische Umsetzung einer Workload-Komponente zur gezielten Erzeugung von DB2-spezifischen Sperrproblemen im Umfeld des Autonomen Datenbank-Tunings*, Institut für Informatik, Friedrich-Schiller-Universität Jena, Studienarbeit, August 2008

[RP81] Reiner, David ; Pinkerton, Tad: A Method for Adaptive Performance Improvement of Operating Systems. In: *ACM SIGMETRICS Conference on Measurement and Modeling of Computer Systems* 10 (1981), September, S. 2–10

[RW07] Rabinovitch, Gennadi ; Wiese, David: Non-linear Optimization of Performance Functions for Autonomic Database Performance Tuning. In: *Proceedings of the Third International Conference on Autonomic and Autonomous Systems*. Washington, DC, USA : IEEE Computer Society, 2007

[RWRA08] Rabinovitch, Gennadi ; Wiese, David ; Reichert, Michael ; Arenswald, Stephan: Datenbank-Tuning mit dem Autonomic Tuning Expert. In: *Datenbank-Spektrum* Heft 27 (2008), Dezember, S. 18–26

[RWRA10] Rabinovitch, Gennadi ; Wiese, David ; Reichert, Michael ; Arenswald, Stephan: *Workflow- und Workload-orientiertes Datenbank-Tuning*. Forschungsbericht. Lehrstuhl für Datenbanken und Informationssysteme. Friedrich-Schiller-Universität Jena, November 2010

[Sch03] Schumacher, Robin: Monitoring Databases the Right Way with Embarcadero Performance Analyst / Embarcadero Technologies, Inc. 2003. – Technischer Bericht

[Sed08] Sedlmayr, Martin: *Proaktive Assistenz zur kontextabhängigen und zielorientierten Unterstützung bei der Indikationsstellung und Anwendung von Behandlungsmaßnahmen in der Intensivmedizin*, Fakultät für Mathematik,

Informatik und Naturwissenschaften. RWTH Aachen, Dissertationsschrift, 2008

[Sel77] Selkow, Stanley M.: The Tree-to-Tree Editing Problem. In: *Information Processing Letters* 6 (1977), Nr. 6, S. 184–186

[SGAL+06] Storm, Adam J. ; Garcia-Arellano, Christian ; Lightstone, Sam S. ; Diao, Yixin ; Surendra, Maheswaran: Adaptive Self-Tuning Memory in DB2. In: *Proceedings of the 32nd International Conference on Very Large Data Bases*, VLDB Endowment, 2006 (VLDB '06), 1081–1092

[SGG+99] Steere, David C. ; Goel, Ashvin ; Gruenberg, Joshua ; McNamee, Dylan ; Pu, Calton ; Walpole, Jonathan: A Feedback-Driven Proportion Allocator for Real-Rate Scheduling. In: *Proceedings of the Third Symposium on Operating systems Design and Implementation*. Berkeley, CA, USA : USENIX Association, 1999 (OSDI '99), 145–158

[SGS03] Sattler, Kai-Uwe ; Geist, Ingolf ; Schallehn, Eike: QUIET: Continuous Query-driven Index Tuning. In: *Proceedings of the 29th International Conference on Very Large Data Bases* Bd. 29, VLDB Endowment, 2003 (VLDB '03), 1129–1132

[SHS05] Saake, Gunter ; Heuer, Andreas ; Sattler, Kai-Uwe: *Datenbanken: Implementierungstechniken*. 2. Auflage. mitp-Verlag, 2005

[SKR01] Schafer, J. Ben ; Konstan, Joseph A. ; Riedl, John: E-Commerce Recommendation Applications. In: *Data Mining and Knowledge Discovery* 5 (2001), Nr. 1-2, S. 115–153

[Sne03] Sneed, Bob: Performance Forensics – SMI Performance and Availability Engineering (PAE) / Sun Microsystems, Inc. 2003. – Technischer Bericht

[Som01] Sommerville, Ian: *Software Engineering*. 6. Auflage. Boston, MA, USA : Addison-Wesley Longman Publishing Co., Inc., 2001

[SR01] Stotz, Jan P. ; Röder, Patrick: *RCS / CVS – Werkzeuge zur Versionsverwaltung*. `http://www.patrick-roeder.de/vortraege/RCS-CVS-Ausarbeitung.pdf`. Version: 2001

[SRR+07] Sedlmayr, Martin ; Rose, Thomas ; Röhrig, Rainer ; Meister, Markus ; Michel-Backofen, Achim: Formalisierung und Automatisierung von SOPs in der Intensivmedizin. In: Oberweis, Andreas (Hrsg.) ; Weinhardt, Christof (Hrsg.) ; Gimpel, Henner (Hrsg.) ; Koschmider, Agnes (Hrsg.) ; Pankratius, Victor (Hrsg.) ; Schnizler, Björn (Hrsg.): *Wirtschaftsinformatik (1)*, Universitaetsverlag Karlsruhe, 2007, 953-

[SRRM06] Sedlmayr, Martin ; Rose, Thomas ; Röhrig, Rainer ; Meister, Markus: A Workflow Approach towards GLIF Execution. In: *Proceedings on Workshop AI Techniques in Healthcare: Evidence-based Guidelines and Protocols, European Conference on Artificial Intelligence (ECAI)*. Trient, 2006

[SS97] Seltzer, Margo ; Small, C.: Self-Monitoring and Self-Adapting Operating Systems. In: *Proceedings of the 6th Workshop on Hot Topics in Operating Systems (HotOS-VI).* Washington, DC, USA : IEEE Computer Society, 1997, 124–129

[SS07] Schill, Alexander ; Springer, Thomas: *Verteilte Systeme: Grundlagen und Basistechnologien.* Springer Verlag, 2007

[SSH10] Saake, Gunter ; Sattler, Kai-Uwe ; Heuer, Andreas: *Datenbanken – Konzepte und Sprachen.* 4. Auflage. Heidelberg : mitp, 2010

[Sto07] Stock, Wolfgang G.: Folksonomies and Science Communication: A Mash-up of Professional Science Databases and Web 2.0 Services. In: *Information Services and Use* 27 (2007), Nr. 3, S. 97–103

[Sul03] Sullivan, David Gerard: *Using Probabilistic Reasoning to Automate Software Tuning.* Cambridge, MA, USA, Harvard University, Ph.D. Thesis, 2003

[Tai79] Tai, Kuo-Chung: The Tree-to-Tree Correction Problem. In: *Journal of the ACM* 26 (1979), Nr. 3, S. 422–433

[TH01] Terveen, Loren ; Hill, Will: Beyond Recommender Systems: Helping People Help Each Other. In: Carroll, J. (Hrsg.): *HCI in the New Millennium*, Addison-Wesley, 2001, S. 487–509

[Thu04] Thums, Andreas: *Formale Fehlerbaumanalyse*, Fakultatät für Angewandte Informatik. Universität Augsburg, Dissertationsschrift, Juni 2004

[Tic85] Tichy, Walter F.: RCS – A System for Version Control. In: *Software - Practice and Experience* 15 (1985), Nr. 7, S. 637–654

[TM06] Tewari, Vijay ; Milenkovic, Milan: Standards for Autonomic Computing. In: *Intel® Technology Journal* (2006), November, 275–284. `http://www.intel.com/technology/itj/2006/v10i4/3-standards/1-abstract.htm`

[TPC07] Transaction Processing Performance Council (Hrsg.): *TPC Benchmark C. Standard Specification. Revision 5.9.* Transaction Processing Performance Council, 2007

[TPC08] Transaction Processing Performance Council (Hrsg.): *TPC Benchmark H. Standard Specification. Revision 2.7.0.* Transaction Processing Performance Council, 2008

[UG96] Uschold, Mike ; Grüninger, Michael: Ontologies: Principles, Methods, and Applications. In: *Knowledge Engineering Review* 11 (1996), S. 93–136

[VDB01] Vuduc, Rich ; Demmel, James ; Bilmes, Jeff: Statistical Models for Automatic Performance Tuning. In: *Proceedings of the International Conference on Computational Sciences-Part I.* London, UK, UK : Springer-Verlag, 2001 (ICCS '01), 117–126

[vKV00] van Deursen, Arie ; Klint, Paul ; Visser, Joost: Domain-specific Languages: An Annotated Bibliography. In: *SIGPLAN Not.* 35 (2000), Juni, S. 26–36

[Vos07] Voss, Jakob: Tagging, Folksonomy & Co - Renaissance of Manual Indexing? In: *Open Innovation : Neue Perspektiven im Kontext von Information und Wissen. Beiträge des 10. Internationalen Symposiums für Informationswissenschaft und der 13. Jahrestagung der IuK-Initiative Wissenschaft Köln.* Köln : UVK Verlagsgesellschaft mbH, Januar 2007, 243–254

[Vos08] Vossen, Gottfried: *Datenmodelle, Datenbanksprachen und Datenbankmanagementsysteme.* 5. Auflage. Oldenbourg, 2008

[vT03] van der Aalst, Wil M. P. ; Ter Hofstede, Arthur H. M.: YAWL: Yet Another Workflow Language. In: *Information Systems* 30 (2003), S. 245–275

[vTKB03] van der Aalst, Wil M. P. ; Ter Hofstede, Arthur H. M. ; Kiepuszewski, Bartek ; Barros, Alistair P.: Workflow Patterns. In: *Distributed Parallel Databases* 14 (2003), Juli, 5–51. `http://portal.acm.org/citation.cfm?id=640475.640516`

[W3C04] W3C (Hrsg.): *Document Object Model (DOM) Level 3 Core Specification. Version 1.0. W3C Recommendation* . W3C, April 2004. `http://www.w3.org/TR/2004/REC-DOM-Level-3-Core-20040407/`

[WBTT04] Wahli, Ueli ; Brown, Jennie ; Teinonen, Matti ; Trulsson, Leif: *Software Configuration Management: A Clear Case for IBM Rational Clearcase and Clearquest UCM.* Riverton, NJ, USA : IBM Corp., 2004

[WDC03] Wang, Yuan ; DeWitt, David J. ; Cai, Jin-Yi: X-Diff: An Effective Change Detection Algorithm for XML Documents. In: *International Conference on Data Engineering* (2003), S. 519

[Weg02] Wegner, Holm: *Analyse und objektorientierter Entwurf eines integrierten Portalsystems für das Wissensmanagement.* Hamburg, Technische Universität Hamburg-Harburg, Dissertationsschrift, Dezember 2002

[Wei97] Weik, Thomas: *DISDBIS.* Bd. 37: *Terminierung und Konfluenz in einer aktiven objektorientierten Datenbank.* Infix Verlag, St. Augustin, Germany, 1997

[WFM08] Workflow Management Coalition (Hrsg.): *Process Definition Interface – XML Process Definition Language. Document Number WFMC-TC-1025. Version 2.1a.* Workflow Management Coalition, Oktober 2008

[Whi91] Whitgift, David: *Methods and Tools for Software Configuration Management.* New York, NY, USA : John Wiley & Sons, Inc., 1991

[WHMZ94] Weikum, Gerhard ; Hasse, Christof ; Mönkeberg, Axel ; Zabback, Peter: The COMFORT Automatic Tuning Project. In: *Information Systems* 19 (1994), Juli, 381–432. `http://dl.acm.org/citation.cfm?id=189746.189748`

[WHW90] Widjojo, Surjatini ; Hull, Richard ; Wile, Dave: A Specificational Approach to Merging Persistent Object Bases. In: *4th International Workshop on Persistent Object Systems*, Morgan Kaufmann, 1990, S. 267–278

[Wie05] Wiese, David: *Framework for Data Mart Design and Implementation in DB2 Performance Expert*, Institut für Informatik, Friedrich-Schiller-Universität Jena, Diplomarbeit, März 2005

[Wie11] Wiese, David: *Gewinnung, Verwaltung und Anwendung von Performance-Daten zur Unterstützung des autonomen Datenbank-Tuning*, Institut für Informatik, Friedrich-Schiller-Universität Jena, Dissertationsschrift, Mai 2011

[WMHZ02] Weikum, Gerhard ; Moenkeberg, Axel ; Hasse, Christof ; Zabback, Peter: Self-tuning Database Technology and Information Services: From Wishful Thinking to Viable Engineering. In: *Proceedings of the 28th International Conference on Very Large Data Bases*, VLDB Endowment, 2002 (VLDB '02), 20–31

[WPT+04] Wang, Dongwen ; Peleg, Mor ; Tu, Samson W. ; Boxwala, Aziz A. ; Ogunyemi, Omolola ; Zeng, Qing ; Greenes, Robert A. ; Patel, Vimla L. ; Shortliffe, Edward H.: Design and Implementation of the GLIF3 Guideline Execution Engine. In: *Journal of Biomedical Informatics* 37 (2004), Oktober, 305–318. `http://portal.acm.org/citation.cfm?id=1044278.1044279`

[WR07a] Wiese, David ; Rabinovitch, Gennadi: A Generic Knowledge Model for Autonomic Database Performance Tuning. Poster. In: *4. Konferenz zum Professionellen Wissensmanagement: Erfahrungen und Visionen*. Potsdam, März 2007

[WR07b] Wiese, David ; Rabinovitch, Gennadi: *Autonomie ohne Aufpreis – Selbstverwaltende Datenbankmanagementsysteme.* Fachgruppentreffen der GI-Fachgruppe Datenbanksysteme: Selbstoptimierende und autonome Datenbanksysteme. Vortrag. Darmstadt, November 2007

[WR07c] Wiese, David ; Rabinovitch, Gennadi: *DB2 Autonomic Performance Management*. IBM CAS Technical Report. IBM Confidential, Juni 2007

[WR09] Wiese, David ; Rabinovitch, Gennadi: Knowledge Management in Autonomic Database Performance Tuning. In: *Proceedings of the Fifth International Conference on Autonomic and Autonomous Systems*. Washington, DC, USA : IEEE Computer Society, April 2009, 129–134

[WR10] Wiese, David ; Rabinovitch, Gennadi: *Evaluierung autonomer Mechanismen in heutigen DBMS*. Forschungsbericht. Lehrstuhl für Datenbanken und Informationssysteme. Friedrich-Schiller-Universität Jena, Februar 2010

[WRRA08] Wiese, David ; Rabinovitch, Gennadi ; Reichert, Michael ; Arenswald, Stephan: Autonomic Tuning Expert: A Framework for Best-practice oriented Autonomic Database Tuning. In: *Proceedings of the 2008 Conference of the Center for Advanced Studies on Collaborative Research: Meeting of Minds (CASCON 2008)*. New York, NY, USA : ACM, 2008, S. 3:27–3:41

[WRRA09] Wiese, David ; Rabinovitch, Gennadi ; Reichert, Michael ; Arenswald, Stephan: ATE: Workload-oriented DB2 Tuning in Action. Demo/Poster. In: *Tagungsband zur 13. GI-Fachtagung Datenbanksysteme für Business, Technologie und Web (BTW) 2009*. Münster, März 2009

[ZS89] Zhang, Kaizhong ; Shasha, Dennis: Simple Fast Algorithms for the Editing Distance between Trees and Related Problems. In: *SIAM Journal on Computing* 18 (1989), Nr. 6, S. 1245–1262

[ZSS92] Zhang, Kaizhong ; Statman, Rick ; Shasha, Dennis: On the Editing Distance between Unordered Labeled Trees. In: *Information Processing Letters* 42 (1992), Nr. 3, S. 133–139

Ehrenwörtliche Erklärung

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 habe, keine Textabschnitte oder Ergebnisse eines Dritten oder eigenen Prüfungsarbeiten ohne Kennzeichnung übernommen und alle von mir benutzten Hilfsmittel, persönliche 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.

Berlin, 2. Februar 2012