Technische Universität Ilmenau

Fakultät für Informatik und Automatisierung

Institut für Biomedizinische Technik und Medizinische Informatik

Fachgebiet Biosignalverarbeitung

Konzipierung und Realisierung einer Software zum Import von ARIS-Modellen in MLDesigner

Diplomarbeit
zur Erlangung des akademischen Grades „Diplom-Informatiker“

vorgelegt von: Danny Ammon

Adresse: Kleine Bahnhofstraße 15a, 98544 Zella-Mehlis

Telefon: (0 36 82) 47 87 37

E-Mail: DannyAmmon@CompuServe.de

Matrikelnummer: 31224

Verantwortlicher Hochschullehrer: Univ.-Prof. Dr. (USA) Horst Salzwedel

Hochschulbetreuer: apl. Prof. Dr.-Ing. habil. Vesselin Detschew, Dipl.-Inf. Katja Eisentraut

Inventarisierungsnummer: 2006-06-01/077/IN00/2222

Einreichungsdatum: 17.08.2006

Im Rahmen des klinischen Prozessmanagements werden klinisch-administrative Vorgänge mit informationstechnischen Mitteln modelliert und simuliert. Zwei hierfür nutzbare Werkzeuge sind das ARIS Toolset und der MLDesigner. In der vorliegenden Arbeit soll eine Möglichkeit geschaffen werden, mit dem ARIS Toolset erstellte Prozessmodelle automatisch und möglichst vollständig und fehlerfrei in den MLDesigner zu transferieren. Zu diesem Zweck werden die beiden Modellierungssysteme zunächst analysiert und ihre Modelltypen sowie deren Elemente näher erläutert und miteinander verglichen. Es zeigt sich, dass die Möglichkeit der Darstellung von klinischen Prozessen mit Hilfe von ereignisgesteuerten Prozessketten (EPK) eine starke Ähnlichkeit gegenüber der Blockstruktur von Modellen im MLDesigner aufweist. Speziell die EPK-Objekte Ereignis, Funktion bzw. Regel sind problemlos als MLDesigner-Modul bzw. Primitiv beschreibbar. Im zweiten Teil der Betrachtungen wird diese theoretische Vorarbeit genutzt, um den Prototyp eines Konverters für ARIS-Toolset-Modelle in das MLDesigner-Format zu entwerfen und zu realisieren. Die XML-Exportfunktion des ARIS Toolset und das XML-Speicherformat des MLDesigner ermöglichen dabei eine Datentransformation mittels XSLT; auf Grund der unterschiedlichen Betriebssysteme, unter denen die beiden Softwaresysteme operieren (Windows kontra Linux), eignet sich die betriebssystemunabhängige Programmiersprache Java für die Implementierung. Das so erstellte Werkzeug zum Transfer von ARIS-Modellen in den MLDesigner hat damit die Form einer Java-Applikation mit XSLT-Einbindung, die sowohl unter Windows als auch unter Linux lauffähig ist. Der fertige ARIS-MLDesigner-Konverter wird nach einer Erläuterung der in seinem Quellcode verwendeten Methoden einer Validierung und Leistungsbewertung unterzogen; schließlich werden Ansätze für weitere Forschungen auf dem Gebiet der Modellkonvertierung mit Augenmerk auf offene Standards aufgezeigt.

Inhaltsverzeichnis

Abbildungsverzeichnis

Tabellenverzeichnis

Abkürzungsverzeichnis

ACM	Association for Computing Machinery
AML	ARIS Markup Language
API	Application Programming Interface
ASG	Allen Systems Group, Inc.
ARIS	Architektur integrierter Informationssysteme
BCS	Balanced Scorecard
BPM	Business Process Model (Geschäftsprozessmodell)
BPMC	Business Process Model Converter
CASE	Computer-Aided Software Engineering
CD-ROM	Compact Disc – Read Only Memory
DE	deutschsprachig
DIN	Deutsches Institut für Normung e. V.
DTD	Document Type Definition
DV	Datenverarbeitung
EBM	Evidence-Based Medicine (Evidenzbasierte Medizin)
EDV	Elektronische Datenverarbeitung
eEPK	erweiterte ereignisgesteuerte Prozesskette
eERM	erweitertes Entity-Relationship-Modell
EN	englischsprachig
EEG	Elektroenzephalogramm
EPC	Event-driven Process Chain
EPK	ereignisgesteuerte Prozesskette
EPML	EPC Markup Language
ERM	Entity-Relation
ext2	Second Extended File System
ext3	Third Extended File System
FAT	File Allocation Table
FSM	Finite State Machine (Endlicher Zustandsautomat)

FS	File System
GI	Gesellschaft für Informatik e. V.
GMDS	Deutsche Gesellschaft für Medizinische Informatik, Biometrie und Epidemiologie e. V.
GUI	Graphical User Interface (grafische Benutzerschnittstelle)
HTML	Hypertext Markup Language
HTTP	Hypertext Transfer Protocol
IBM	International Business Machines Corp.
ICD	International Statistical Classification of Diseases and Related Health Problems
ID	Identification Number
IDE	Integrated Development Environment (integrierte Entwicklungsumgebung)
IEEE	Institute of Electrical and Electronics Engineers, Inc.
ISBN	International Standard Book Number
ISO	International Organization for Standardization
IT	Information Technology (Informationstechnologie)
IWi	Institut für Wirtschaftsinformatik, Universität des Saarlandes
JAR	Java Archive
JDK	Java Software Development Kit
JRE	Java Runtime Environment
MAR	MLDesigner Archive
MML	MLDesigner Markup Language
MSXML	Microsoft XML and XSLT Processor
MOSAIK-M	Modellierung, Simulation und Animation von Informations- und Kommunikationssystemen in der Medizin
NT	New Technology
NTFS	NT File System
OLE	Object Linking and Embedding
P2A	Processes to Applications
PC	Personal Computer

PDF	Portable Document Format
PEPE	Process Endorsing Prototype Engine
PHP	PHP: Hypertext Preprocessor
PL	Ptolemy Language
PTCL	Ptolemy Tool Command Language
RSA	Rational Software Architect
RSM	Rational Software Modeler
SAP	Systeme, Anwendungen und Produkte in der Datenverarbeitung
SGB	Sozialgesetzbuch
SML	Standard Meta Language
TAR	Tape Archiver
TMP	Temporary Files (temporäre Dateien)
UML	Unified Modeling Language
UNIX	Uniplexed Information and Computing System
VDI	Verein Deutscher Ingenieure e. V.
VKD	Vorgangskettendiagramm
W3C	WWW Consortium
WKD	Wertschöpfungskettendiagramm
WWW	World Wide Web
XML	Extended Markup Language
XPath	XML Path Language
XSL	Extended Stylesheet Language
XSLT	XSL Transformations

1 Einleitung

1.1 Modellierung und Simulation klinischer Prozesse

Als umfangreiches sozioökonomisches System, das immer wieder große Veränderungen durchläuft, ist das Gesundheitswesen in Deutschland auf ein adäquates Management und eine Sicherung der hier angestrebten Qualität sowohl aus wirtschaftlichen als auch aus gesetzgeberischen Gründen (SGB V, §§ 135-137) angewiesen.

Qualitätsmanagement in der Medizin umfasst nach DIN/EN/ISO9000:2000 „aufeinander abgestimmte Tätigkeiten zum Lenken und Leiten einer Organisation, die darauf abzielen, die Qualität der produzierten Produkte oder der angebotenen Dienstleistung zu verbessern. Konkret handelt es sich dabei um alle Maßnahmen und Tätigkeiten, durch die die Qualitätspolitik, Ziele und Verantwortungen in einem Betrieb festgelegt sowie diese durch Mittel wie Qualitätsplanung, Qualitätslenkung, Qualitätssicherung und Qualitätsverbesserung verwirklicht werden."

In diesem Sinne lässt sich die angestrebte Qualität nach [Donabedian 1986] in drei Kategorien einteilen:

Strukturqualität bezeichnet die Beschreibung der für die medizinische Versorgung im Einzelfall gegebenen Rahmenbedingungen,

Prozessqualität beinhaltet die Eigenschaften aller medizinischen Tätigkeiten innerhalb und zwischen den Anbietern und Verbrauchern von Gesundheitsdienstleistungen,

Ergebnisqualität schließlich ist die Erläuterung der dem medizinischen Handeln zuschreibbaren Veränderungen des Gesundheitszustandes eines Patienten [Pietsch-Breitfeld u. Selbmann 1997, S. 157]

Als für den medizinischen Sektor hochrelevant wird darüber hinaus als vierte Kategorie die *Lebensqualität* bezeichnet:

Lebensqualität ist die Verlängerung der Lebenszeit eines Patienten bei gleichzeitiger Verbesserung seiner Lebensumstände [Kunhardt et al. 2005, S. 779f.].

Es existieren verschiedene Methoden und Instrumente, um die jeweiligen Qualitätskategorien dem Management zuzuführen, so etwa Qualitätszirkel, Ringversuche, Benchmarkings, die Evidenzbasierte Medizin oder Managed-Care-Modelle [Kunhardt et al. 2005, S. 796–803]. Bezogen auf Verständnis, Management und Sicherung der *Prozessqualität*, also der Effizienz und Effektivität aller zeitlichen Abläufe während der Behandlung von Patienten, lässt sich eine Optimierung klinischer Prozesse, die konträr etwa zu industriellen Prozessen ihrer Natur nach meist hochkomplex und schwer planbar sind, vor allem im administrativen Bereich durch informationsverarbeitende Systeme mit Hilfe des *Workflow-Management* erreichen.

Ein Workflow ist ein formal beschriebener und ganz oder teilweise automatisierter Geschäftsprozess. Er beinhaltet die für eine automatische Steuerung des Arbeitsablaufes auf einer operativen Ebene erforderlichen zeitlichen, fachlichen und ressourcenbezogenen Spezifikationen. Die anzustoßenden Arbeitsschritte sind dann jeweils zur Ausführung durch Mitarbeiter oder Anwendungsprogramme vorgesehen [Gadatsch 2003, S. 33].

Workflow-Management stellt Methoden und Werkzeuge zur computergestützten Analyse, Planung, Simulation, Steuerung und Überwachung von Arbeitsabläufen bereit [Gadatsch 2003, S. 40].

Ein integraler Bestandteil eines Workflow-Management im medizinischen Bereich ist damit eine Modellierung klinischer Prozesse.

Als Modell eines klinischen Prozesses wird dabei die Zusammenstellung der Menge aller Informationen über diesen Prozess betrachtet, die für seine Untersuchung relevant sind, so dass seine Komplexität und Planbarkeit in akzeptable Bereiche gelangen [Gumbel et al. 2005, S. 200].

Die Nachbildung der Realität in einem Modell mit Hilfe der Informationstechnik ist den *Grundsätzen ordnungsgemäßer Modellierung* unterworfen [Rosemann 1996, S. 94–104]. Sie

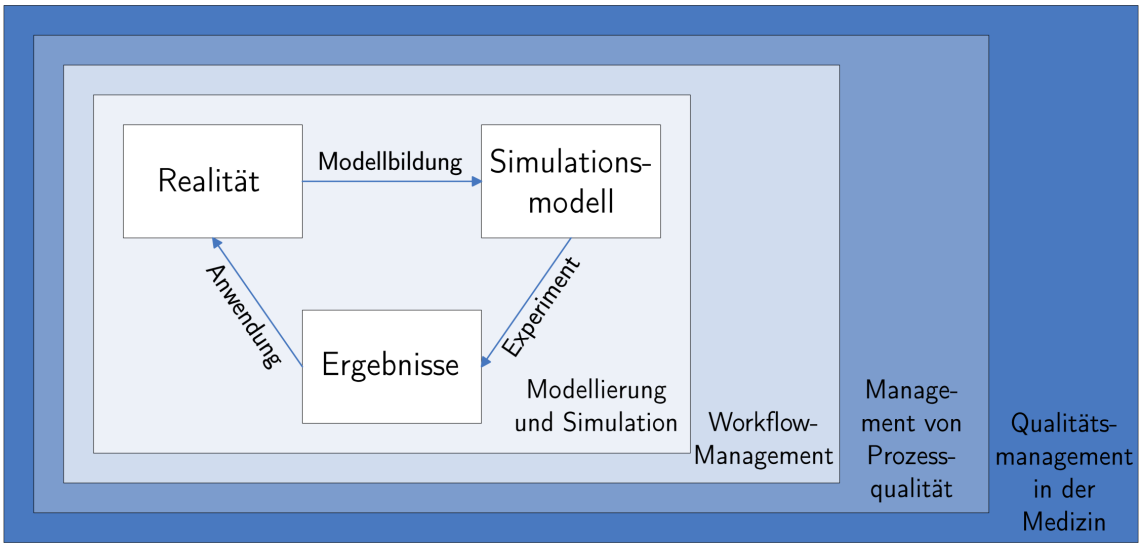

Abbildung 1.1: Themenbereich der vorliegenden Arbeit (nach [Gadatsch 2003, Abb. 171])

hat den Vorteil, dass die hierfür verwendete Software oftmals in der Lage ist, die Abläufe nicht nur modellhaft darzustellen, sondern auch mittels eigener Parameter konkrete Vorfälle zu simulieren.

Die Simulation von klinischen Prozessen ist eine parametrisierte Nachahmung konkreter Abläufe innerhalb dieses Prozesses mit Hilfe eines formalen Modells zum Zwecke des Prozessstudiums und seiner Optimierung [Heidenberger 1990, S. 452]. Unterschieden wird hierbei zwischen *kontinuierlichen*, durch Differentialgleichungen beschriebene Simulationen und *diskreten*, algorithmisierbaren Verfahren.

Ergebnisse einer solchen Simulation können wiederum dem Qualitätsmanagement z. B. im Sinne eines Benchmarking zugeführt und damit in der Realität angewendet werden. Eine Veranschaulichung der erläuterten Zusammenhänge bietet Abb. 1.1.

Durch die Modellierung und Simulation klinischer Prozesse mit den Mitteln einer spezifischen Software geht aber die allgemeine, standardisierte Form einer vereinfachten Prozessdarstellung, also ein Teil des Abstraktionsprozesses bei der Modellierung, zugunsten eines proprietären Speicherformats verloren. Eine Möglichkeit zur Extraktion der Modelldaten in allgemeiner Form aus solchen Speicherformaten (vgl. auch [Büyükyilmaz u. Forbrig 2003]) soll in dieser Arbeit vorgestellt werden.

1.2 Aufgabenstellung

Wie im vorangegangenen Abschnitt erläutert, ist es Aufgabe der Medizinischen Informatik, klinisch-administrative Prozesse aus Gründen des Qualitätsmanagement als Workflow zu modellieren und zu simulieren, um zu einer Optimierung spezifischer (variabler und steuerbarer) Prozessparameter gelangen zu können. Bei dieser Tätigkeit wird eine korrespondierende Software-Kategorie verwendet, wobei der Nutzer hier die Wahl zwischen einer Großzahl an Modellierungs-, Simulations- und Workflow-Management-Werkzeugen hat, welche mehr oder weniger auf verschiedene Branchen abgestimmt sind, was bei der Nutzung im klinischen Umfeld zu beachten ist, um die dortigen Anforderungen an solche Systeme zu erfüllen [Schäfer et al. 1998].

Wenn das Erarbeiten von Modellen und Simulationen mit Hilfe eines konkreten Programms einer Abstraktion im Sinne der hier angebotenen Formalisierungsmechanismen Grenzen setzt, so werden diese durch eine notwendige Übertragung erstellter Modelle von einer bestimmten Software in eine andere aufgehoben. Das Thema der vorliegenden Arbeit ist eine solche Übertragung von klinischen Prozessmodellen.

Eine aus der Wirtschaftsinformatik stammende und damit für ökonomische Prozesse ausgelegte Modellierungs- und Simulations-Software ist die ARIS-Plattform. Sie baut auf die von Prof. Dr. Dr. hc mult. August-Wilhelm Scheer zu Beginn der 90er Jahre vorgestellte Architektur integrierter Informationssysteme auf [Scheer 1997, 1998, 2001], bei der es sich um eine ganzheitliche Beschreibung von Informationssystemen aus fünf zentralen Perspektiven handelt, deren spezifische Details als Vorgangsketten formalisiert werden können. Sowohl das ARIS-Referenzmodell in der Wirtschaftsinformatik als auch die ARIS-Plattform in der Wirtschaft selbst sind langjährig etablierte bzw. marktführende Instrumente auf dem Gebiet der Geschäftsprozess- und Workflow-Modellierungssysteme [Gadatsch 2003, S. 159f.]

Ein weiteres, an der Technischen Universität Ilmenau mitentwickeltes und aus dem Gebiet der System- und Steuerungstheorie kommendes Softwaresystem ist MLDesigner. Dabei handelt es sich um eine integrierte Multidomänen-Plattform für die Modellierung, Simulation und Analyse der Architektur, Funktion und Leistung jeglicher Formen von komplexen System-Designs [MLDesign 2004a, S. 1].

Mit MLDesigner werden Modelle grafisch als hierarchische Blockdiagramme dargestellt, deren kleinste Elemente konkrete Algorithmen sind, wodurch diese Modelle einer umfassenden

Analyse durch eine parametrisierte Simulation unterzogen werden können. Der MLDesigner stellt damit das allgemeinere, breiter anwendbare und funktionsreichere Tool für System- und Prozessmodellierung, Analyse und Simulation dar.

Eine Vielzahl von Modellen klinischer Prozesse liegt mit dem ARIS Toolset als Bestandteil der ARIS-Plattform implementiert vor. Die Aufgabenstellung der vorliegenden Arbeit ist die Konzeption und Realisierung einer Möglichkeit, diese ARIS-Modelle in automatisierter Form in Modelle des MLDesigner möglichst fehlerfrei zu konvertieren. Dabei sollen Elemente des Modells unverändert beibehalten und wenn möglich in MLDesigner-spezifische Pendants überführt werden. Durch diese Konvertierung können dann die umfangreichen Simulationsmöglichkeiten des MLDesigner genutzt werden, um die mit ARIS modellierten Workflows und Prozesse zu simulieren, zu optimieren und damit ihre Qualität bewerten und beeinflussen zu können.

1.3 Aufbau der Arbeit

Ausgehend von der im vorigen Abschnitt konkretisierten Problemstellung werden in den folgenden beiden Kapiteln zunächst die Grundlagen für die Lösung etabliert: Im ersten Teil (Kapitel 2) wird das ARIS-Konzept vorgestellt, die daraus hervorgegangene Software erfährt eine Kategorisierung und die grundlegenden Elemente, mit denen in ARIS Modelle konfiguriert werden, werden einer Analyse unterzogen. Schließlich wird der Einsatz von ARIS in der Medizinischen Informatik untersucht.

Der zweite Grundlagenteil befasst sich mit dem MLDesigner (Kapitel 3). Dazu wird zunächst eine kurze Einführung in die diesem Werkzeug zugrunde liegende Systemtheorie geboten. Auf dieser Basis wird der MLDesigner selbst näher vorgestellt. Auch die Elementbibliothek des MLDesigner wird näher analysiert, um schließlich den Einsatz des MLDesigner in der Medizinischen Informatik zu beleuchten.

Im folgenden Kapitel 4 wird der Forschungsstand zur hier behandelten Thematik aufgearbeitet; dabei wird vom Allgemeinen zum Speziellen hingeführt. Damit markiert das klinische Prozessmanagement als grober Themenrahmen der Arbeit den Beginn, um so über Modellierung und Simulation klinischer Prozesse hin zur spezifischen Konvertierung von ARIS-Modellen in andere Softwaresysteme zu führen und dort jeweils den Stand der aktuellen Arbeiten zu referieren.

Nach diesen Vorarbeiten kann mit der Konzeptmodellierung einer Konvertierung von Modellen aus ARIS in den MLDesigner begonnen werden (siehe Kapitel 5). Dazu werden ebenfalls vom Allgemeinen zum Speziellen führend zunächst die beiden Modellierungswerkzeuge verglichen, um daran eine Analyse der Möglichkeiten einer adäquaten Beschreibung von ARIS-Modellelementen im MLDesigner anzuschließen. Aus dieser Analyse wiederum kann dann eine Möglichkeit zum Export einzelner bereits bestehender ARIS-Modelle und Modellelemente abgeleitet werden, ebenso ein Verfahren zum Import dieser in den MLDesigner.

Im folgenden Kapitel wird das bis dahin erschlossene Konverter-Design mit den Begriffen der Softwaretechnik formuliert und konkretisiert. Der Anwendungsbereich des Konverters wird beschrieben und daraus die genauen Anforderungen an das Werkzeug ermittelt. Hieraus können dann konkrete formale Spezifikationen für die Implementierung abgeleitet werden. wodurch der Lösungsweg klar beschrieben ist.

Die programmtechnischen Details der Konverter-Implementierung werden im nächsten Schritt dokumentiert (Kapitel 7); die Konvertierung einzelner ARIS-Modelle mit XSLT in das MLDesigner-Format wird erläutert und deren Einbettung in einen mit Java programmierten Konverter, der aus einer Anzahl einzelner Klassen besteht, dargelegt, um so den eigentlichen Prototyp eines ARIS-MLDesigner-Konverters zusammenfassend darzustellen.

An die Dokumentation der Konverterprogrammierung schließt sich in einem weiteren Kapitel eine Validierung und Leistungsbewertung an, während derer die Richtigkeit und Vollständigkeit der realisierten Export-Import-Funktion geprüft und beurteilt wird, indem bestehende Modelle aus der klinischen Praxis, welche in ARIS etabliert wurden, in das MLDesigner-Format transferiert werden. Es wird verdeutlicht, welche ARIS-Modellbestandteile der Konverter übertragen kann, welche Struktur diese im MLDesigner annehmen und an welchen Stellen der Benutzer nacharbeiten muss, um die erweiterten Möglichkeiten des MLDesigner dazu einzusetzen, das aus dem ARIS Toolset importierte Modell mit implementierter Funktionalität und damit Simulierbarkeit zu versehen.

Abschließend werden die einzelnen Schritte zur Realisierung des Konverter-Prototyps noch einmal rekapituliert und es wird ein Ausblick auf mögliche Verbesserungen sowohl am hier entwickelten Konverter selbst, als auch mit den Mitteln des MLDesigner, um ARIS-Modellstrukturen einer noch detailgetreueren, weiter automatisierten Abbildung zuzuführen, durchgeführt. Mit diesen Betrachtungen schließt die vorliegende Arbeit.

2 ARIS

2.1 ARIS-Informationsmodell

Die Architektur integrierter Informationssysteme wurde im Jahre 1991 als Konzept zur Beschreibung von Unternehmen und betriebswirtschaftlichen Anwendungssystemen vorgestellt [Scheer 1991]. Als *integrierte Informationssysteme* wird dabei Anwendungssoftware verstanden, deren Komponenten betriebswirtschaftliche Aufgaben wie Ablaufsteuerung (Dispositionssystem), Bereitstellung von Führungsinformationen (Management-Informationssystem), einfache Datenverarbeitung (Administrationssystem) oder langfristige Entscheidungsvorgänge (Planungssystem) lösen und dabei durch eine integrierte Datenbasis miteinander verbunden sind, wodurch das Gesamtsystem in der Lage ist, beliebige Unternehmensprozesse vollständig abzubilden und zu unterstützen [Scheer 1997, S. 48].

2.1.1 ARIS-Haus

Das ARIS-Konzept wurde entwickelt, um eine Beschreibung von Geschäftsprozessen innerhalb und außerhalb der Anwendungssoftware mit reduzierter Komplexität zu erreichen, die dennoch ganzheitlich, aus allen Perspektiven und über alle Entwicklungsphasen verläuft. Zu diesem Zweck werden alle Elemente des Konzeptes in das sogenannte ARIS-Haus eingeordnet, eine übergeordnete Darstellung, welche die Beschreibung von Unternehmensprozessen in fünf zentrale Sichten und in vier verschiedene Phasen eines Life-Cycle-Modells strukturiert (siehe Abb. 2.1) [Scheer 1998, S. 2].

2.1.2 ARIS-Sichten

Die Zusammenfassung von Elementen und Begriffsklassen des ARIS-Konzepts in fünf Sichten wird in Abb. 2.1 deutlich, die das ARIS-Haus zeigt. Demzufolge existieren folgende ARIS-Sichten [Scheer 1997, S. 36], [Scheer 1998, S. 13], [IDS Scheer AG 2000, S. 6–8]:

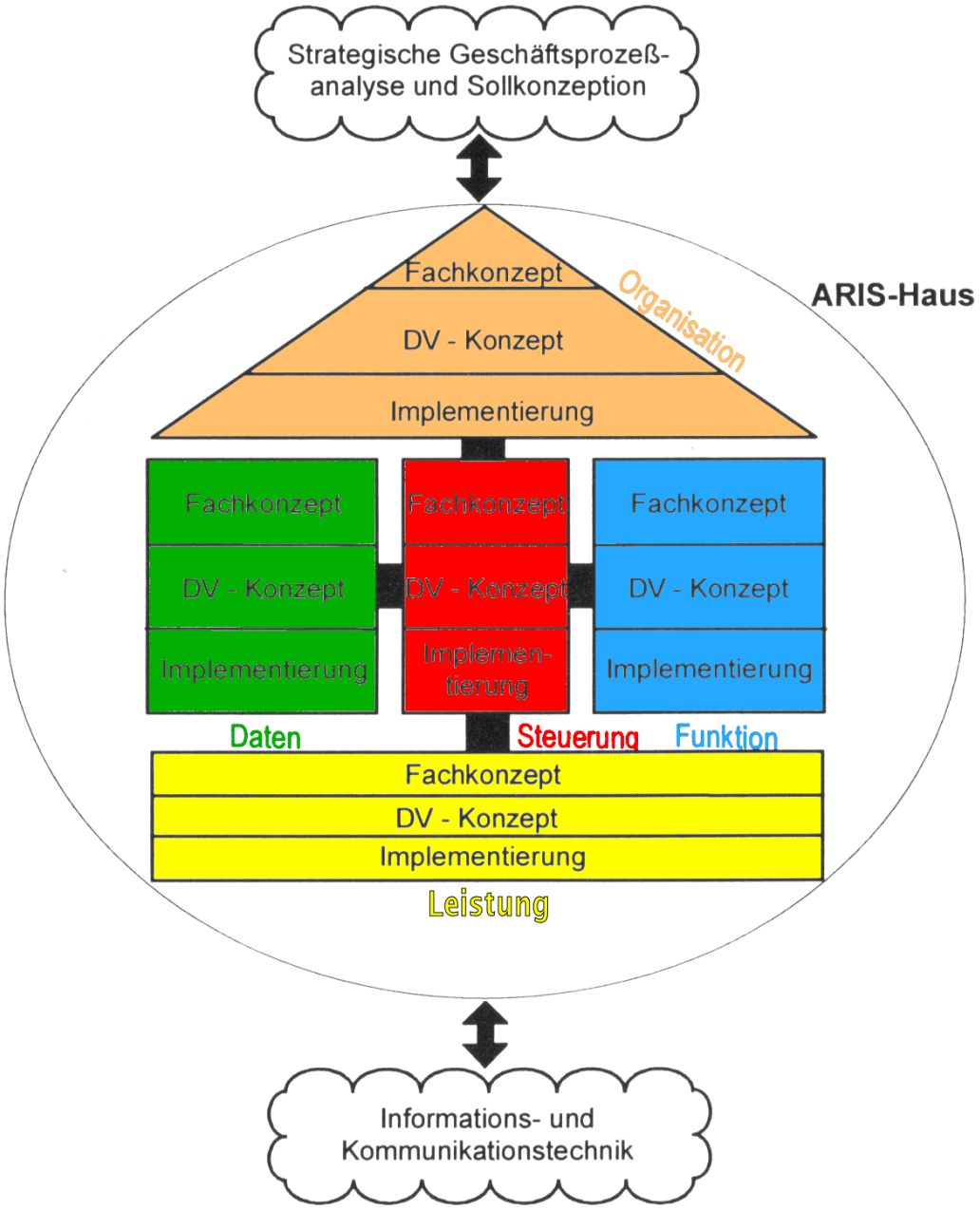

Abbildung 2.1: ARIS-Haus mit Phasenkonzept [Scheer 1998, Abb. 17]

Die Organisationssicht (orange) beschreibt die Struktur und Beziehungen von Organisationseinheiten (Aufgabenträger, Bearbeiter) eines Unternehmens. Das Darstellungsmittel der Wahl ist hier das *Organigramm*. In Organigrammen werden Organisationsstrukturen durch grafische Symbole wiedergegeben und über Kanten miteinander hierarchisch vernetzt [IDS Scheer AG 2000, S. 54–59], [IDS Scheer AG 2005a, S. 4-85–4-90]. Das ARIS-Konzept sieht verschiedene Objekttypen vor, wie etwa *Gruppe*,

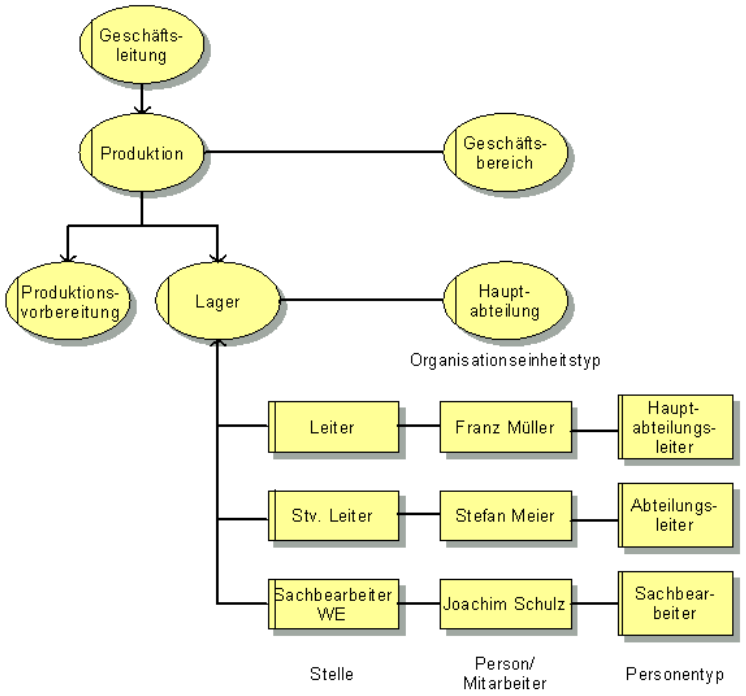

Abbildung 2.2: ARIS-Modelltypen – Organigramm [IDS Scheer AG 2005a, Abb. 4.3.1-5]

Stelle, *Person*, *Personentyp* und *Standort*, welche Art und Funktion einer Organisationseinheit konkretisieren können. Ein Beispiel für ein Organigramm findet sich in Abb. 2.2.

Die Datensicht (grün) umfasst die Umfelddaten einer Vorgangsbearbeitung sowie Nachrichten bzw. Ereignisse, die währenddessen ausgelöst und Zustände des Bezugsumfelds, die eingenommen werden. Ein *erweitertes Entity-Relationship-Modell* (eERM)

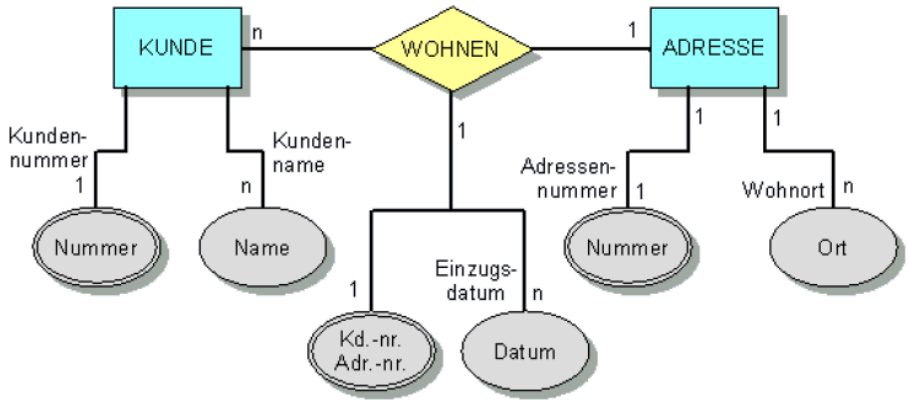

Abbildung 2.3: ARIS-Modelltypen – eERM [IDS Scheer AG 2005a, Abb. 4.2.1-7]

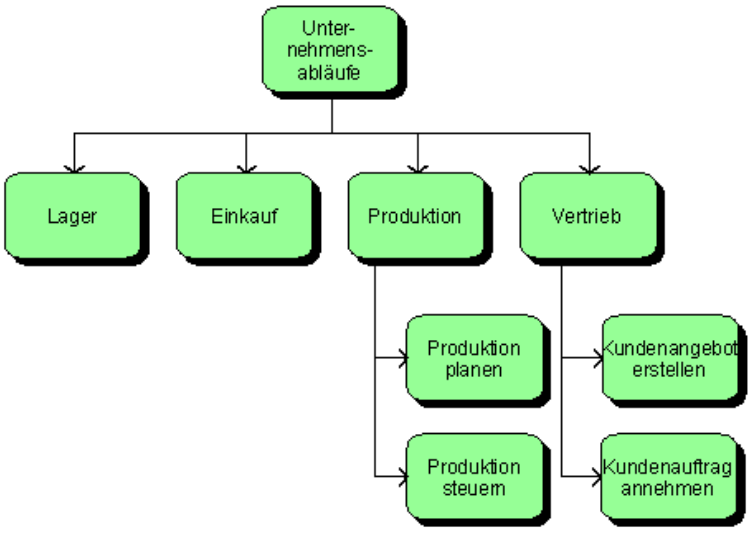

Abbildung 2.4: ARIS-Modelltypen – Funktionsbaum [IDS Scheer AG 2005a, Abb. 4.1.1-2]

wird für die Darstellung verwendet [Chen 1976]. In diesem Modell werden relevante abstrakte oder reale Dinge als Entities veranschaulicht, die durch Attribute und Beziehungen untereinander spezifiziert und eingeordnet werden können [IDS Scheer AG 2000, S. 258–262], [IDS Scheer AG 2005a, S. 4-22–4-50]. Erweiterungen betreffen dabei bestimmte Eigenschaften der ERM wie Konstruktionsoperatoren, Kardinalitäten und Abhängigkeiten. Ein Beispiel für ein eERM stellt Abb. 2.3 dar.

Die Funktionssicht (blau) enthält die Vorgänge oder Tätigkeiten, welche Input-Leistungen des Unternehmens zu Output-Leistungen transformieren sowie ihre Beziehungen untereinander. Durch hierarchische *Funktionsbäume* werden diese Vorgänge veranschaulicht [IDS Scheer AG 2000, S. 228 f.], [IDS Scheer AG 2005a, S. 4-2–4-7]. Abb. 2.4 zeigt beispielhaft einen Funktionsbaum.

Die Leistungssicht (gelb) beinhaltet alle materiellen und immateriellen Input- und Output-Leistungen, die in den Geschäftsprozess eingebracht oder währenddessen erzeugt werden. Sie können in *Leistungsbäumen* dargestellt werden, die verschiedene Abstraktionsebenen oder Substitutionsbeziehungen eines Produkts oder einer Dienstleistung veranschaulichen können [IDS Scheer AG 2000, S. 13 f.], [IDS Scheer AG 2005a, S. 4-188 f.]. Ein typischer Leistungsbaum findet sich in Abb. 2.5.

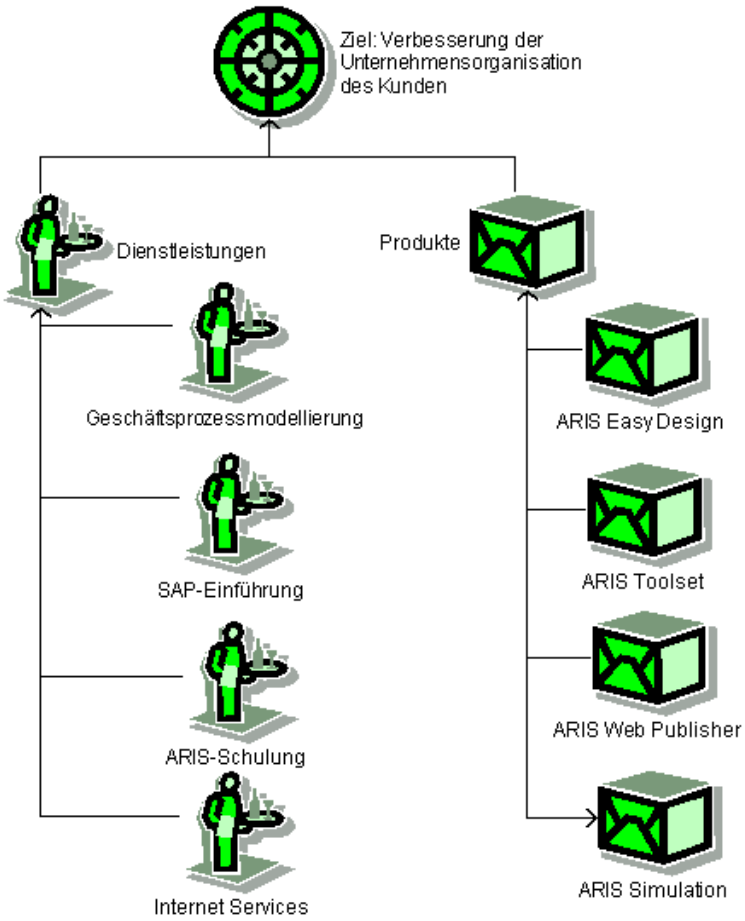

Abbildung 2.5: ARIS-Modelltypen – Leistungsbaum [IDS Scheer AG 2005a, Abb. 4.5-2]

Die Steuerungssicht (rot) ist der zentrale Teil des ARIS-Hauses. Sie erfasst die Beziehungen zwischen den vorangehenden Sichten sowie die Dynamik des gesamten Geschäftsprozesses. Sie stellt damit den Rahmen für eine vollständige Prozessbeschreibung bereit. Zu diesem Zweck existieren mehrere Darstellungsmöglichkeiten, zu denen

Objekttyp	eEPK
Ereignis	
Funktion	
Regel	

Abbildung 2.6: Elemente einer eEPK (nach [IDS Scheer AG 2005a, Abb. 4.4.1-36])

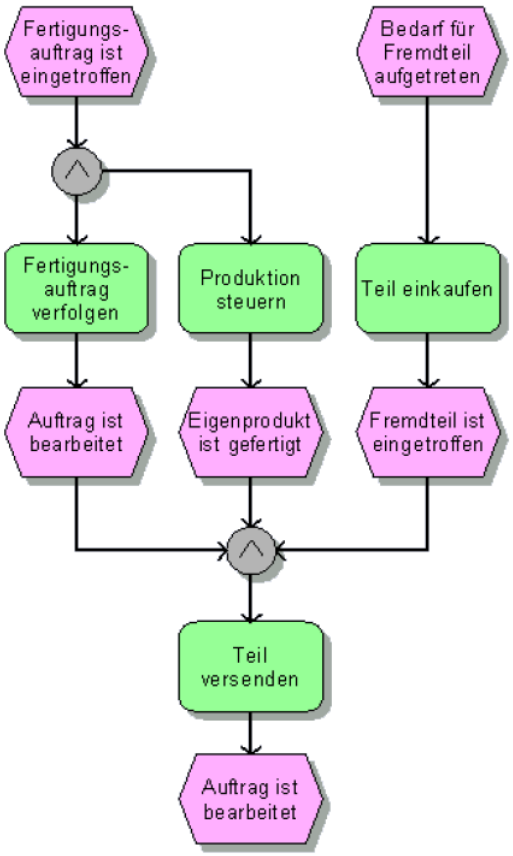

Abbildung 2.7: ARIS-Modelltypen – eEPK [IDS Scheer AG 2005a, Abb. 4.4.1-3]

Wertschöpfungskettendiagramme, *Funktionszuordnungsdiagramme* und *Office Processes*, hauptsächlich aber die *erweiterten ereignisgesteuerten Prozessketten* (eEPK) gehören. In eEPK werden die in Abb. 2.6 aufgeführten Objekttypen genutzt, um eine Darstellung von ereignisauslösenden Funktionen und deren logischer Verknüpfung miteinander zur Prozessbeschreibung zu erreichen [IDS Scheer AG 2000, S. 124 f.], [IDS Scheer AG 2005a, S. 4-105–4-115]. Ein Beispiel für eine eEPK, deren Funktion und Struktur in Abschnitt 5.1 näher betrachtet werden, ist in Abb. 2.7 zu finden.

2.1.3 ARIS-Phasen

Wie aus Abb. 2.1 außerdem ersichtlich ist, enthält das ARIS-Konzept ein Phasenmodell (schwarze Beschriftung), das über alle Sichten hinweg ein Life-Cycle-Modell für die Realisierung spezieller informationstechnischer Anwendungen aus den betriebswirtschaftlichen Fachbegriffen des Geschäftsprozesses etabliert [Scheer 1998, S. 38–40]:

Eine strategische Ausgangslösung beschreibt zunächst grundsätzliche Wirkungen der Informationstechnik auf neue Unternehmenskonzepte.

Im Fachkonzept werden dann die einzelnen Sichten des Anwendungssystems durch formale Beschreibungssprachen strukturiert.

Das DV-Konzept (Datenverarbeitungs-Konzept) ermittelt aus den Anforderungen des Fachkonzeptes konkrete Spezifikationen.

Die technische Implementierung schließlich setzt die Spezifikationen in physische Datenstrukturen, Hardware-Komponenten und Programme um.

Das ARIS-Informationsmodell enthält zusätzlich zur übergeordneten Struktur des ARIS-Hauses noch eine Vielzahl spezieller betriebswirtschaftlicher Management-, Modellierungs- und Vorgehensprinzipien, etwa das ARIS-House of Business Engineering als Business-Process-Management-Modell [Scheer 1996], konkrete Grundsätze und Ebenen ordnungsgemäßer Modellierung [Scheer 1998, S. 119–131] oder Vorgehensmodelle zur Geschäftsprozessoptimierung [Scheer 1998, S. 149–153] und zum Qualitätsmanagement [Scheer 1998, S. 154–161] sowie zur unternehmensinternen Software-Einführung und Systementwicklung [Scheer 2001, S. 177–196]. Es erweist sich dadurch als breit angelegtes und vielfältiges Konzept zur Anwendung informationstechnischer Prinzipien, Methoden und Kenntnisse in der Betriebswirtschaft.

2.2 ARIS Platform

Die Inhalte des vorgestellten ARIS-Informationsmodells werden von der IDS Scheer AG in einer Sammlung fertiger Anwendungssysteme umgesetzt, welche für das Geschäftsprozess-Management in diversen Branchen entwickelt wurden. Die ARIS Platform unterteilt dabei diese Systeme in vier Kategorien [IDS Scheer AG 2005b]:

1. Systeme der *Strategy Platform* zum Unternehmensstrategie-Management:

- ARIS Balanced Scorecard (BSC),
- ARIS Business Optimizer zur Kennzahlenermittlung und Szenario-Simulation

2. Systeme der *Design Platform* als Prozess-Design, -Analyse und -Optimierungs-Tools:

- ARIS Business Architect als funktionsreichstes System zum verteilten Geschäftsprozess-Management,
- ARIS Business Designer als erweitertes Business-Process-Management-Tool,
- ARIS Toolset als Standardanwendung zum Management von Unternehmensprozessen,
- ARIS Simulation als Programm zur Simulation von Geschäftsabläufen,
- ARIS Business Publisher und ARIS Web Publisher als Publikations-Tools,
- ARIS Quality Management Scout als Anwendung zum Qualitätsmanagement

3. Systeme der *Implementation Platform* als Software zur Anwendungsentwicklung:

- ARIS for SAP NetWeaver,
- ARIS UML Designer,
- ARIS Processes to Applications (P2A),
- ARIS Redocumentation Scout,
- ARIS Software Engineering Scout

4. Systeme der *Controlling Platform* als Controlling-Instrumente:

- ARIS Process Performance Manager als Ist-Prozess-Analyse-Tool,
- ARIS Audit Manager als Audit-Workflow-System,
- ARIS Process Risk Scout als Risikomanagementsystem

Darüber hinaus bietet die ARIS Platform noch zahlreiche branchenspezifische Lösungen an, etwa die ARIS Healthcare Solution als Prozessmanagement-Anwendung für Institutionen des Gesundheitswesens.

In den folgenden Betrachtungen dieser Arbeit wird jedoch nur auf das ARIS Toolset als das zentrale, verbreitetste und dienstälteste Modellierungssystem nach den Prinzipien des ARIS-Informationsmodells Bezug genommen.

2.3 ARIS Toolset

Auf Basis des ARIS-Konzeptes stellt das ARIS Toolset eine große Zahl wissenschaftlich fundierter und praxiserprobter Verfahren des Software Engineering zugeschnitten auf die betriebswirtschaftliche Aufgabe des Geschäftsprozess-Management zur Verfügung [Gadatsch 2003, S. 164].

Als vollständige Umsetzung des ARIS-Informationsmodells bietet das ARIS Toolset in der aktuellen Version 7.0 über 150 verschiedene Modelle, über 200 verschiedene Modellelemente (Objekte) und über 1400 Attribute an, um Unternehmensprozesse umfassend beschreiben zu können [Davis 2001, S. 61]. Die wichtigsten Modelltypen werden in Abb. 2.8 noch einmal den zugehörigen Sichten im ARIS-Haus zugeordnet. Die Wahl des Modelltyps wird dabei in Abhängigkeit des jeweiligen Projektzieles getroffen [IDS Scheer AG 2000, S. 8]. Um bei der Modellierung die Übersicht bewahren zu können, lassen sich spezielle Methodenfilter auswählen oder selbst definieren, welche die Auswahl der Objekte auf die aktuell relevanten beschränken können [Davis 2001, S. 61–63].

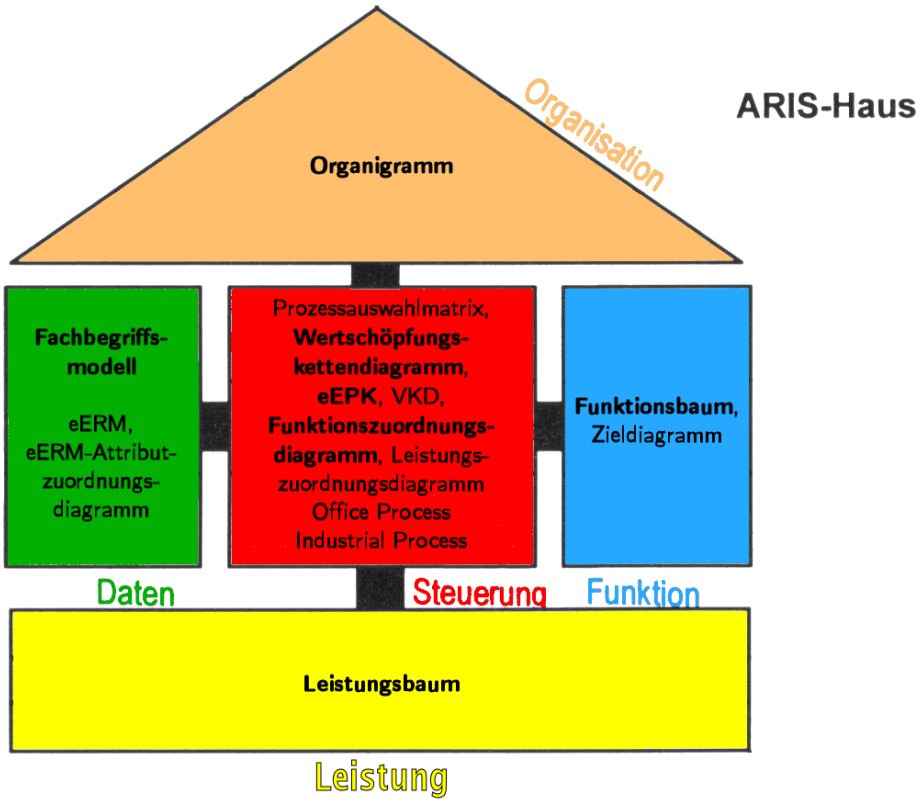

Abbildung 2.8: Auswahl von ARIS-Modelltypen (nach [IDS Scheer AG 2000, Abb. 1.9])

2.3.1 Datenbank

Alle mit dem ARIS Toolset erstellten Modelle und eingegebenen Informationen werden in einer zentralen Datenbank gespeichert, die entweder auf dem eigenen PC als lokaler Server oder, bei der Vernetzung mehrerer Computer miteinander, auf einem zentralen Server angelegt werden kann [IDS Scheer AG 2000, S. 9], [Davis 2001, S. 21 f.]. Zu Übersichts-, Zuordnungs- und Hierarchiezwecken sind alle Inhalte der Datenbank durch eine Verzeichnisstruktur unterteilbar, deren Ordner ebenfalls als Datenbanken bezeichnet werden.

2.3.2 ARIS Explorer

Die Inhalte der Datenbank werden beim Start des ARIS Toolset in einer zweispaltigen Ansicht des sogenannten ARIS Explorer angezeigt [Davis 2001, S. 55–64]. Der ARIS Explorer ist die administrative Zentrale des Toolset, mit deren Hilfe sich bestehende Modelle anzeigen, verwalten, löschen und neue Ordner, Datenbanken und Modelle anlegen lassen. Eine Beispielansicht des Explorer ist in Abb. 2.9 wiedergegeben.

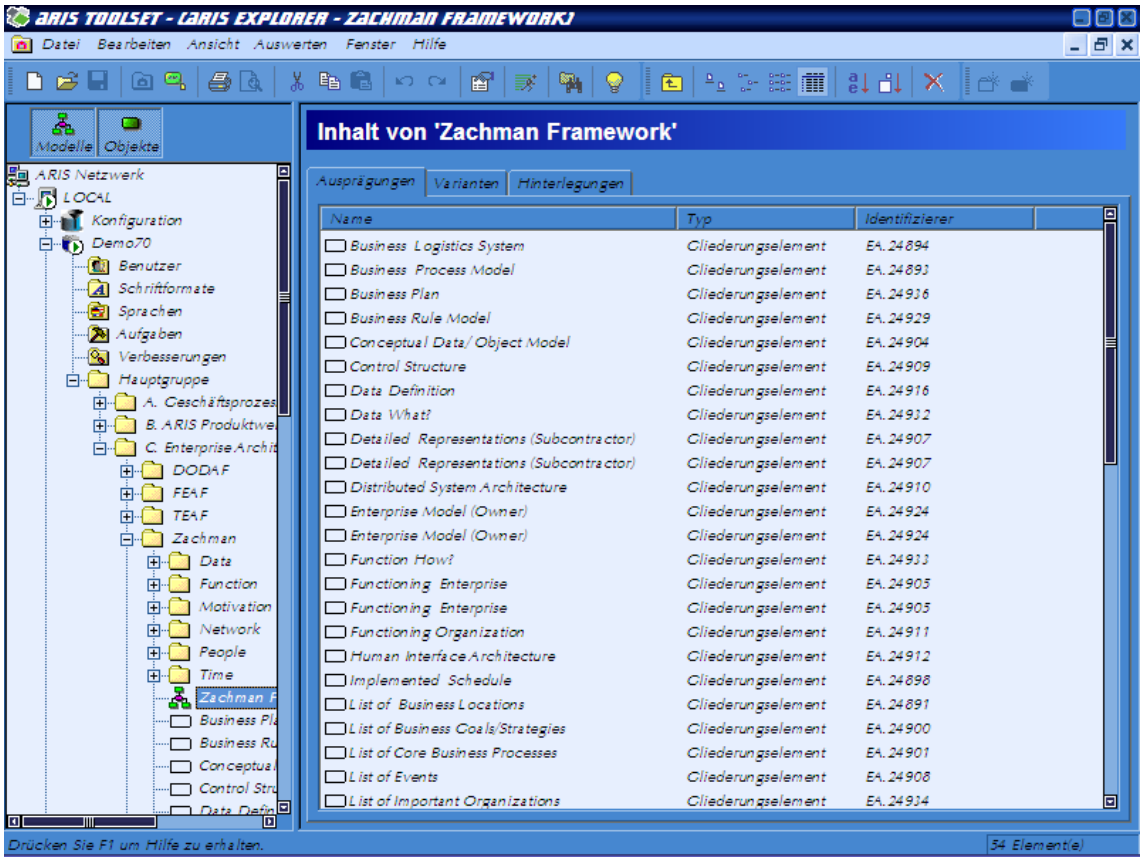

Abbildung 2.9: GUI des ARIS Explorer

2.3.3 ARIS Designer

Neben dem ARIS Explorer zur Navigation dient der ARIS Designer als weiterer Bestandteil des Toolset zur Erstellung und Bearbeitung von Modellen [Davis 2001, S. 65–84]. Er enthält die Modellierungsfläche, umgeben von Symbolleisten, die es dem Nutzer erlauben, alle Objekte, sofern nicht durch Methodenfilter deren Anzahl limitiert wurde, auszuwählen, auf der Modellierungsfläche zu platzieren und durch Kanten zu verbinden (siehe Abb. 2.10). Auf diese Weise können alle vom ARIS-Konzept unterstützten und durch das Toolset realisierbaren Modelltypen umgesetzt und gespeichert werden.

Auch das Hinterlegen von einzelnen Objekten mit weiteren Modellen als hierarchische Struktur ist möglich [Davis 2001, S. 117 f.].

Die beim Erstellen von ARIS-Modellen nutzbaren Objekte als Modellierungselemente sollen im Folgenden einer näheren Analyse unterzogen werden.

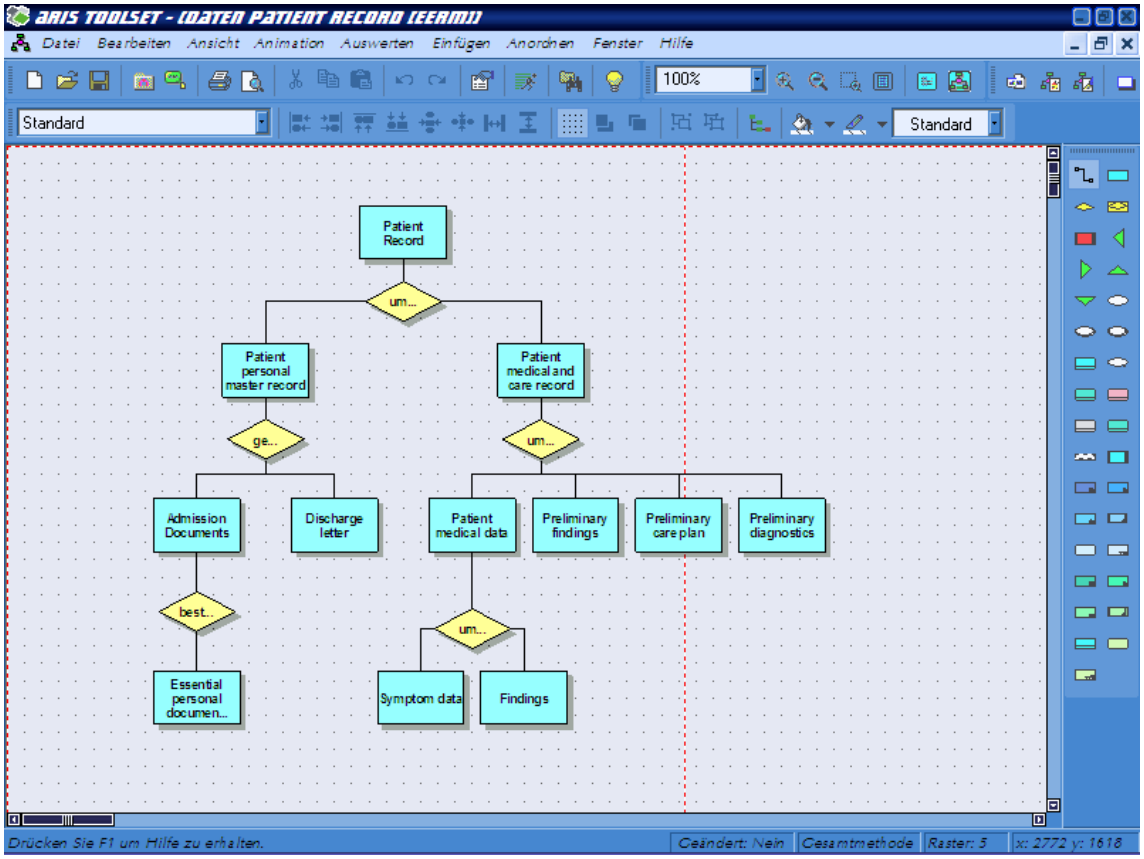

Abbildung 2.10: GUI des ARIS Designer

2.4 Modellierungselemente in ARIS

Die bereits erläuterte Vielfalt von möglichen Modelltypen, um in ARIS bzw. mit dem ARIS Toolset Unternehmensprozesse zu beschreiben, bedingt eine korrespondierend große Anzahl einzelner Modellierungselemente, von denen eine bestimmte Menge jeweils spezifisch für einen bestimmten Modelltyp ist (beispielsweise Organisationseinheiten wie Gruppe, Stelle, Person, Personentyp und Standort für ein Organigramm, siehe Abschnitt 2.1.2, Abb. 2.2). Entsprechend den freien Modellierungsmöglichkeiten mit ARIS können aber praktisch alle Objekte in allen Modelltypen zum Einsatz kommen. Eine Eingrenzung dieser Vielfalt lässt sich mit Methodenfiltern realisieren, welche die Auswahl auf bestimmte, am häufigsten gebrauchte Objekte reduzieren.

Die folgende Analyse der Modellierungselemente von ARIS umfasst eine Auswahl nach drei Kriterien:

1. die gebräuchlichsten und grundlegenden Objekte,
2. diejenigen Objekte, welche explizit für die Prozessmodellierung zur Verfügung stehen,
3. Objekte für die Erstellung von Modellen aus der Perspektive der Steuerungssicht, also vornehmlich Wertschöpfungskettendiagramme, Vorgangskettendiagramme und die erweiterten ereignisgesteuerten Prozessketten (eEPK).

Die Begründung für diese Auswahl wird im Abschnitt 5.1 getroffen, der sich mit der sinnvollen Umsetzbarkeit von Modellen aus dem ARIS Toolset in MLDesigner-Format beschäftigt.

2.4.1 Ereignisse

Das zentrale Objekt einer prozessbasierten Modellierung ist das (Elementar-)Ereignis. Ereignisobjekte sind Bestandteil der Datensicht und wie folgt definiert [IDS Scheer AG 2005a, S. 4-105]:

Ein Ereignis innerhalb des ARIS-Konzeptes ist das Eingetretensein eines betriebswirtschaftlich relevanten Zustandes eines Informationsobjektes, der den weiteren Ablauf des Geschäftsprozesses steuert oder beeinflusst. Ereignisse lösen Funktionen aus und sind

Ergebnisse von Funktionen. Im Gegensatz zu einer Funktion, die ein zeitverbrauchendes Geschehen darstellt, ist ein Ereignis auf einen Zeitpunkt bezogen.

Ereignisse werden grafisch durch Sechsecke dargestellt (siehe Abb. 2.11).

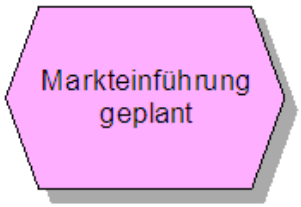

Abbildung 2.11: ARIS-Objekttypen – Ereignis

2.4.2 Funktionen

Objektausprägung der Funktionssicht ist die einzelne Funktion.

Eine Funktion ist eine fachliche Aufgabe bzw. Tätigkeit an einem Objekt zur Unterstützung eines oder mehrerer Unternehmensziele.

Die grafische Darstellung einer Funktion erfolgt mittels eines Rechtecks mit abgerundeten Ecken (siehe Abb. 2.12).

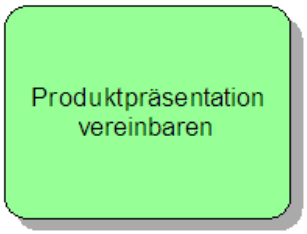

Abbildung 2.12: ARIS-Objekttypen – Funktion

2.4.3 Regeln

In einer ereignisgesteuerten Prozesskette können von einem Ereignis mehrere Funktionen gleichzeitig ausgehen, auch kann eine Funktion mehrere Ereignisse als Ergebnis haben. Diese Verzweigungen werden durch logische Verknüpfungen ausgedrückt.

Regeln sind logische Verknüpfungen, die Ereignisse und Funktionen miteinander verbinden können.

Eine Regel wird durch einen Kreis mit dem entsprechenden Verknüpfungstyp als Beschriftung dargestellt. Es sind AND-, OR- und XOR- (Exklusiv-Oder)-Verknüpfungen möglich (siehe Abb. 2.13).

Die erlaubten Formen von Verbindungen durch Regeln innerhalb von eEPK sind in Abschnitt 5.1 näher erläutert.

Abbildung 2.13: ARIS-Objekttypen – Regeln

2.4.4 Daten

Wie in Abschnitt 2.1.2 beschrieben, werden Zusammenhänge zwischen Umfelddaten eines Unternehmensprozesses in der Datensicht von ARIS mit erweiterten Entity-Relationship-Modellen veranschaulicht. Solche komplexen ERM können in der Modellierung in Datenclustern zusammengefasst werden.

Ein Datencluster beschreibt eine logische Sicht auf eine Ansammlung von Entity- und Beziehungstypen eines Datenmodells, die zur Beschreibung eines komplexen Objektes benötigt wird.

Datencluster werden wie in Abb. 2.14 gezeigt dargestellt.

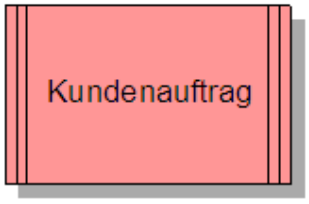

Abbildung 2.14: ARIS-Objekttypen – Datencluster

2.4.5 Organisationseinheiten

Organisationseinheiten werden hier repräsentativ für Objekte der Organisationssicht und damit ursprünglich Organigrammen entstammend vorgestellt.

Organisationseinheiten sind Träger der zur Erzielung der Unternehmensziele durchzuführenden Aufgaben.

Die Darstellung von Organisationseinheiten erfolgt wie in Abb. 2.15.

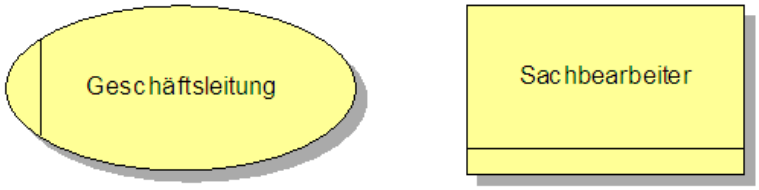

Abbildung 2.15: ARIS-Objekttypen – Organisationseinheiten

2.4.6 Weitere Objekttypen

Weitere zentrale Objekte des ARIS-Informationsmodells sind *Leistungen*, *Ziele*, *Software* oder *Ressourcen*. Sie entsprechen jedoch nicht den im Vorfeld festgelegten Kriterien und werden daher hier nicht näher betrachtet.

2.4.7 Eigenschaften und Attribute von Objekttypen

Alle Objekte, die sich mit dem ARIS Toolset modellieren lassen, besitzen bestimmte Eigenschaften. Zu den Grundtypen zählen:

- **Name** des Objekts
- **Kanten**, also Verbindungen zu anderen Objekten (auch als *Beziehungen* bezeichnet)
- **Hinterlegungen** eines Objekts mit weiteren Modellen
- **Verknüpfungen** / OLE-Objekte, die mit dem Objekt aufgerufen werden können
- **Objektdarstellung**, grafische Optionen wie Textplatzierung, Farbe, Größe etc.

Neben diesen Grundeigenschaften lassen sich jedem Objekt noch eine Vielzahl von Attributen zuordnen. Eine spezifische Menge von Attributen ist in jedem Objekt bereits mit Werten belegt (siehe Tab. 2.1). Alle anderen Attribute dagegen sind optional, bis hin zur Existenz von hunderten freier Attribute, welche der Nutzer mit eigenen Werten nach seinen Vorstellungen füllen kann (siehe Tab. 2.2).

Attributname	**Attributtyp**
Name	Text
Typ	Objekttyp
Erstellzeitpunkt	Datum/Uhrzeit
Ersteller	Benutzername
letzte Änderung	Datum/Uhrzeit
letzter Bearbeiter	Benutzername

Tabelle 2.1: Kernattribute von ARIS-Objekten

Attributklasse	**Attributbeispiel**
Systemattribute	Verknüpfungen
Analyseattribute	Regelattribut
Fremdsystem-Attribute	SAP-ID
Change Management	Verbesserungspotenzial
Gültigkeit	Beginn
Workflow	Hinterlegungsart
Simulation	Einarbeitungszeit
freie Attribute	Benutzerattribut verschiedenen Datentyps

Tabelle 2.2: Weitere Attribute von ARIS-Objekten

2.5 Einsatz von ARIS in der Medizinischen Informatik

Das ARIS-Konzept als Informationsmodell aus der Wirtschaftsinformatik und als Modellierungssystem für Workflows im Unternehmen findet in der Medizinischen Informatik zweierlei Anwendungsansätze – zum einen die Top-Down-Methode der Modellierung von konkreten Abläufen etwa in einem bestimmten Krankenhaus, zum anderen der Bottom-Up-Ansatz des allgemeinen Qualitätsmanagements.

2.5.1 Top-Down-Ansatz

Zum einen sind Institutionen des Gesundheitswesens in ihrer Eigenschaft als ökonomische Systeme durch den Einsatz von Informationssystemen, branchenspezifisch als Medizinische Informationssysteme, gekennzeichnet. Kernaufgabe dieser Informationssysteme ist die Abbildung von Geschäftsprozessen, hier im Sinne von klinischen Abläufen. Die Architektur integrierter Informationssysteme ermöglicht hierfür z. B. den Einsatz von ereignisgesteuerten Prozessketten (EPK, siehe Abschnitt 5.1), um Ereignisse, Aufgaben und auslösende Verbindungen zwischen diesen als Informationsobjekte herauszuarbeiten [Haas 2005, S. 25], [Ammenwerth u. Haux 2005, S. 154–158].

Generell kann das ARIS-Konzept zur modellhaften Abbildung etwa eines Krankenhauses als Unternehmen durch eine Geschäftsarchitektur dienen [Haas 2005, S. 61], [Ammenwerth u. Haux 2005, S. 167 f.], [Scheer et al. 1996a], [Scheer et al. 1996b].

2.5.2 Bottom-Up-Ansatz

Zum anderen kann etwa das speziell auf die Architektur integrierter Informationssysteme ausgerichtete ARIS Toolset der Firma IDS Scheer AG genutzt werden, um unternehmensunabhängig klinische Prozesse zu modellieren und so etwa einem Qualitätsmanagement zuzuführen.

Ein Beispiel für ein solches Modell liegt mit der Umsetzung des *Siemens Process Framework*, einer umfangreichen hierarchischen Abstraktion von Krankenhaus-Prozessen in Form von Behandlungspfaden [Holzner u. Fleischer 2004], mit Hilfe des ARIS Toolset in [Detschewa 2005] vor.

Allgemein ist die Architektur integrierter Informationssysteme mit ihren zahlreichen Modellierungsmöglichkeiten eine adäquate Form für die Abbildung und das Management klinisch-ökonomischer Prozesse [Scheer et al. 1996a], [Scheer et al. 1996b].

3 MLDesigner

3.1 Systemtheoretische Grundlagen

Im Kontrast zur betriebswirtschaftlich-informationstechnischen Basis, auf der die Architektur integrierter Informationssysteme und somit auch das ARIS Toolset entwickelt wurde, ist der MLDesigner ein Werkzeug, dessen Wurzeln in der Systemtheorie verhaftet sind.

Die Systemtheorie ist ein interdisziplinäres Erkenntnismodell für die Beschreibung unterschiedlichster komplexer Phänomene. Ein Systemmodell ist mathematisch abbildbar (meist durch Differentialgleichungen) und beschreibt die Funktionsweise des Systems durch Regelkreisschemata [von Bertalanffy 1969].

Da die Systemtheorie insgesamt eine heterogene Metatheorie für die unterschiedlichsten Begriffe und Definitionen bildet, mit denen physikalische, biologische, psychische und soziale Phänomene erklärt werden sollen, wird sich die Darstellung im Folgenden auf die ingenieurwissenschaftlichen bzw. kybernetischen Formulierungen beschränken, welche auch dem MLDesigner zugrunde liegen.

Zentral für diese Perspektive der Systemtheorie sind die Begriffe *Signal* und *System*.

3.1.1 Signale

Ein Signal ist eine Funktion oder eine Wertefolge, die Information repräsentiert [Girod et al. 2005, S. 4].

Es existiert damit eine Vielzahl an Daten, die als Signal interpretiert werden können und somit auch verschiedene Möglichkeiten der Kategorisierung von Signalen. Die wichtigsten Einteilungen unterscheiden zwischen kontinuierlichen und diskreten bzw. zwischen determinierten und stochastischen Signalen.

Beispiele für Signale sind Sprache als Schallphänomen (kontinuierlich), relative Häufigkeiten z. B. bei Würfel-Zufallsexperimenten (diskret), die 230-V-Wechselspannung aus einer Steckdose (deterministisch) oder ein EEG eines Menschen (stochastisch).

3.1.2 Systeme

Systeme sind Prozesse oder Gebilde, die Signale umwandeln [Unbehauen 2002, S. 1]. In der Systemtheorie werden diese als Modelle abgebildet, die dann ebenfalls Systeme heißen.

Modellierte Systeme werden als *black box* dargestellt, die eine Anzahl an Eingangssignalen und eine Anzahl an Ausgangssignalen besitzt (siehe Abb. 3.1) [Girod et al. 2005, S. 5 f.].

Aufgabe der Systemtheorie ist damit die Untersuchung von Fragestellungen zu den konkreten Eigenschaften eines Systems (Systemanalyse), etwa wie es auf eine bestimmte Klasse von Signalen reagiert (Systemcharakterisierung), welches Systemmodell sich am besten zur Repräsentation eines praktischen Systems eignet (Systemidentifikation) usw. [Unbehauen 2002, S. 1].

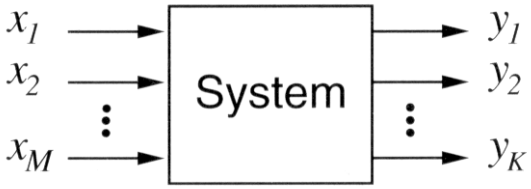

Abbildung 3.1: Ein-Ausgangssystemmodell [Girod et al. 2005, Abb. 1.8]

3.1.3 Weitere relevante systemtheoretische Begriffe

Ein zentrales Instrument bei der Untersuchung von Systemen bildet die Umformung der Ein- und Ausgangsfunktionen vom Zeit- in den Frequenzbereich durch Laplace- oder Fourier-Transformation [Unbehauen 2002, S. 199–538], [Girod et al. 2005, S. 43–248].

Ein wichtiger Begriff zur Charakterisierung von Systemen ist dann die *Übertragungsfunktion* $H(\mathrm{j}\omega)$, welche im Frequenzbereich das Ausgangssignal eines Systems zum Eingangssignal in Beziehung setzt:

$$H(\mathrm{j}\omega) = \left.\frac{y(t)}{x(t)}\right|_{x(t)=\mathrm{e}^{\mathrm{j}\omega t}}$$

Als bedeutende Klasse von Systemen sind die sogenannten *rückgekoppelten Systeme* zu nennen, bei denen Ausgangssignale neben der eigentlichen Erregung ebenfalls als Eingangs-

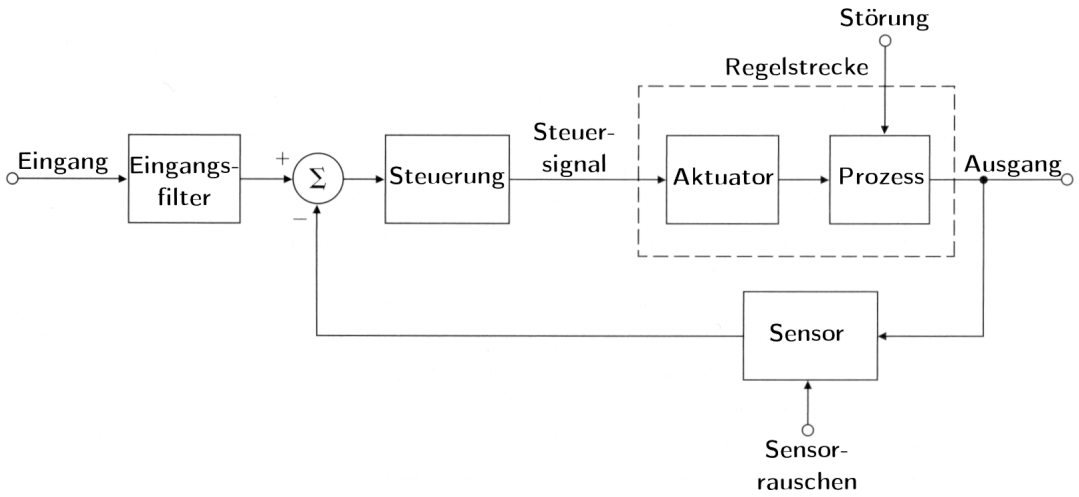

Abbildung 3.2: Blockdiagramm eines typischen rückgekoppelten Systems

[Franklin et al. 2002, Abb. 1.2]

signale einwirken [Unbehauen 2002, S. 168]. Eine Vielzahl physikalischer und elektrischer dynamischer Systeme ist durch Rückkopplung von Ausgangsgrößen effizient steuerbar.

Solche rückgekoppelten Systeme wie Systeme allgemein lassen sich als Regelkreise veranschaulichen. Ein Beispiel für einen Regelkreis ist in Abb. 3.2 als Blockdiagramm mit den typischen Elementen eines rückgekoppelten Systems (Sensoren, Aktuatoren, Steuerung etc.) dargestellt.

Auf diese systemtheoretischen Grundlagen baut die Modellierungs- und Simulationssoftware MLDesigner auf, die regelungstechnisch beschreibbare Systeme (z. B. elektronische Bauteile) ebenso wie komplexe Prozesse mit diesen Mitteln umfassend abbilden kann und so ein effizientes Design und sofortige Überprüfung auf Korrektheit solcher Systeme ermöglicht.

3.2 MLDesigner als Werkzeug

3.2.1 Eigenschaften

Die Simulations- und Modellierungsplattform MLDesigner (in dieser Arbeit verwendet wurde die Version 2.5.r00) vereint die Abbildung komplexer Systeme und Prozesse in Architektur und Funktion mit deren Simulation durch konkrete Parameter verschiedenen Cha-

rakters in einem Werkzeug [Schorcht et al. 2003, S. 2].

Modelle können in unterschiedlichen *Domänen* realisiert werden, welche zeitkontinuierliche Vorgänge oder diskrete Ereignisse, endliche Zustandsautomaten und drei Datenflüsse (dynamisch, synchron und Boole'sch) einschließen. Dabei ist auch die Nutzung von Elementen aus verschiedenen Domänen in einem heterogenen (*Multi-Domänen-*)Modell möglich.

Die Modellierung findet im MLDesigner über *hierarchisch strukturierte Blockdiagramme* statt [MLDesign 2004a, S. 1]. Einzelne Blöcke, *Module* genannt, besitzen Ein- und Ausgangskanten, durch die sie mit dem Gesamtmodell verbunden und mit Steuerinformationen und Daten versorgt werden. Die Verarbeitung dieser Informationen geschieht in C++-Quellcode, der wiederum in weiteren Blöcken, sogenannten *Primitiven*, gekapselt ist. Ein Modell besteht damit aus einem Netz einzelner Module, welche selbst aus Modulen oder aber aus Primitiven bestehen können (siehe Abb. 3.3).

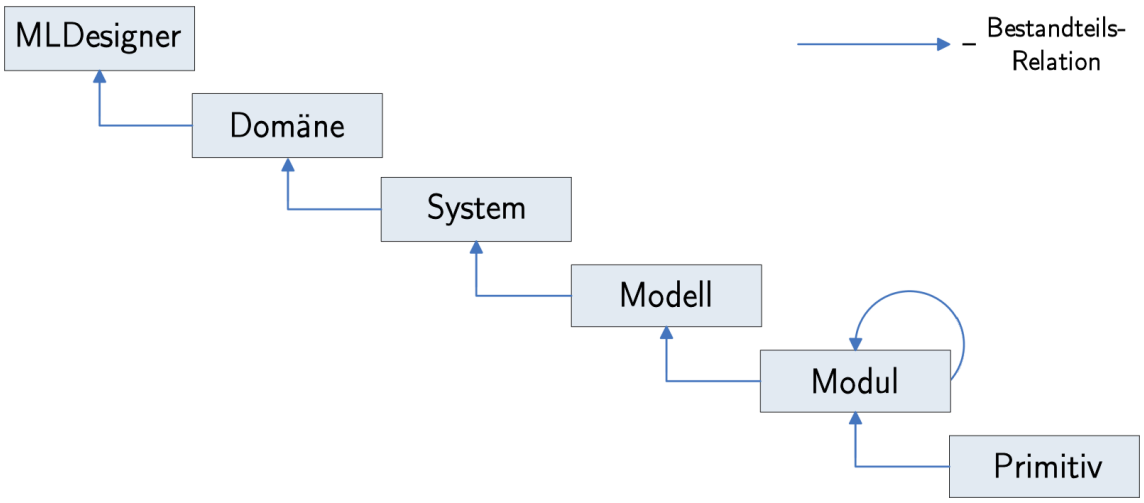

Abbildung 3.3: MLDesigner-Bestandteilhierarchie

3.2.2 Bedienungsprinzipien

MLDesigner ist ein grafisches Werkzeug für den Betrieb unter Linux oder Solaris mit laufendem X Server [MLDesign 2004c, S. A-2].

Für die Erstellung von Modellen und deren Simulation bietet der MLDesigner folgende Hilfsmittel [MLDesign 2004c, S. -1-7 f.]:

- *grafische Benutzeroberfläche* für das Erstellen und Editieren der Blockdiagramme und entsprechender Parameter
- Ptolemy Tool Command Language (PTCL)-*Kommandoumgebung* zur befehlsgesteuerten Modellgenerierung
- Möglichkeiten zur *Animation* und zum *Ergebnisausdruck* für die Auswertung von Simulationsergebnissen

Für die Modellierungsarbeit enthält der MLDesigner bereits mehr als 2000 Primitive, kategorisiert in verschiedene Bibliotheken, sowie über 200 fertige Beispielmodelle [MLDesign 2004a, S. 2].

Erstellte Modelle werden mit dem MLDesigner in sogenannten Librarys, also Bibliotheken gespeichert, die selbst angelegt werden können und eine Modellhierarchie mit ihren Einzelbestandteilen miteinander verbinden und auflisten [MLDesign 2004a, S. 6].

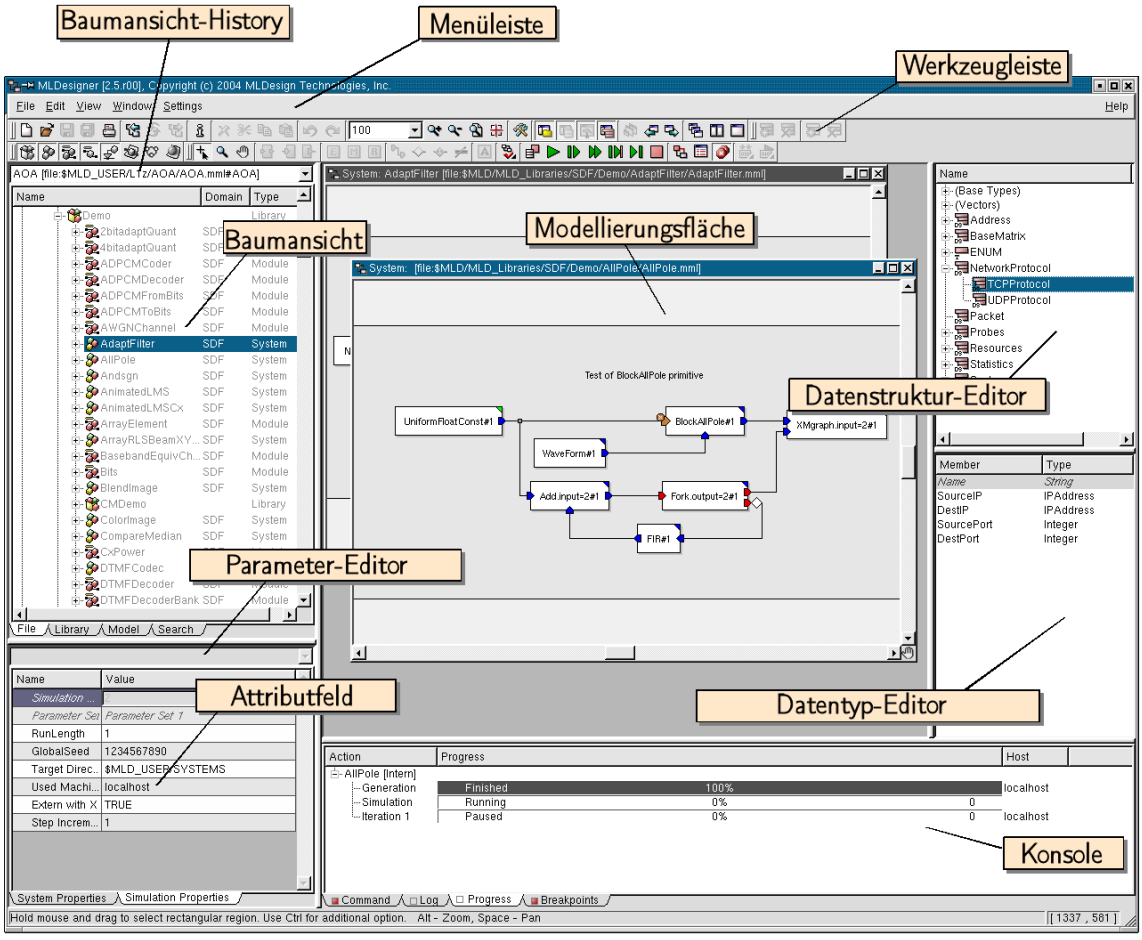

Abbildung 3.4: MLDesigner-Benutzeroberfläche [MLDesign 2004a, Abb. 1.2]

Demzufolge besteht der Bildschirmaufbau beim MLDesigner neben der Modellierungsfläche aus einem Navigationsfenster mit Baumansicht, in dem die Librarys angezeigt werden, einen Parametereditor, einer Konsole für PTCL-Eingaben und Rückmeldungen des Systems sowie zwei Fenstern für die Manipulation von Datenstrukturen (siehe Abb. 3.4) [MLDesign 2004a, S. 3 f.].

Erstellte Modelle können im MLDesigner auf ihre Gültigkeit hin überprüft werden, Fehler wie nicht vorhandene Verbindungen etc. werden dann dem Nutzer angezeigt [MLDesign 2004c, S. 24-15 f.].

Ein Beispiel für ein fertiges MLDesigner-Modell ist in Abb. 3.5 mit dem mitgelieferten Demo *testPacket* dargestellt.

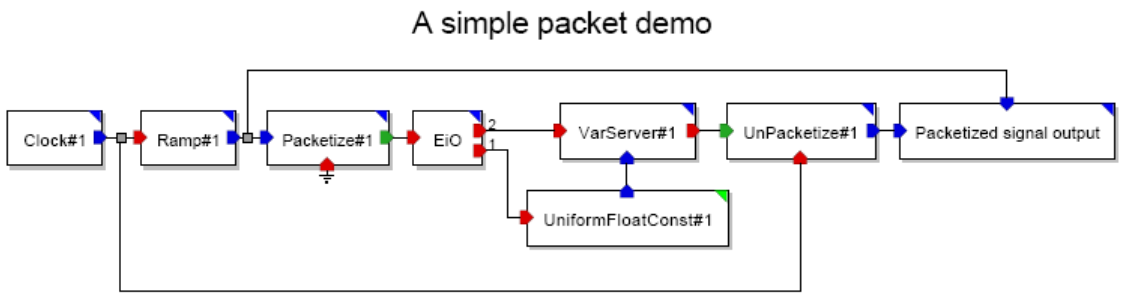

Abbildung 3.5: MLDesigner-Beispielmodell *testPacket* [MLDesign 2004a, Abb. 1.3]

3.3 Modellierungselemente in MLDesigner

Im vorangehenden Abschnitt wurden die Modellierungsprinzipien des MLDesigner vorgestellt, die mit der Einordnung der zu erstellenden Modelle (Systeme) in spezifische Domänen beginnen. Alle Modelle werden dann in Module und Primitive gegliedert.

3.3.1 Module

Ein Modul ist ein Teil eines Systems, das aus weiteren Systemelementen besteht. Es besitzt Ein- und Ausgänge zur Verbindung mit dem Restsystem und kann eine Einzelstruktur, aber auch eine Hierarchie weiterer Module oder Primitive sein [MLDesign 2004c, S. 1-2].

3.3.2 Primitive

Primitive sind die Elementarstrukturen im MLDesigner. Sie bestehen aus zwei Teilen, dem *Modell-* und dem *Code-Teil*. Der Modellteil eines Primitivs enthält die Ein- und Ausgänge sowie das externe Interface (die Parameter), während der Code-Teil durch spezifischen C++-Code gekennzeichnet ist, welcher zur Modellsimulation ausgeführt wird und die gewünschte Funktion eines Modellbestandteils realisiert. Alternativ kann ein sogenanntes FSM-Primitiv im Code-Teil durch einen endlichen Zustandsautomaten (finite state machine) beschrieben werden.

3.3.3 Parameter

Sowohl Modulen als auch Primitiven können bestimmte Attribute in Form von Parametern zugewiesen werden. Sie werden als sogenannte *formale Parameter* bei der Modellierung definiert und enthalten dann während der Simulation spezifische Werte, die *aktuellen Parameter* [MLDesign 2004c, S. 3-11].

Es lassen sich zu jedem Modellelement beliebige Parameter definieren, welche stets eine feste Anzahl konkreter Eigenschaften besitzen müssen [MLDesign 2004c, S. 3-12 f.]:

- **Name** – ein eindeutiger Bezeichner für den Parameter
- **Data Type** – ein spezifischer Datentyp (siehe Tab. 3.1)
- **Data Structure** – im Falle des Datentyps `datastruct` können hier konkrete Daten festgelegt werden
- **Value** – ein Anfangswert
- **Attributes** (nur bei Primitiven) – Parameterattribute
- **Description** – eine Parameterbeschreibung, die in der HTML-Dokumentation, welche automatisch für jedes Modell generiert wird, eingebunden wird

3.3.4 Beispiel-Primitive

Wie in Abschnitt 3.2.2 angegeben, werden mit dem MLDesigner mehr als 2000 fertige Primitive mitgeliefert. Eine Auswahl davon soll hier näher beschrieben werden. Dabei wird

aus Gründen, die in Abschnitt 5.1 erläutert sind, auf die DE-Domäne (diskrete Ereignisse) zurückgegriffen [MLDesign 2004c, S. 22-8–22-10]. Innerhalb dieser Domäne existieren die folgenden Primitiv-Kategorien (für die jeweils enthaltenen Primitive siehe Tab. 3.2):

- **Quell-Primitive**: Diese Primitive besitzen keinen Eingang und sind hauptsächlich Generatoren wie Zeit-, Impuls-, Nullwertgeneratoren oder Konstanten und Rampenfunktionen.
- **Senken-Primitive**: Diese verarbeiten Daten zu einem Endprodukt, schreiben sie in Dateien, generieren Diagramme, Bildschirmausgaben oder Töne.
- **Steuerungs-Primitive**: Hiermit werden Datenflüsse manipuliert, also verworfen, aufgeteilt, zusammengeführt etc.
- **Konverter-Primitive**: Umformungen zwischen Datentypen können mit diesen Primitiven durchgeführt werden.
- **Queues, Server, Delays**: Hiermit können Verarbeitungen beeinflusst werden, etwa durch Stacks, Verzögerungen oder Warteschlangen.
- **Timing-Primitive**: Betrachtungen in simulierter oder Echtzeit können mit Timing-Primitiven realisiert werden.
- **Netzwerk-Primitive**: Sie dienen zur Kommunikation zwischen Modellen auf Basis der Netzwerktechnik.

Datentyp	Datenstruktur	Name des Datentyps
beliebig	skalar	`anytype`
Gleitkomma	skalar	`float`
Gleitkomma	Matrix	`float_matrix_env`
komplex	skalar	`complex`
komplex	Matrix	`complex_matrix_env`
Integer	skalar	`int`
Integer	Matrix	`int_matrix_env`
fest	skalar	`fixed`
fest	Matrix	`int_matrix_env`
Nachricht	skalar	`message`
Zeichenkette	skalar	`string`
Datei	skalar	`file`
Datenstruktur	skalar	`datastruct:`

Tabelle 3.1: Parameter-Datentypen in MLDesigner (nach [MLDesign 2004c, Tab. 3.3])

Primitiv-Kategorie	Namen der enthaltenen Primitive
Quell-Primitive	*Clock, Impulse, Null, Poisson, PulseGen, TclScript, TkButtons, TkSlider, Const, Ramp, RanGen, singen, WaveForm*
Senken-Primitive	*BarGraph, Printer, Xhistogram, XMgraph TclScript, TkBarGraph, TkMeter, TkPlot TkShowEvents, TkShowValues, TkStripChart, TkText, TkXYPlot, Beep*
Steuerungs-Primitive	*Discard, Fork, LossyInput, Merge, PassGate, Router, Sampler, LeakBucket, Case, EndCase*
Konverter-Primitive	*Packetize, UnPacketize, MxToImage, ImageToMx*
Queues, Server, Delays	*Delay, VarDelay, PSServer, Server, VarServer, FIFOQueue, FlushQueue, PriorityQueue, Stack, QueueBase*
Timing-Primitive	*MeasureDelay, MeasureInterval, StopTimer, Timeout, TimeStamp, Synchronize, Timer*
Netzwerk-Primitive	*CellLoad, CellUnload, ImageToCell, CellToImage, CellRoute, PriorityCheck, Switch4x4, VirtClock, PCMVoiceRecovery, SeqATMSub, SeqATMZero, Ether, EtherRec, EtherRecMes, EtherSend*
Logik-Primitive	*Logic, TestLevel, FlipFlop{JK, T, D}*
Weitere Primitive	*Arbitrate, HandShake, handshakeQ, TclScript, Statistics, UDCounter*

Tabelle 3.2: MLDesigner-Primitive der DE-Domäne

- **Logik-Primitive**: Hiermit können Logik-Operationen durchgeführt werden. Die Eingänge werden als *Boolean-Variablen* interpretiert.

- **Weitere Primitive**: Zusätzlich existieren Primitive zur Hardware-Modellierung, etwa des Handshake-Verfahrens, oder Statistik- und Monitoring-Primitive.

3.4 Einsatz des MLDesigner in der Medizinischen Informatik

Der MLDesigner als Modellierungs- und Simulationswerkzeug auf systemtheoretischer Basis setzt voraus, dass sich mit dieser Software abzubildende reale Systeme oder Prozesse regelungstechnisch beschreiben lassen. In der Medizinischen Informatik ist dies insbesondere für klinische Prozesse als abgrenzbare Menge aufeinander bezogener Operationen von sozialen Einheiten und technischen Einrichtungen der Fall, welche sich daher als sozio-technische Systeme bezeichnen lassen [Wollnik 1990, S. 462].

Ein Beispiel für eine allgemeine Abstraktion realer administrativer Prozesse im Kranken-

haus als hierarchisch strukturierter Verband von Tätigkeitsfolgen ist das *Siemens Process Framework* [Holzner u. Fleischer 2004]. Dieses Rahmenwerk kann etwa verwendet werden, um einen konkreten klinischen Prozess als sozio-technisches System zu modellieren. Dieses Modell wiederum kann dann am Rechner mit Hilfe des MLDesigner abgebildet werden. Das mehrfache „Durchspielen“ des Modells unter Einspeisung diskret-stochastischer Eingangssignale und Parameter sowie die darauf folgende Auswertung der Kumulation der erhaltenen Ausgangsgrößen, also die Prozesssimulation, bietet dann die Möglichkeit eines Qualitätsmanagement des zugrunde liegenden realen Vorgangs. An der Technischen Universität Ilmenau ist dieses Vorgehen gewählt worden, um den konkreten klinischen Prozess der Therapie des akuten Koronarsyndroms mit Hilfe des MLDesigner zu modellieren und zu simulieren [Eisentraut 2004]. Als Ergebnis konnten dabei z. B. die Wartezeit der Patienten als lebenskritischer Faktor und die Herzkatheteruntersuchung als (warte-)zeitintensivster Prozessschritt identifiziert werden, woraus sich wiederum Folgen für die Qualitätssicherung bei dem realen Prozess ergeben [Eisentraut 2004, S. 54–57].

Die Bedeutung des MLDesigner als Modellierungs- und Simulationssoftware für klinische Behandlungspfade oder konkrete klinische Abläufe konnte daher in der referenzierten Arbeit herausgestellt werden.

4 Forschungsstand

4.1 Klinisches Prozessmanagement

Um neben dem Einsatz der beiden Modellierungswerkzeuge ARIS Toolset und MLDesigner in der Medizinischen Informatik (Abschnitte 2.5 und 3.4) allgemein die bisherige Forschung in Richtung der hier entwickelten Konzepte aufzuarbeiten, soll repräsentativ mit Projekten zum übergeordneten Thema des Managements klinischer Abläufe begonnen werden.

4.1.1 Modellierung klinischer Prozesse

Die Hintergründe der Entwicklung von Prozessmodellen im Krankenhaus beschreiben [von Eiff u. Ziegenbein 2001a]. Sie kennzeichnen die medizinische Behandlung als Leistungsprozess, bestehend aus spezifischen Tätigkeiten mit korrespondierender Zeitdauer, die durch vorangehende Ereignisse ausgelöst werden. Demnach heben sie die Modellierung dieser Prozesse zum einen als komplexitätsreduzierte Abbildung der Realität, zum anderen aber auch als Vorstellung zukünftiger Idealzustände hervor. Die Aufgabe der Prozessmodellierung als Werkzeug zum Management der Abläufe im Krankenhaus wird hierdurch begründet.

Weiterhin werden Gestaltungsprinzipien der Modellentwicklung und Vorgehensmodelle zur Prozessmodellierung vorgestellt. Es wird dabei explizit auf die Rolle der ereignisgesteuerten Prozesskette eingegangen, welche nach Meinung der Autoren durch ihre Etabliertheit in anderen Branchen sowie durch die transparente Visualisierung von Prozessen und ihre ausreichende Konkretisierung für den Prozessvergleich und die Überführung in die Realwelt einen geeigneten Modelltyp bildet.

So sehen die Autoren in der Modellierung von Leistungsprozessen einen „ersten Schritt im Rahmen eines ganzheitlichen zweckorientierten Prozessmanagements“, der durch einen interaktiven Kommunikationsprozess aller an den Abläufen Beteiligten ergänzt werden muss [von Eiff u. Ziegenbein 2001a, S. 78–80].

Einen anderen Ansatz verfolgen [Măruşter u. Jorna 2005], die eine seltener eingenommene Perspektive vertreten, indem sie die Modellierung klinischer Prozesse mit den Mitteln

des Wissensmanagement durchführen. Auf diese Weise integrieren die Autoren z. B. theoretisches Wissen über logistische Prozesse im Krankenhaus in spezielle Petri-Netze (vgl. Abschnitt 5.1.3), um daraus neues Wissen über konkrete Einzelfälle mit Patienten zu generieren. Das neue Wissen kann dann dazu dienen, Patienten in Klassen einzuteilen, deren Wahrscheinlichkeit für bestimmte Diagnostik- und Therapiemaßnahmen konkrete Werte besitzt, wodurch sich für den Einzelpatienten Vorhersagen über seine Aufenthalte an bestimmten Behandlungsplätzen des Krankenhauses treffen lassen. Diese Erkenntnisse können schließlich für das Workflow-Management im Krankenhaus genutzt werden.

4.1.2 Krankenhausvergleich über Standardprozesse

Die Möglichkeit des Betriebsvergleichs von Krankenhäusern behandelt [Mosa 2001]. Die Verfasserin kennzeichnet diesen als Instrument des Qualitätsmanagements zum Zwecke der Steigerung von Qualität und Wirtschaftlichkeit. Als wesentlichen Punkt der Ermöglichung eines Betriebsvergleichs sieht sie dabei die Bildung von Standardprozessen, welche die berufsgruppenübergreifende Zusammenarbeit und die Leistungstransparenz im Unternehmen Krankenhaus fördern. Auch hier wird die EPK als Standardbeschreibungsmittel genutzt. Die Autorin befürwortet eine Ausdehnung von Standardprozessen auf alle Gesundheitseinrichtungen zur Optimierung der zahlreichen Schnittstellen in der Patientenbehandlung.

4.2 Modellierung und Simulation klinischer Prozesse

4.2.1 Prozessmodellierung zum Management klinischer Abläufe

In einem Beitrag zum Krankenhausmanagement befassen sich [Scheer et al. 1996b] mit der Prozessorientierung im Krankenhaus. Sie schlagen eine Modellierung von Abläufen nach dem ARIS-Konzept vor, um die Geschäftsprozessgestaltung und die Einführung anforderungsgerechter Informationssysteme im Krankenhaus zu ermöglichen. Das ARIS Toolset wird genutzt, um beispielhaft Prozesse, Organisation und Funktionen im Krankenhaus zu modellieren und auszuwerten. Außerdem wird auf die Notwendigkeit integrierter und prozessorientierter Informationssysteme im Krankenhaus eingegangen, das sich einem permanenten Verbesserungszyklus unterwerfen muss.

4.2.2 Einzelprozessmodellierung

An diesem Punkt setzt die Arbeit von [Detschewa 2005] an, in der klinische Behandlungspfade als eine Form von Standardprozessen im Krankenhaus mit Hilfe des ARIS Toolset modelliert werden. Den Hintergrund bildet das *Siemens Process Framework* (siehe Abschnitte 2.5.2 und 3.4). Auch hier wird die EPK als zentrale Modellierungsmöglichkeit genutzt und dadurch eine Funktionsbibliothek etabliert, die anschließend in der Modellierung eines konkreten Prozesses, der Therapie des akuten Koronarsyndroms, genutzt werden konnte.

4.2.3 Einzelprozesssimulation

Ein Beispiel für die Simulation klinischer Prozesse bietet [Eisentraut 2004], die sich ebenfalls mit der Interventionstherapie beim akuten Koronarsyndrom beschäftigt. Ebenso auf Basis des *Siemens Process Framework* wird hier der Prozess mit dem MLDesigner modelliert, um ihn anschließend durch die Anwendung konkreter Parameter (Patientenzahl, Wartezeiten etc.) einer Simulation zu unterziehen. So konnten Aussagen zur Prozessoptimierung getroffen werden, die aus der reinen Modellierung nicht als Ergebnis hervorgegangen wären.

Darüber hinaus konnten ebenfalls an der Technischen Universität Ilmenau mit dem MLDesigner die klinisch-administrativen und logistischen Prozesse einer Tagesklinik in ihrer Gesamtheit modelliert werden, woraus sich ein konsistentes, simulierbares Modell einer solchen Klinik ergab [Kühn 2006]. Dieses Modell ermöglichte dann eine Prüfung und Optimierung aller Prozessabläufe derart, dass die Einrichtung nach Anwendung der vorgeschlagenen Verbesserungen auf Grund verkürzter Wartezeiten ihre Öffnungszeiten um eine Stunde reduzieren konnte.

Außerdem wurde an der Technischen Universität in Braunschweig der sogenannte *Braunschweiger Krankenhaus-Simulator* entwickelt [Bott 2001]. Er nutzt das ebenfalls dort entstandene Modellierungswerkzeug MOSAIK-M (Modellierung, Simulation und Animation von Informations- und Kommunikationssystemen in der Medizin), das der „formalen Beschreibung von Informationssystemen durch Modelle zum Zwecke ihrer simulationsgestützten Analyse in der Virtualität, d. h. im Rechner“ dient, speziell für die Simulation von klinischen Abläufen. Auch in diesem Projekt konnten Schwachstellen und Optimierungspotenziale bei der evaluierenden Anwendung in einem Krankenhaus identifiziert werden,

so etwa der erhöhte Dokumentationsaufwand, spezifische Wartezeiten oder Kollisionen bei der Nutzung von Arbeitsmitteln während der Leistungserbringung.

4.2.4 Klinische Prozessbibliotheken

Eine Vertiefung des Potenzials von Standard-Prozessbibliotheken, die mehrfache Verwendung bei der konkreten Prozessmodellierung finden können, wird in [Ziegenbein 2001] und [von Eiff u. Ziegenbein 2001b] erreicht. Fallklassifikationssysteme wie die ICD-10, Evidenzbasierte Medizin (EBM) sowie klinische Leitlinien und Behandlungspfade werden als referenzielle Instrumente eines klinischen Prozessmanagements identifiziert, welche die Grundlage für Prozessbibliotheken bilden können. Solche Bibliotheken können dann einer individualisierten Verwendung bei der konkreten Prozessabbildung zugeführt werden und damit das klinische Prozessmanagement unterstützen und vereinfachen.

Die Schaffung einer solchen klinischen Prozessbibliothek ist auch an der Technischen Universität Ilmenau Gegenstand der Forschung. Hierbei wird der MLDesigner als Werkzeug verwendet, um seine Stärken als abstraktes, simulationsfähiges Modellierungssystem zu nutzen. Die vorliegende Arbeit soll zum Voranschreiten dieses Projektes beitragen.

4.3 Übertragung von ARIS-Modellen in andere Softwaresysteme

Die Möglichkeit, im ARIS Toolset erstellte Modelle in ein XML-Format zu exportieren (auch als AML für ARIS Markup Language bezeichnet), wurde in einigen Projekten genutzt, um ARIS-Daten in anderen Programmen weiter zu verwenden.

4.3.1 Standardisierung des EPK-Speicherformats

So haben [Mendling u. Nüttgens 2004] ein Verfahren zur Weiterverwendung von mit ARIS erstellten EPK-Modellen vorgestellt. Ihrer Ansicht nach stellt das proprietäre Speicherformat vieler Modellierungswerkzeuge ein Problem für das Geschäftsprozess-Management dar. Aus diesen Gründen haben sie eine auf XML basierende Standardbeschreibungssprache speziell für EPK entwickelt. In ihrem Beitrag wird erläutert, wie im AML-Format vorliegende EPK-Modelle in dieses standardisierte Format für ereignisgesteuerte Prozessketten, die

EPC Markup Language (EPML), umgewandelt werden können. In EPML vorliegende Modelle könnten dann in einem weiteren Schritt zurück in das Speicherformat einer beliebigen anderen Modellierungssoftware transferiert werden, was die Komplexität der Konvertierung zahlreicher Speicherformate in andere Formate durch ein Standard-Zwischenformat erheblich reduzieren würde.

4.3.2 ARIS-Einzelmodellimport in der Bauinformatik

An der Bauhaus-Universität Weimar wurde der Prototyp PEPE (Process Endorsing Prototype Engine) entwickelt, der die Visualisierung mittels Prozessmodellen z. B. zur Planung einer Gebäudebauplanungsphase unterstützt [Vad et al. 2002]. Das in Java programmierte Werkzeug ist Bestandteil eines Vorgehensmodells zur Planung der Bauplanung, das eine Prozessmodellierung der Planungsphase zum Inhalt hat. Der Modellkern wird dabei zunächst mit einem anderen Werkzeug – Microsoft Access oder das ARIS Toolset – erzeugt und danach in PEPE eingelesen und vervollständigt. Daher muss PEPE in der Lage sein, solche Modellkerne zu importieren, was im Falle des ARIS Toolset über den XML-Export geschieht. Die so entstandenen AML-Daten werden über Xerces, einen frei verfügbaren XML-Parser, in PEPE eingelesen und können dann vervollständigt werden. Das Modellierungswerkzeug PEPE wurde außerdem an ein bestehendes Modellverwaltungssystem angepasst, um auf dort vorgehaltene Objekte zugreifen und diese wiederverwenden zu können.

4.3.3 Wiedergabe und Abarbeitung von ARIS-Modellen

Markus Stoy hat in einer Praktikumsarbeit zur objektorientierten Softwaretechnik an der Universität Rostock einen AML-Interpreter entwickelt, der – ebenfalls nach Nutzung der XML-Exportfunktion – ARIS-Geschäftsprozessmodelle ohne die Verwendung von ARIS-Software abarbeiten kann [Stoy 2002], [Büyükyilmaz u. Forbrig 2003]. Der AML-Interpreter ist dabei in der Lage, das im XML-Format vorliegende ARIS-Modell grafisch darzustellen, außerdem erzeugt er eine Baumansicht mit allen Modellelementen sowie ein Protokoll der aktuellen Aktionen des Programms. Bei der Abarbeitung des Modells werden die Inhalte entsprechender Ereignisse oder Funktionen angezeigt und es wird bei Verzweigungen ein Nutzerdialog zur Entscheidung, welcher Modellzweig durchlaufen werden soll, eingeblendet. So kann der Ablauf eines Geschäftsvorfalls schrittweise nachvollzogen werden, was

einer einfachen Simulation eines ARIS-Modells gleichkommt.

4.3.4 Kommerzielle Werkzeuge zur Modellkonvertierung

Neben diesen wissenschaftlichen Arbeiten existieren noch zwei kommerzielle Werkzeuge, welche ebenfalls das XML-Exportformat von ARIS nutzen, um mit diesem System erstellte Modelle in eine Vielzahl anderer Visualisierungs-, Modellierungs- und Simulationsprogramme zu importieren.

4.3.4.1 BPM-Converter

Hierzu zählt der BPM-Converter [HRW Consulting Factory AG 2006], der beispielsweise auch ARIS-Modelle ins Microsoft-Visio-Format transferieren kann [Reiter 2004, 2005]. Bei diesem Instrument wurde speziell Wert gelegt auf eine methodische Modelltransformation neben der rein technischen XML-Formatumwandlung. Unterstützt wird derzeit die Konvertierung von und in das Format von sechs verschiedenen Werkzeugen:

- ARIS Toolset
- Semtation Semtalk
- Pulinco TopEase
- Microsoft Visio
- Casewise Corporate Modeler
- SAP Solution Manager

4.3.4.2 TOOLBUS

Weiterhin ist die TOOLBUS-Technologie zu nennen, die generell eine Vielzahl Datenformate untereinander umwandeln kann [Reischmann Informatik GmbH 2006]. Im Falle von ARIS ist die Konvertierung in zehn verschiedene CASE-Tools möglich:

- AllFusion ERwin
- case/4/0

- ER/Studio
- Oracle Designer
- PowerDesigner / PowerAMC
- Rational Rose
- IBM Rational Software Modeler (RSM) / Architect (RSA)
- Select Component Architect (Select Enterprise)
- System Architect
- ASG Rochade

Insgesamt umfasst TOOLBUS derzeit aber die Umwandlung von und in 34 unterschiedliche Speicherformate. Allgemein handelt es sich bei dieser im Jahre 1986 entwickelten Technologie um die Realisierung einer homogenen Methode zur Konvertierung von Metadaten beliebiger Art.

4.3.5 Fazit

Die Bestrebungen zum einen in die Richtung der Konvertierung des ARIS-Toolset-Formats für EPK in andere Formate und zum anderen auf eine Standardisierung für Prozessmodelle in offene, XML-basierte Formate sind, wie das vorliegende Kapitel aufzeigt, beide weit gediehen und resultierten in konkreten Ergebnissen, lauffähigen multifunktionalen Konvertern und vorgeschlagenen Standardformaten für EPK. Offensichtlich wird dabei die bisher fehlende Anbindung des MLDesigner als auf systemtheoretischer Basis arbeitendes, aber problemlos in der Prozessmodellierung und -simulation einsetzbares Werkzeug. Die Aufgabe der vorliegenden Arbeit definiert sich aus diesem Grunde in der Nutzung der bestehenden und referierten allgemeinen Erkenntnisse für die Umwandlung von Prozessmodellen für die hier durchzuführende Konzipierung und Realisierung eines Konverters, der ARIS-Toolset-Modelle in das MLDesigner-Format transferiert.

5 ARIS-Modelle im MLDesigner – Konzeptmodellierung

5.1 Vergleich ARIS – MLDesigner

Nach der Erläuterung von Prinzipien und Strukturen der beiden Modellierungswerkzeuge ARIS Toolset und MLDesigner sowie einer spezifischen Analyse der jeweiligen Modellierungselemente in den beiden vorangehenden Kapiteln und der Aufarbeitung des Forschungsstandes zur hier behandelten Thematik kann in den folgenden Abschnitten ein genauer Vergleich zwischen den Funktionen beider Anwendungen erfolgen. Aus dem Ergebnis dieses Vergleichs ableitend können dann die Umsetzbarkeit notwendiger Strukturen aus ARIS in das MLDesigner-Format betrachtet und die Möglichkeiten einer adäquaten Beschreibung sowohl der einzelnen Elemente als auch kompletter Modelle des ARIS Toolset mit dem Mitteln des MLDesigner aufgezeigt werden.

5.1.1 Vergleich der Modellierungswerkzeuge

Das ARIS Toolset ist eine betriebswirtschaftliche Anwendungssoftware zur Modellierung von Geschäftsprozessen. Es basiert auf der Architektur integrierter Informationssysteme, einem Informationsmodell aus der Wirtschaftsinformatik, das komplexe Unternehmensabläufe veranschaulichend abbilden kann, indem es den Blick aus fünf verschiedenen Perspektiven ermöglicht, die jeweils eigene Schwerpunkte in der Darstellung besitzen (vgl. Abschnitt 2.3). Für die Modellierung unter verschiedenen Sichten ist das ARIS Toolset mit einer Vielzahl unterschiedlicher Modelltypen inklusive deren jeweiliger Modellierungselemente ausgestattet.

Die Modellierungs- und Simulationssoftware MLDesigner dagegen ist ein Werkzeug, mit dessen Hilfe sich komplexe Systeme oder Prozesse beliebiger Art auf der Basis systemtheoretischer Betrachtungen zum einen modellieren, zum anderen durch die Implementierungsmöglichkeit mit eigenem Programmcode und spezifischen Parametern auch simulieren lassen

(siehe Abschnitt 3.2). Der MLDesigner bietet hierfür zwei verschiedene Modellelemente an, Module und Primitive, wobei sich jedes erstellte Modell als Baum darstellen lässt, dessen Blätter aus Primitiven, die restlichen Elemente dagegen aus Modulen bestehen.

Die konkrete Gegenüberstellung der beiden beschriebenen Modellierungswerkzeuge erfolgt tabellarisch (siehe Tab. 5.1).

	ARIS Toolset	**MLDesigner**
Ursprung	Wirtschaftsinformatik	System- und Steuerungstheorie
Modellierungsbasis	ARIS-Informationsmodell	Systemtheorie
Modellierungsobjekte	Unternehmensprozesse	komplexe Systeme und Prozesse beliebiger Art
Abbildungsform	mehr als 150 verschiedene Modelltypen, korrespondierend zu den fünf Sichten des ARIS-Hauses	Modellierung in Datenflussdomänen als vernetztes Blockdiagramm bestehend aus Modulen und Primitiven
Abbildungszweck	Modellierung zur grafischen Veranschaulichung, Etablierung geordneter hierarchischer Abläufe	hierarchische Modellierung zur Abbildung von Architektur und Funktion, Simulation zur Verifikation der Modelle und der zugrunde liegenden Realität
Modellierungs-Interface	ARIS Designer mit Modellierungsfläche, Elementauswahl und Parametereditor	MLDesigner-GUI mit Modellierungsfläche, Parametereditor, PTCL-Kommandokonsole und Darstellungsmöglichkeiten der Simulationsergebnisse
Modellnavigation	ARIS Explorer mit Ordnerstruktur	Tree View mit Librarys
Modellspeicherung	netzwerkfähige Datenbank	exportfähige Librarys
Anwendungstyp	Client/Server-Software	Single-User-Software
Betriebssystem	Windows	Linux / Solaris

Tabelle 5.1: Gegenüberstellung ARIS Toolset – MLDesigner

Hierbei wird deutlich, dass der Einsatzzweck des ARIS Toolset und jener des MLDesigner in wesentlichen Punkten differieren. So bildet das ARIS Toolset mit seinen auf kontextueller Ebene angesiedelten Modelltypen (abgeleitet aus dem Informationsmodell „ARIS-Haus“) ein System zur Modellierung aus dem Grunde der Verdeutlichung, der Komplexitätsreduktion, der reinen Visualisierung und aus Planungs- und Managementzwecken. Der MLDesigner hingegen ermöglicht auf einer systemtheoretischen Basis – der denkbar einfachste Ansatz, nämlich die Kategorisierung der Umwelt in Systeme und diese in Subsysteme – weitaus abstraktere und vom Funktionsumfang her mächtigere Modellierungsmöglichkei-

ten. Das schließt auch die Modellierung von Funktion mit der Hinterlegung von Modellen mit eigenem Programmcode und dadurch deren mögliche Simulation ein. In einer an der Technischen Universität Ilmenau durchgeführten Studie wurden ebenfalls das ARIS Toolset und der MLDesigner gegenübergestellt und die Möglichkeiten des MLDesigner als Simulationswerkzeug gegenüber des stark überwiegenden Darstellungszwecks bei ARIS identifiziert [Tröbs 2006, Kap. 6]. Hierin liegt demzufolge eine Motivation für die vorliegende Arbeit, die automatisierte Konvertierung von ARIS-Toolset-Modellen in das MLDesigner-Format, um sie mit den dortigen Mitteln einer Weiterverarbeitung und Simulation zu unterziehen.

5.1.2 Analyse der ARIS-Modelltypen

Die Vielfalt der Modellierungsmöglichkeiten mit dem ARIS Toolset erfordert eine genaue Analyse, welche Modelltypen eine Umsetzung in MLDesigner erfahren können und sollen. Dafür können zwei Kriterien aufgestellt werden:

1. Die prinzipielle Konvertierbarkeit eines ARIS-Modelltyps.
2. Eine sinnvolle Umsetzbarkeit in ein MLDesigner-behandelbares Problem.

Das Kriterium 2 lässt sich dabei weiter verfeinern, um „sinnvolle Umsetzbarkeit“ zu definieren:

Da im MLDesigner komplexe Systeme oder Prozesse modelliert und simuliert werden, sollten zu konvertierende Modelltypen ebenfalls simulierbare Systeme oder Prozesse abbilden. Weil mit dem ARIS Toolset ursprünglich Unternehmensprozesse näher betrachtet werden sollen, umfassen die Darstellungsmöglichkeiten komplexer Systeme hauptsächlich Strukturen aus dem Geschäftsumfeld, die einer grafischen Ordnung und Veranschaulichung bedürfen. Dazu zählen:

- Organigramme
- Entity-Relationship-Modelle
- Funktions- und Leistungsbäume

Diese Modelle sind einer Simulation unzugänglich, denn ihre Funktion ist mit der korrekten Abbildung der realen Strukturen erschöpfend beschrieben. Es erscheint also nicht sinnvoll,

z. B. Organigramme mit dem Mitteln des MLDesigner zu modellieren. Solche Darstellungsformen würden dem zweiten Kriterium zur Umsetzung in das MLDesigner-Format also nicht genügen.

Die zweite Kategorie von Modellen, welche auf eine Konvertierbarkeit hin zu prüfen ist, umfasst die Abbildungen von Prozessen aller Arten, die in ARIS vorgesehen sind. Hierfür ist ein Blick auf das ARIS-Haus vorzunehmen, um die Modellierungsmöglichkeiten für Prozesse im Unternehmen auszumachen (siehe Abb. 5.1). Während die im vorigen Absatz betrachteten Modelltypen exemplarische Repräsentanten der statischen Organisations-, Daten-, Funktions- und der Leistungssicht sind, findet die eigentliche Prozessmodellierung mit den Mitteln der dynamischen Steuerungssicht, die auch als Prozesssicht bezeichnet wird [IDS Scheer AG 2000, S. 6, 72], statt.

Unterstützend wirkt dabei die Tatsache, dass zum einen das Ziel der vorliegenden Arbeit ist, Modelle klinischer Prozesse von ARIS-Format in den MLDesigner zu übertragen, zum anderen erfolgte eine solche Modellierung klinischer Prozesse in ARIS bereits mit den Mitteln der Steuerungssicht in [Detschewa 2005].

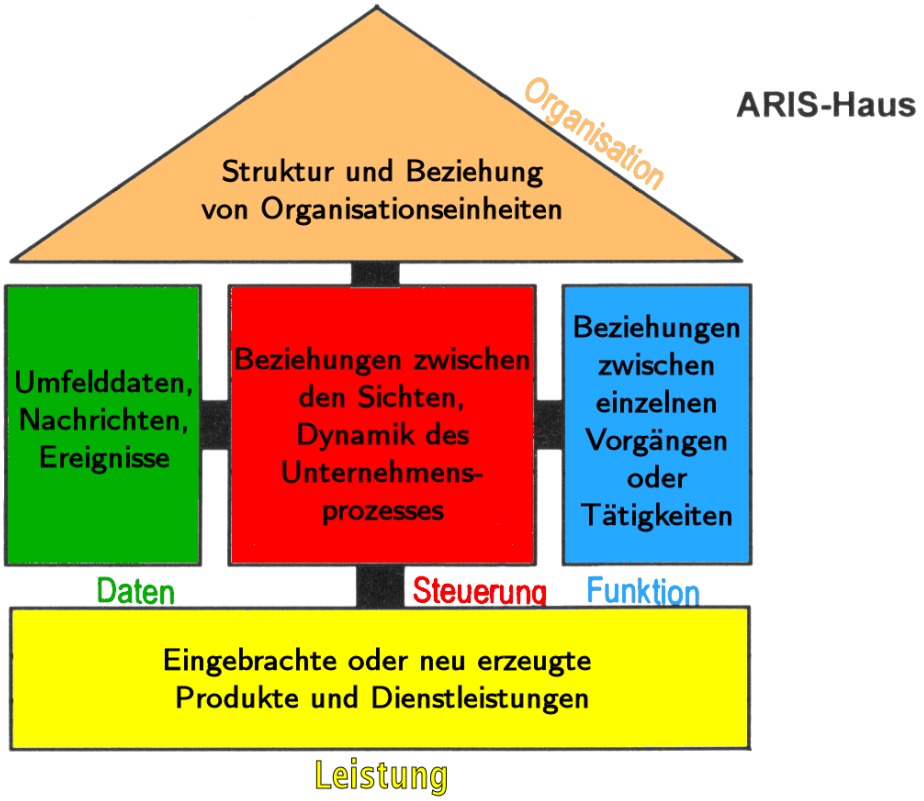

Abbildung 5.1: ARIS-Haus mit Erläuterung der Sichten

Modelltyp	ARIS-Sicht	Prozessdarstellung	Prozessabbildung	Systeme und Daten	Geschäftsprozessmodellierung
Attributzuordnungsdiagramm	Daten			×	×
eBusiness-Szenario-Diagramm	Prozess				×
eERM	Daten			×	×
eERM-Attributzuordnungsdiagramm	Daten			×	×
eEPK	Prozess	×	×	×	×
Fachbegriffsmodell	Daten	×	×	×	×
Funktionsbaum	Funktion		×	×	×
Funktionszuordnungsdiagramm	Prozess	×	×	×	×
Klassendiagramm	Daten				×
Leistungsbaum	Leistung		×	×	×
Office / Industrial Process	Prozess			×	×
Organigramm	Organisation	×	×	×	×
Prozessauswahlmatrix	Prozess		×	×	×
Relationsdiagramm	Daten			×	
Vorgangskettendiagramm	Prozess	×	×	×	×
Wertschöpfungskettendiagramm	Prozess	×	×	×	×
Zieldiagramm	Funktion			×	×

Tabelle 5.2: Klassifizierung von ARIS-Modelltypen (nach [Davis 2001, Tab. 20.2])

Die sinnvolle Umsetzung von Modellen betrifft also solche Modelltypen, mit denen Prozesse abgebildet werden und die der Perspektive der Steuerungssicht angehörig sind (vgl. Tab. 5.2, die sich an vorhandenen Methodenfiltern orientiert). Damit lassen sich folgende spezifischen Modelltypen als Kandidaten für die Konvertierung identifizieren [IDS Scheer AG 2000, S. 8], [Davis 2001, S. 485]:

- Wertschöpfungskettendiagramme
- Vorgangskettendiagramme
- erweiterte ereignisgesteuerte Prozessketten (eEPK)

Die weiteren Modelltypen der Steuerungssicht gestatten nur statische Blicke auf Unternehmenszusammenhänge (Prozessauswahlmatrix, Funktions- und Leistungszuordnungsdia-

gramm) und / oder sind aufgrund ihres Zwecks als reine Veranschaulichung nicht simulierbar (Office bzw. Industrial Process) [IDS Scheer AG 2000, S. 170].

Die hier mit Hilfe der Kriterien zur Umsetzung in MLDesigner-Modelle ermittelten Modelltypen wurden bereits genutzt, um das *Siemens Process Framework*, eine Abstraktion konkreter Abläufe im Krankenhaus, in ARIS abzubilden [Detschewa 2005].

Da die Objekte der Wertschöpfungsketten und der Vorgangskettendiagramme eine Untermenge der Modellierungselemente einer EPK sind (siehe Abschnitt 5.1.3), können die Betrachtungen im Folgenden auf ereignisgesteuerte Prozessketten beschränkt werden, da hiermit gleichzeitig die Modellierungsmöglichkeiten aller anderen für eine Konvertierung in das MLDesigner-Format in Frage kommenden Modelltypen abgedeckt sind.

5.1.3 eEPK versus MLDesigner-Modell

Nachdem in Abschnitt 5.1.1 die eigentlichen Werkzeuge zur Modellierung analysiert und gegenübergestellt wurden, soll nun der wichtigste zu konvertierende Modelltyp näher erläutert und mit der Modellierungsmethode des MLDesigner verglichen werden, um seine prinzipielle Umsetzbarkeit nach Kriterium 1 (siehe Abschnitt 5.1.2) zu untersuchen.

Die *erweiterte ereignisgesteuerte Prozesskette* als zentraler Modelltyp zur Abbildung einzelner Abläufe in einem Unternehmen ist, wie in Abschnitt 2.1.2, Abb. 2.6 und 2.7 dargestellt, ein Netz aus Ereignissen und Funktionen, welche durch Regeln miteinander verknüpft werden können (vgl. auch Abschnitt 2.4) [IDS Scheer AG 2000, S. 95–105]. Diese Modellierungsmethode wurde aus den allgemeineren Petri-Netzen abgeleitet [Chen u. Scheer 1994] und dient primär der prozessorientierten Analyse eines Unternehmens als System [Keller et al. 1992].

Eine ereignisgesteuerte Prozesskette (EPK) entsteht durch das Hintereinanderschalten eines Ereignis-Funktionswechsels, bei dem Ereignisse Funktionen auslösen und Ergebnisse von Funktionen sind. Sie zeigt den logisch-zeitlichen Ablauf eines Geschäftsprozesses [IDS Scheer AG 2005a, S. 4-106].

Beispiele für EPK sind in Abb. 5.2 dargestellt.

Da Ereignisse definieren, durch welchen Zustand oder Bedingung eine Funktion gestartet wird und welcher Zustand den Abschluss einer Funktion definiert, sind Start- und End-

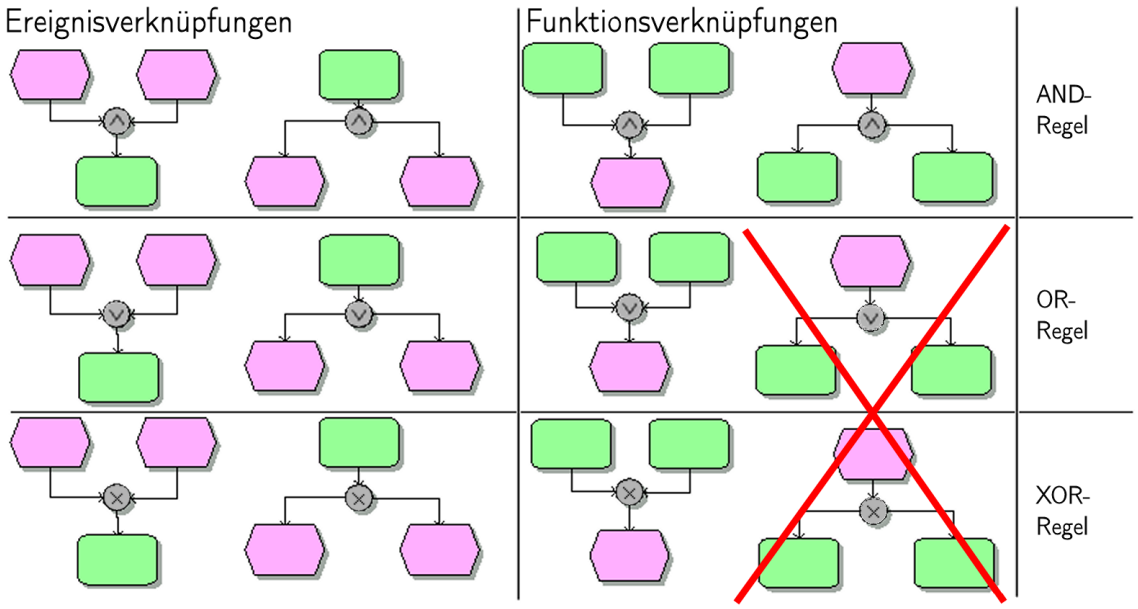

Abbildung 5.2: Regeln in eEPK (nach [IDS Scheer AG 2005a, Abb. 4.4.1-5])

knoten einer EPK immer Ereignisse [IDS Scheer AG 2005a, S. 4-106]. Von ihnen können dann einerseits mehrere Funktionen gleichzeitig ausgehen, andererseits kann eine Funktion mehrere Ereignisse als Ergebnis haben. Diese Verzweigungen werden mit Hilfe einer Regel, also einer logischen Verknüpfung, modelliert.

Grundsätzlich können zwei Arten von Verknüpfungen unterschieden werden [IDS Scheer AG 2005a, S. 4-108–4-110]:

1. *Ereignisverknüpfungen*
2. *Funktionenverknüpfungen*

Zu beachten sind Einschränkungen bezüglich der Funktionenverknüpfungen. Ereignisse können im Gegensatz zu Funktionen keine Entscheidungen treffen, daher ist die Verknüpfung eines auslösenden Ereignisses mit OR- und XOR-Verknüpfungen nicht erlaubt (vgl. Abb. 5.2).

Die Funktionalität von ereignisgesteuerten Prozessketten ist auch durch *Vorgangskettendiagramme* (VKD) abbildbar, wobei die gleichen Elemente verwendet werden, die hier jedoch tabellarisch in Ereignis- und Funktionsspalten gegliedert werden (siehe Abb. 5.3) [IDS Scheer AG 2005a, S. 4-115].

Zusätzliche Elemente aus anderen Sichten, die hier ebenfalls eine Rolle spielen, sind zum einen auch in EPK verwendbar und lassen sich zum anderen bei der Konvertierung wie Ereignisse oder Funktionen behandeln oder entfallen aufgrund fehlender Verbindung zum eigentlichen Prozessmodell, so dass in der Modellierung kein Unterschied zur EPK besteht.

Wertschöpfungskettendiagramme (WKD) dagegen sind im Wesentlichen Funktionsfolgen, bestehen also lediglich aus Hintereinanderschaltungen von Funktionen, um zu spezifizieren, welche davon an einer bestimmten Wertschöpfung im Unternehmen beteiligt sind [IDS Scheer AG 2005a, S. 4-126]. Ein Beispiel für eine Wertschöpfungskette findet sich in Abb. 5.4.

Die Gegenüberstellung von zu konvertierenden Modelltypen und dem MLDesigner-Modellierungsformat kann sich damit auf die wichtigste Prozessabbildungsmethode mit ARIS, die EPK, beschränken.

Ereignisgesteuerte Prozessketten bilden mit Hilfe dreier Objekttypen komplette Abläufe ab und sind hierarchisierbar. Die Vernetzungsmöglichkeiten der Objekte sind auf bestimmte

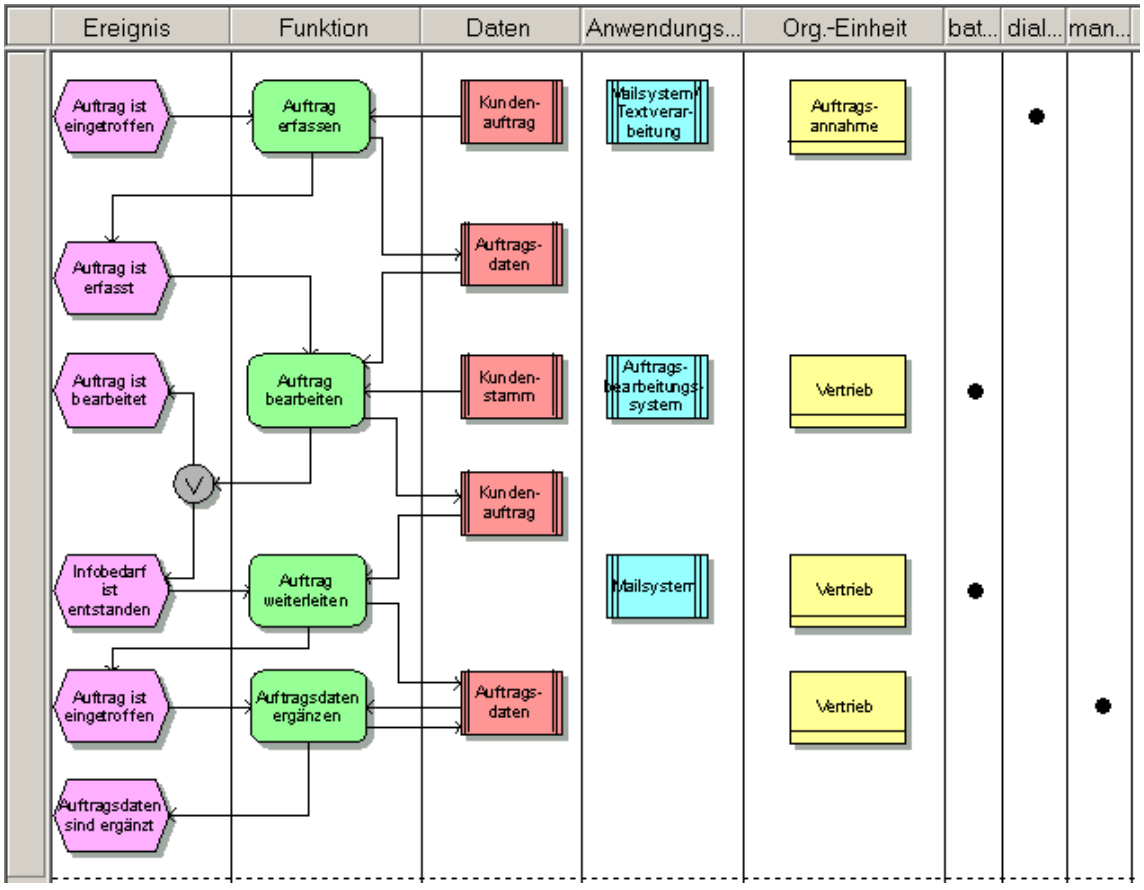

Abbildung 5.3: ARIS-Modelltypen – VKD [IDS Scheer AG 2005a, Abb. 3.2-1]

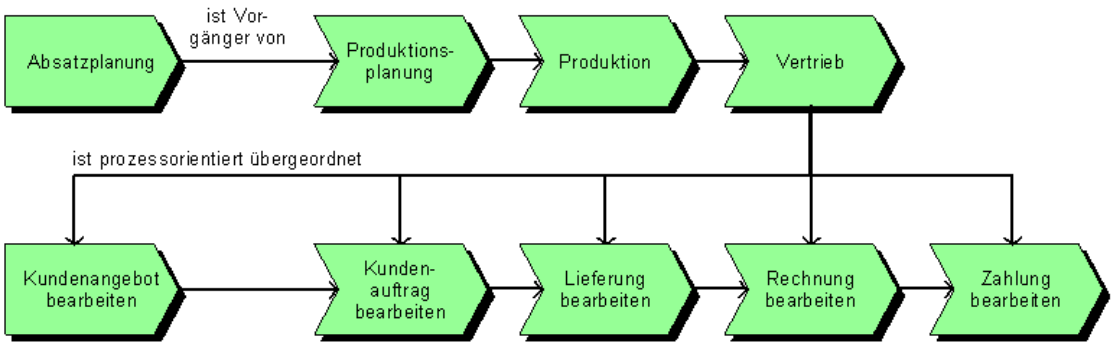

Abbildung 5.4: ARIS-Modelltypen – WKD [IDS Scheer AG 2005a, Abb. 4.4.1-24]

Konstellationen begrenzt (siehe oben, Abb. 5.2) und die Verwendung der Objekttypen ist optional.

MLDesigner-Modelle können ebenfalls zur Modellierung komplexer Prozesse verwendet werden. Es existieren zwei verschiedene Modellelemente, deren Verknüpfung beliebig und optional vorgenommen werden kann; für die Simulation sind gewisse Regeln der Modellierung für ein gültiges MLDesigner-Modell zu beachten [MLDesign 2004c, S. 125–172].

Der nähere Vergleich EPK – MLDesigner-Modell ist in Tab. 5.3 zu finden.

	eEPK	**MLDesigner**
Modellierungsobjekt	Unternehmensprozesse	komplexe Systeme und Prozesse beliebiger Art
Abbildungsform	hierarchisches vernetztes Blockdiagramm	hierarchisches vernetztes Blockdiagramm
Modellierungselemente	Ereignisse, Funktionen, Regeln, Verbindungskanten	Module, Primitive, Verbindungskanten
Modellierungsregeln	vollständige Vernetzung, Verbot von auf ein Ereignis folgenden OR- / XOR-Regeln	vollständige Vernetzung, korrekte Datenflüsse

Tabelle 5.3: Gegenüberstellung eEPK – MLDesigner-Modell

Es zeigt sich, dass die Modellierungsmöglichkeiten mit EPK in allen Fällen eine Untermenge der Fähigkeiten des MLDesigner sind. Damit ist eine prinzipielle Umsetzbarkeit von ereignisgesteuerten Prozessketten (und dadurch auch aller anderen Prozessabbildungstypen von ARIS, die auf den gleichen Prinzipien und Objekten basieren) im MLDesigner gegeben. Zu beachten ist aber, dass die *kontextuelle Information* ereignisgesteuerter Prozessketten, z. B.

die Definition und Eigenschaften eines „Ereignisses“, nicht automatisiert in den MLDesigner übertragbar sind. Die Abbildung dieser informalen Daten bleibt dem Benutzer mit der Erzeugung einer Funktionalität des MLDesigner-Modells im Sinne seines Programmcodes vorbehalten. Im nächsten Schritt ist nun zu eruieren, welche Möglichkeiten zur Konvertierung einzelner Objekte eines ARIS-Modells in MLDesigner-Modellelemente bestehen.

5.2 Analyse einer Elementumsetzung

Da wie in Abschnitt 2.4 erläutert sich die Objekttypen einer EPK im Wesentlichen durch ihre grafische Darstellung, die Regeln ihrer Vernetzung und den Inhalt ihrer Attribute unterscheiden, sollen in den folgenden Betrachtungen nur die primären Objekte Ereignis, Funktion und Regel eine Rolle spielen, deren Umsetzung in MLDesigner-Modellelemente analysiert wird. Alle weiteren Objekttypen können ohne Weiteres die gleiche Behandlung in der Konvertierung erfahren, wie sie bei Ereignissen und Funktionen vorgeschlagen wird. Im Allgemeinen lässt die abstraktere Modellierungsvariante des MLDesigner wie in den vorangegangenen Untersuchungen gezeigt die Umsetzung beliebiger konkreter Objekttypen aus dem ARIS Toolset zu.

5.2.1 Ereignisse

Der erste umzusetzende Objekttyp ist das Ereignis. In Anbetracht seiner Rolle in EPK (vgl. Abschnitt 5.1.3), sowie seiner Eigenschaften und Anwendungsregeln (Abschnitt 5.2.1) wird vorgeschlagen, Ereignisse als Module abzubilden. Dieser Vorschlag erfährt folgende Begründungen:

- Ereignisse und Module lassen sich äquivalent als Elemente eines Blockdiagramms bezeichnen.
- Sowohl Ereignisse als auch Module können Untermodelle beliebigen Typs enthalten.
- Ereignissen und Modulen fehlen die Grundlagen zur Simulation; sie wird ausschließlich in MLDesigner-Primitiven realisiert.

5.2.2 Funktionen

Als nächster Objekttyp muss für die Funktion ein korrespondierendes MLDesigner-Element gefunden werden. Aufgrund der Ähnlichkeit von Ereignissen und Funktionen und der Tatsache, dass es sich bei keinem dieser Objekttypen um eine speziellere oder allgemeinere Variante von ARIS-Modellelementen handelt (siehe auch Abschnitt 5.2.2), wird auch für Funktionen die Umsetzung als Modul vorgeschlagen. Damit gelten die gleichen Argumente wie im vorigen Abschnitt; des Weiteren wird die gleichrangige Vernetzung von Ereignissen und Funktionen im MLDesigner durch ein Netz von Modulen adäquat abgebildet.

Eine nähere Behandlung von Vernetzungsregeln bei Ereignissen und Funktionen ist im MLDesigner nicht notwendig, da die Korrektheit dieser Verbindungen bereits im ARIS Toolset geprüft wird; korrekt modellierte ARIS-Modelle können demnach durch die Transformation von Ereignissen und Funktionen sowie weiteren Objekttypen in miteinander verbundene Module des MLDesigner keine ungültige Verschaltung erfahren.

5.2.3 Regeln

Eine Sonderstellung unter den Objekttypen des ARIS Toolset nehmen einzig die Regeln ein, welche logische Verknüpfungen der anderen Objekte vornehmen. In diesem Sinne sind Regeln nicht nur für die grafische Veranschaulichung und Abstraktion zuständig, sie kontrollieren auch eventuelle Datenflüsse bzw. treffen Entscheidungen über zu durchlaufende oder zu vermeidende Prozessabschnitte. Diese Funktionalität muss in einem MLDesigner-Modell, das einen ARIS-Modelltyp korrekt repräsentieren soll, ebenfalls verfügbar sein. Aufgrund dieses spezielleren Charakters von Regeln im Gegensatz zu anderen Objekttypen des ARIS Toolset wird vorgeschlagen, diese als Primitive umzusetzen.

An dieser Stelle ist zu beachten, dass sich Regeln in ARIS nicht vollkommen konform zu Boole'schen Verknüpfungen verhalten. So ist neben einer OR-Funktion mit zwei Eingängen und einem Ausgang im ARIS Toolset auch problemlos eine OR-Funktion mit einem Eingang und zwei Ausgängen denkbar; diese würde das am Eingang anliegende Signal dann mit einer Wahrscheinlichkeit von jeweils 33,3 % an einen der beiden oder an beide Ausgänge weiterleiten.

Bevor zur Realisierung von AND-, OR- oder XOR-Verknüpfungen von als Modulen umgesetzten Ereignissen oder Funktionen aber eigene Primitive mit selbst implementiertem

Quellcode realisiert werden müssen, sollte untersucht werden, ob die Vielzahl an mitgelieferten Beispielprimitiven des MLDesigner nicht etwa bereits für diesen Zweck einsatzfähige Exemplare enthält. Erinnert werden soll hierbei an die vorgestellten Primitive, speziell die Gruppe der Steuerungsprimitive, das u. a. ein Primitiv *fork* enthält, welches die Daten, die an seinem Eingang anliegen, direkt an seine beiden Ausgänge weiterleitet. Weiterhin existiert das Beispielprimitiv *ProbSwitch*, das Daten an seinem Eingang je nach Wahrscheinlichkeit (die sich justieren lässt), an jeweils einen seiner beiden Ausgänge überträgt. So lassen sich logische Verknüpfungen in einem Datenfluss durch das MLDesigner-Modell zu Simulationszwecken, welcher dann auch das Durchlaufen oder Vermeiden von Modellabschnitten beinhaltet, realisieren (siehe Abschnitt 3.3.4).

Da sowohl das *fork*-Primitiv und der *ProbSwitch* als auch die ARIS-Regeln im äquivalenten Maße die logische Verknüpfung von Modellelementen realisieren, wird vorgeschlagen, Regeln durch eine entsprechend justierte Instanz dieser Bibliotheksprimitive im MLDesigner abzubilden. Damit wird also jede UND-Funktion durch *fork* ersetzt und jede XOR-Funktion durch einen *ProbSwitch*. Für die OR-Funktion existiert derzeit noch kein äquivalentes Beispiel-Primitiv, deshalb wird diese als Modul abgebildet, das der Nutzer dann nach seinen Wünschen bearbeiten kann.

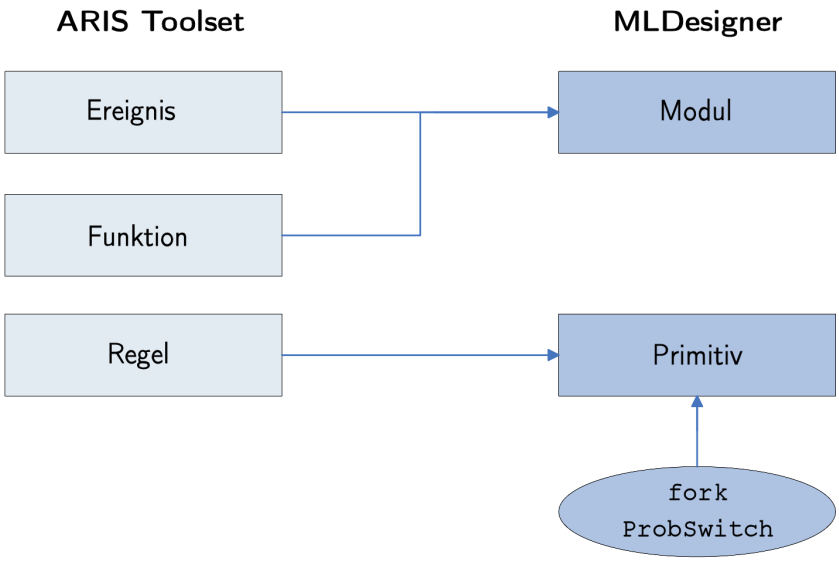

Abbildung 5.5: Schema zur ARIS-Objekttypumsetzung in MLDesigner

Mit der Umsetzung der drei wichtigsten Objekttypen in MLDesigner-Modellelemente, die noch einmal in Abb. 5.5 verdeutlicht und zusammengefasst dargestellt wird, ist die Analyse einer Elementumsetzung abgeschlossen. Ein verbleibender und zugehöriger Aspekt, die Umsetzung von Attributen der Objekte, wird im nächsten Abschnitt erläutert.

5.2.4 Umsetzung von Eigenschaften und Attributen

In diesem gesonderten Abschnitt soll für alle ARIS-Objekttypen gemeinsam der Transfer ihrer Eigenschaften in ihre gewählten Äquivalente im MLDesigner behandelt werden. Eine erste Betrachtung soll die Grundeigenschaften (vgl. Abschnitt 2.4.7) betreffen:

- Der *Name* des Objekts wird als Modul- bzw. Primitivname übernommen.
- *Kanten*, also Verbindungen zu anderen Objekten werden ebenfalls analog übernommen.
- *Hinterlegungen* eines Objekts mit weiteren Modellen werden nicht übernommen; der Nutzer soll damit in die Lage versetzt werden, hinterlegte Modelle je nach Wunsch einzeln zu konvertieren oder von der Konvertierung auszuschließen.
- *Verknüpfungen / OLE-Objekte*, die mit dem Objekt aufgerufen werden können, sind Windows-spezifisch und finden im MLDesigner keine Verwendung.
- Die *Objektdarstellung*, also grafische Optionen wie Textplatzierung, Farbe, Größe etc. werden nicht übernommen, einzig die Objektplatzierung selbst, die für den Modellaufbau relevant ist, soll eine Umsetzung erfahren.

Alle weiteren Attribute im ARIS Toolset sind optionale Objektparameter. Daher erscheint es sinnvoll, das Vorhandensein konkreter weiterer Attribute zunächst zu überprüfen und dann die mit Werten hinterlegten Attribute zu entnehmen, um ein überflüssiges Umsetzen einer großen Anzahl nicht verwendeter Attribute in den MLDesigner zu vermeiden.

Die aufgefundenen und entnommenen verwendeten Attribute können im MLDesigner dann wie in Abschnitt 3.3.3 dargestellt als optionale Parameter realisiert werden, welche im MLDesigner in beliebiger Anzahl definiert werden können. So kann sichergestellt werden, dass vom Nutzer verwendete und festgelegte Eigenschaften eines Objekts in jedem Falle nach der Konvertierung im MLDesigner zur Weiterverwendung ebenfalls vorhanden sind.

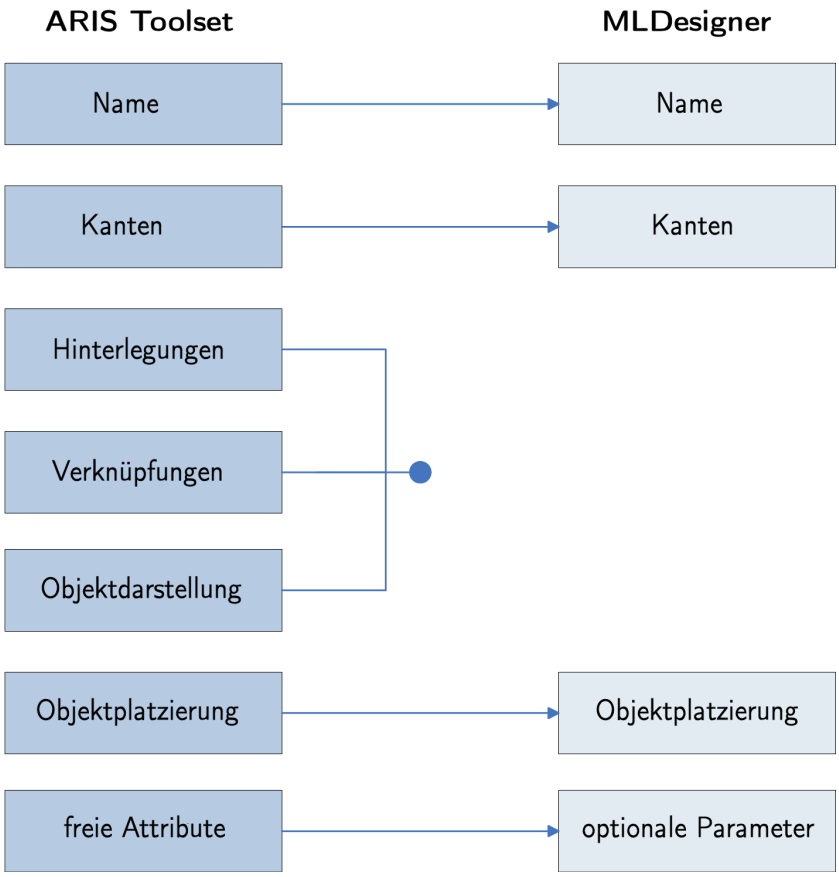

Abbildung 5.6: Schema zur ARIS-Attributumsetzung in MLDesigner

In Abb. 5.6 wird die Umsetzung von Objektattributen in MLDesigner-Parameter, wie sie hier vorgeschlagen wurde, schematisiert.

5.3 Konvertierung von Modellen

Nachdem im vorangehenden Abschnitt erläutert wurde, wie sich einzelne Objekte eines ARIS-Modells allgemein und einer EPK als zentraler Prozessabbildungsmöglichkeit im Besonderen in das MLDesigner-Format transferieren lassen, soll hier die Umsetzbarkeit kompletter bestehender Modelle aus dem ARIS Toolset in den MLDesigner geprüft und eine Lösung erarbeitet werden.

Die Aufgabe der Konvertierung teilt sich dabei in zwei Bereiche, zum einen den Export bestehender Modelle aus dem ARIS Toolset heraus und zum anderen deren Import in den MLDesigner.

5.3.1 Modellexport aus ARIS

Mit dem ARIS Toolset erstellte Modelle werden in einer zentralen Datenbank gespeichert [IDS Scheer AG 2000, S. 9]. Da der Zugriff darauf von Seiten des MLDesigner nicht möglich ist und zudem die Hürde eines jeweils unterschiedlichen Betriebssystems besteht, muss eine Exportfunktion genutzt werden, um einzelne ARIS-Modelle in ein konvertierbares Datenformat zu bringen. Dazu eignet sich der vom ARIS Toolset unterstützte XML-Export, der mit ARIS erstellte Modelle in ein spezielles XML-Format, die ARIS Markup Language (AML), überführt.

Die *Extensible Markup Language* (XML) ist eine betriebssystemübergreifende Auszeichnungssprache [W3C 2004], [Vonhoegen 2005] bzw. Inhaltsbeschreibungssprache [Rothfuss u. Ried 2003] und sowohl maschinen- als auch menschenlesbar.

Mit dem ARIS Toolset wird eine umfassende Dokumentation der XML-Schnittstelle des Werkzeugs geliefert, so dass die Weiterverarbeitung exportierter Dokumente kein Problem darstellt [IDS Scheer AG 2005c].

Beim Export eines beliebigen Modells aus ARIS wird eine XML-Datei erzeugt, die alle Daten der exportierten Elemente enthält. Damit ist der Transfer eines ARIS-Modells, wenn gewünscht unter Auflösung bestehender Modellhierarchien, in eine Textdatei erreicht, die betriebssystemübergreifend genutzt und umgewandelt werden kann (siehe Abb. 5.7).

Diese Datei muss nun in ein MLDesigner-importfähiges Format gebracht werden.

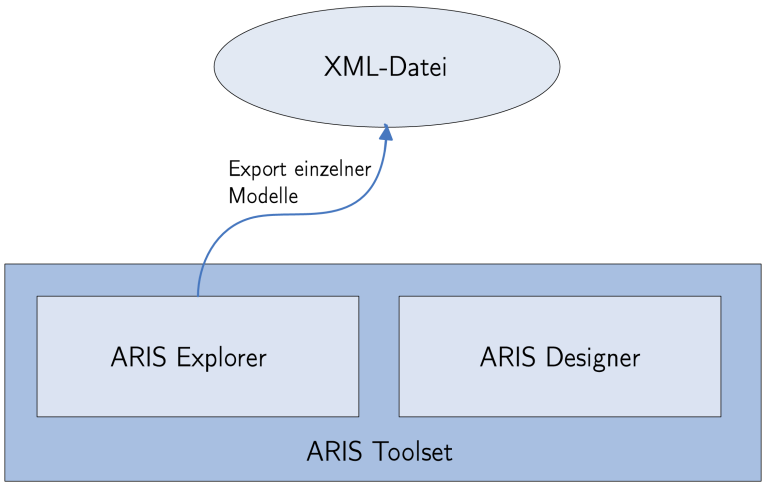

Abbildung 5.7: Schema zum Modellexport aus ARIS Toolset

5.3.2 Modellimport in den MLDesigner

In MLDesigner erstellte Modelle werden in sogenannten Librarys gespeichert. Die Librarys sind auf Betriebssystemebene als eine Menge von Dateien realisiert, welche sich in Unterordnern des MLDesigner-Arbeitsverzeichnisses des aktuellen Benutzers befinden. Diese Dateien mit der Endung MML sind in einem XML-basierten Dokumentenformat abgefasst [Hauguth 2000, S. 8–28]. Damit besteht eine Korrespondenz zu den exportierten ARIS-Modellen im XML-Format (vgl. Abschnitt 5.3.1). Ein Import dieser Modelle in den MLDesigner kann demnach dann erfolgen, wenn es gelingt, die XML-Daten des ARIS-Modells aus der Datei zu extrahieren, in das MML-Format des MLDesigner umzuwandeln und auf die notwendige Dateistruktur einer MLDesigner-Library aufzuteilen. Das hierfür geeignete Werkzeug ist die *Extensible Stylesheet Language for Transformations* (XSLT) [W3C 1999b], [Bongers 2004], eine Programmiersprache, mit der XML-Daten in verschiedene andere Formate, darunter auch erneut XML, umgewandelt werden können.

Eine auf diese Weise gewonnene Verzeichnisstruktur mit Library-Dateien im MML-Format kann dann über die Importfunktion des MLDesigner in dessen Modellverwaltung eingebunden werden (siehe Abb. 5.8) [MLDesign 2004c, S. 12-9 f.].

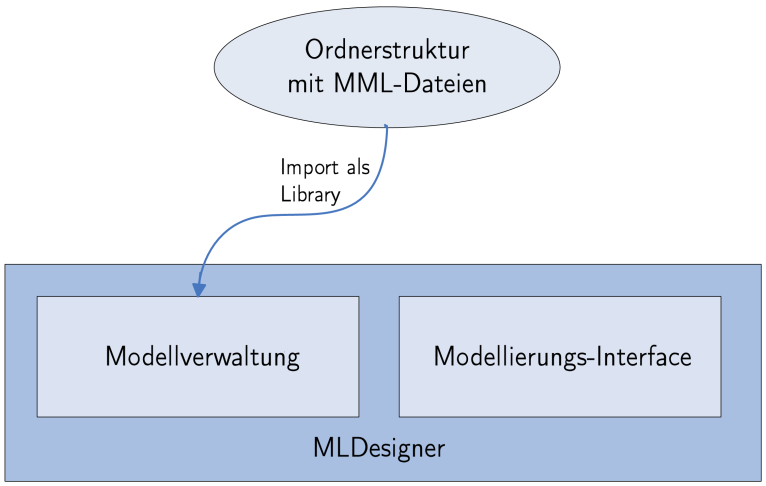

Abbildung 5.8: Schema zum Modellimport in MLDesigner

Damit ist der Konvertierungsvorgang, der in dieser Arbeit konzipiert und realisiert werden soll, komplett beschrieben. ARIS-Modelle werden durch die diesem Werkzeug eigene XML-Exportfunktion in ein anwendungs- und betriebssystemunabhängiges Datenformat

umgewandelt, aus dieser XML-Datei ausgelesen, vom Konverter in das MML-Format transformiert und können schließlich als gültige Library in den MLDesigner importiert werden (siehe Abb. 5.9).

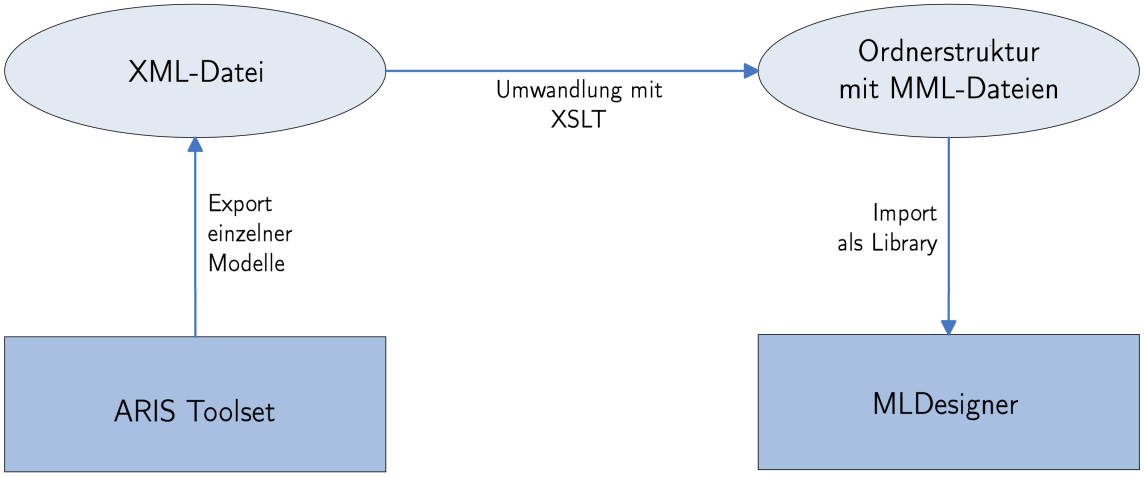

Abbildung 5.9: Schema zur Modellübertragung von ARIS Toolset zu MLDesigner

Mit der Vollendung der Konzeptmodellierung kann die hier beschriebene Modellumsetzung nun einer konkreten Anforderungsanalyse unterworfen werden. Daraus ableiten lassen sich dann formale Spezifikationen für den Modelltransfer, welche schließlich zur Grundlage für die konkrete Programmierung und somit Implementierung eines Konvertierungswerkzeugs dienen können. Diese softwaretechnische begründeten Schritte zur Erstellung eines funktionsfähigen Prototyps sind Bestandteil des Konverter-Designs.

6 Konverter-Design

Mit der abgeschlossenen Analyse der Umsetzungsmöglichkeiten von ARIS-Toolset-Modelltypen in das MLDesigner-Format beginnt die Untersuchung und Realisierung einer Transfermöglichkeit existierender Modelle in den MLDesigner. Das hierfür zu implementierende Werkzeug – ein Modellkonverter – soll im nächsten Schritt mit den etablierten Mitteln der Softwaretechnik entworfen werden [Jackson 1995, S. 169–172, 193–196]. Dazu gehört die Beschreibung der Eigenschaften des Anwendungsbereichs sowie eine Definition notwendiger Begriffe, um die Anforderungen an das Programm zu erarbeiten. Aus diesen Anforderungen können dann implementierbare Spezifikationen abgeleitet werden, die schließlich zum konkreten Programmentwurf führen.

6.1 Eigenschaften des Anwendungsbereichs

Die erste Untersuchung betrifft die Arbeitsumgebung des zu realisierenden Konverters. Sie gliedert sich in zwei Bestandteile, die zum ARIS Toolset bzw. zum MLDesigner gehörig sind.

6.1.1 Anwendungsbereich ARIS Toolset

Das ARIS Toolset als ökonomisch-informationstheoretisches Modellierungswerkzeug unter Windows ermöglicht die Erstellung und Weiterverarbeitung einer Vielzahl von Modelltypen, korrespondierend zu den Sichten des ARIS-Hauses.

Die hier relevante Funktion des datenbankbasierten ARIS Toolset ist die Möglichkeit, einzelne solcher Modelle in Dateien zu exportieren. Für den Export ist dabei das XML-Format wählbar. Auswahl und Export eines speziellen Modells führen dazu, dass die Eigenschaften dieses Modells in einer Einzeldatei im XML-Format vorliegen (siehe Abb. 6.1). Dieses Modellformat ist umfassend dokumentiert [IDS Scheer AG 2005c] und bietet die Möglichkeit zur Weiterverarbeitung mit eigenen XML-fähigen Werkzeugen.

Der erste Teil des Anwendungsbereichs eines zu realisierenden Modellkonverters ist damit

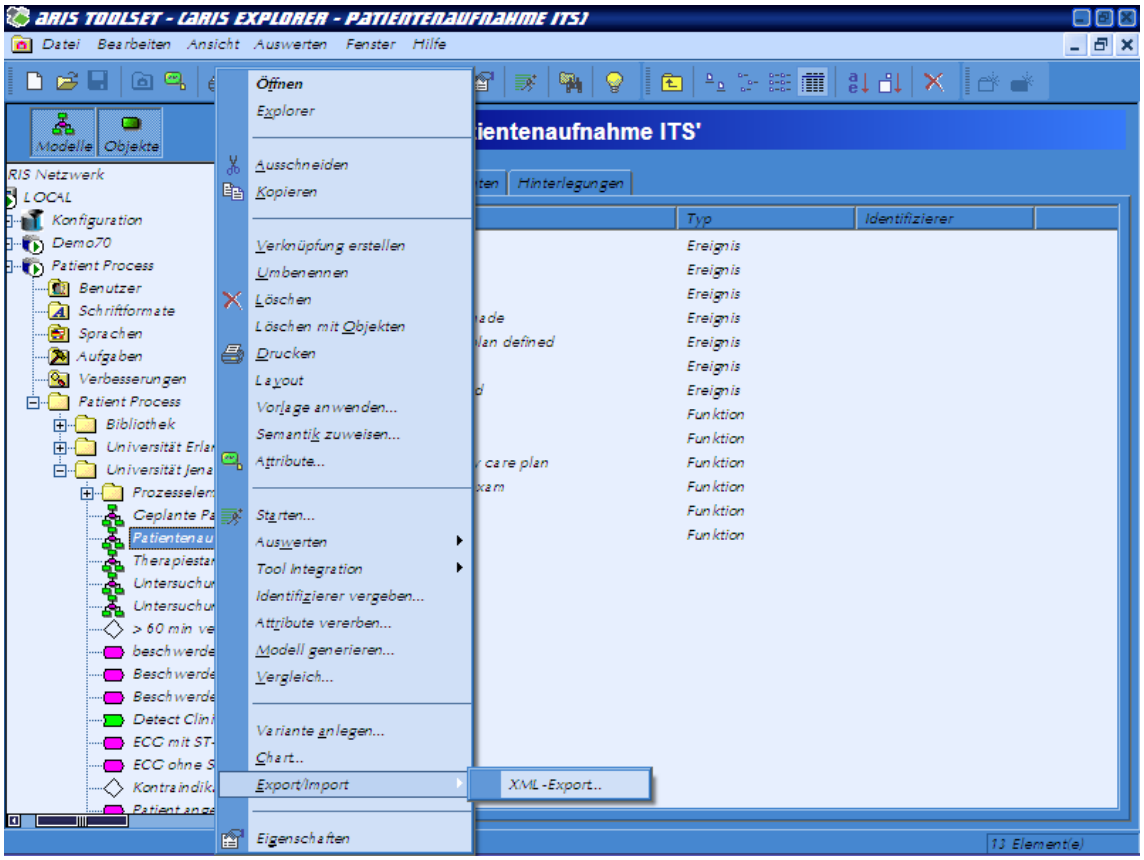

Abbildung 6.1: ARIS Toolset – XML-Exportmöglichkeit

die zugrunde liegende Windows-Modellierungssoftware ARIS Toolset, im Besonderen deren Exportfunktion in das nutzbare Datenformat XML.

6.1.2 Anwendungsbereich MLDesigner

Auf der anderen Seite des Anwendungsbereichs steht die Modellierungs- und Simulationssoftware MLDesigner, die hierarchische Blockdiagramme als Modelltyp verwendet und diese in Librarys verwaltet. Auf Dateiebene werden Librarys in einer Verzeichnisstruktur abgelegt, die verschiedene Dateien im MLDesigner-spezifischen XML-Format – die MML-Dateien [Hauguth 2000] – enthält.

Im Einzelnen ist diese Struktur wie folgt aufgebaut (vgl. Abb. 6.2):

- Ein Verzeichnis mit dem Namen der Library selbst. Es enthält eine MML-Datei, in der im XML-Format Verweise auf alle weiteren Unterordner und die dazugehörigen MML-Dateien der Einzelsysteme /-modelle niedergelegt sind.

- Je ein Unterordner für ein System oder Modell, das in der jeweiligen Library abgespeichert wurde. In diesem findet sich eine MML-Datei mit den entsprechenden Eigenschaften des Modells im XML-Format.

- Je ein Unterordner für jedes einzelne Modul oder Primitiv, das Bestandteil eines der Modelle der Library ist. Auch in diesen Verzeichnissen liegen MML-Dateien vor, welche konkrete Eigenschaften und Parameter des einzelnen Modellelements enthalten.

- Jeder der genannten Ordner enthält noch weitere Dateien: Eine HTML-Datei zur Modelldokumentation, eine PCTL-Datei mit Definitionen zu eigenen Datenstrukturen sowie eventuell Sicherungsdateien. Im Library-Verzeichnis finden sich zusätzlich noch die Datei `.mld.fvx` und eine LVX-Datei, die noch einmal Angaben zu Modell-

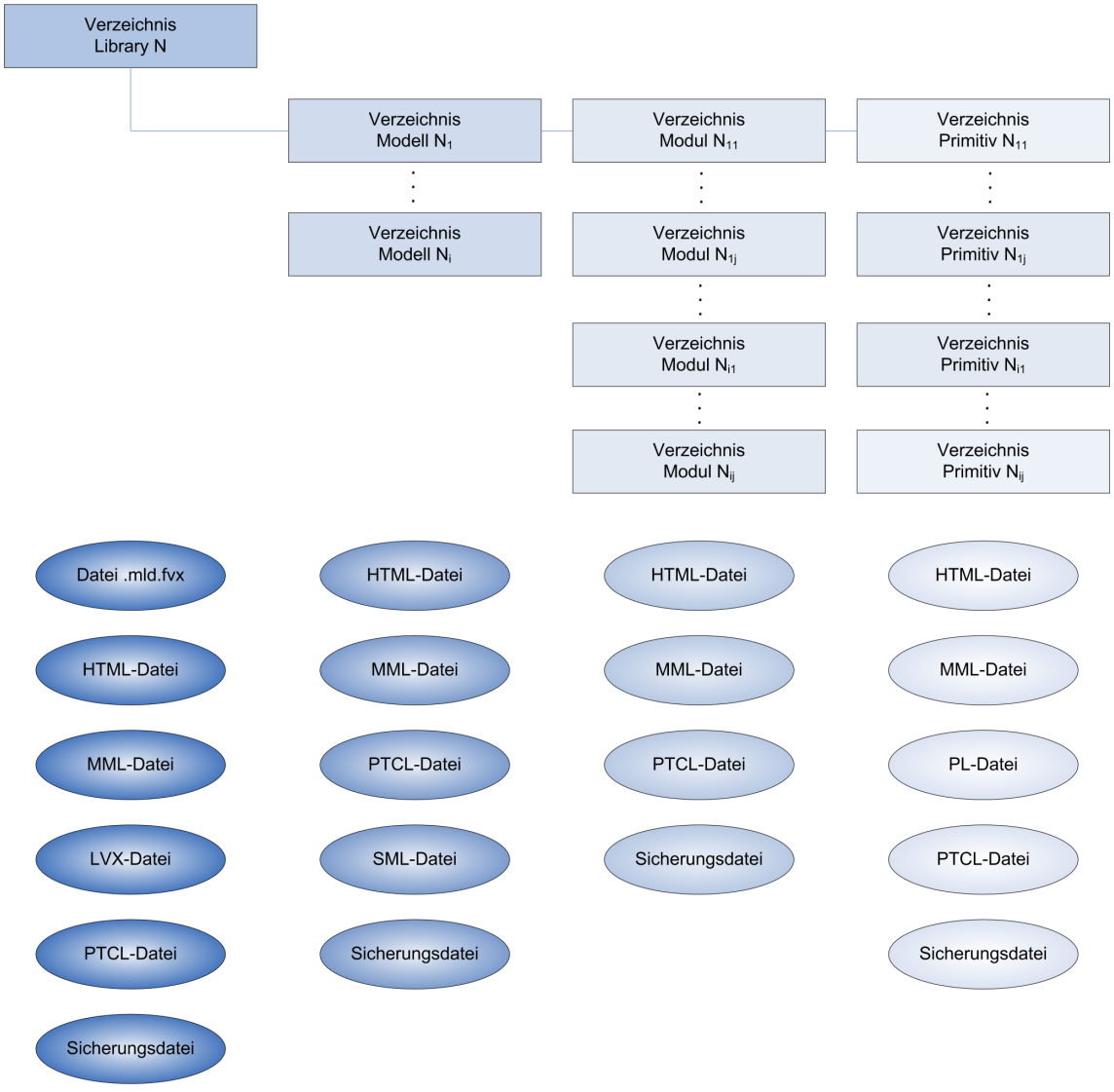

Abbildung 6.2: Schema zur Dateistruktur von MLDesigner-Librarys

dateien und den Namen der Unterordner enthalten. In Modellverzeichnissen finden sich SML-Dateien mit Eigenschaften zur Simulation. Primitiv-Verzeichnisse enthalten PL-Dateien mit dem Quellcode.

Diese Verzeichnisstruktur stellt eine Library im MLDesigner dar, welche in das laufende System und seine Modellverwaltung eingebunden ist. Um in das System fremde, neue Librarys einzubringen, wird diese Ordnerstruktur mit dem UNIX-heimischen TAR-Format in einer einzelnen Container-Datei zusammengefasst, welche hier MLDesigner-spezifisch die Dateiendung MAR besitzt.

Solchermaßen (etwa durch einen früheren Export) kreierte Dateien können dann über die MLDesigner-Importfunktion in die aktuelle Library-Verwaltung eingebunden werden (siehe Abb. 6.3) [MLDesign 2004b].

Der zweite Teil des Konverter-Anwendungsbereichs enthält daher den MLDesigner unter Linux / Solaris und dessen erläutertes Modellspeicherformat bzw. Importformat für die Nutzung der Library-Importfunktion.

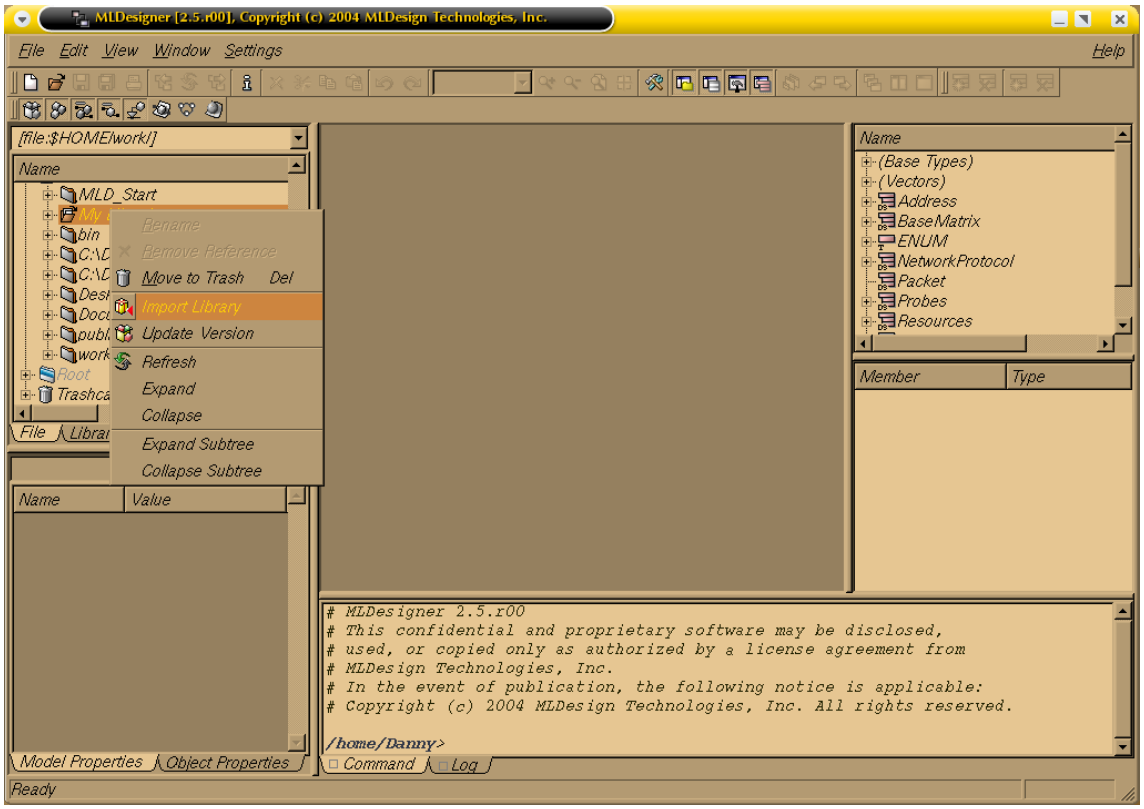

Abbildung 6.3: MLDesigner – Library-Importmöglichkeit

6.2 Begriffe

Für die nachfolgenden Ausführungen werden nun in den nächsten Unterabschnitten die in der Anforderungsanalyse notwendigen Begriffe eingeführt.

6.2.1 Quelldatei

Als Quelldatei werden XML-Dateien bezeichnet, die durch den ARIS-Toolset-Export erzeugt wurden und damit im in [IDS Scheer AG 2005c] beschriebenen Format ein spezielles ARIS-Modell enthalten.

Quelldateien stehen am Beginn des Konvertierungsprozesses und werden betriebssystembedingt auf FAT16-, FAT32- oder NTFS-formatierten Datenträgern gespeichert.

6.2.2 Zieldatei

Die Zieldatei ist eine im MML-Format vorliegende XML-Datei, die Spezifikationen und Eigenschaften eines MLDesigner-Modells enthält.

Der Speicherort einer Zieldatei in einer Verzeichnisstruktur wird in Abschnitt 6.1.2 beschrieben.

Ein vollständiger Verbund solcher Dateien ist, wenn zusätzlich TAR-„gepackt“, in eine laufende MLDesigner-Anwendung importierbar. Dazu muss sie auf einem von Linux oder Solaris lesbaren Datenträger (meist FAT16, FAT32, ext2, ext3 oder ReiserFS) vorliegen.

6.2.3 Konverter

Als Konverter wird das in dieser Arbeit zu entwerfende und zu implementierende Programm bezeichnet, das Quelldateien in den Verbund aus Zieldateien umwandeln kann.

Die Eigenschaften des Konverters werden im Folgenden in der Anforderungsanalyse, den Spezifikationen und dem Systementwurf ermittelt und danach umgesetzt.

6.3 Anforderungsanalyse

In der Anforderungsanalyse soll beschrieben werden, welche neuen Funktionen im Anwendungsbereich verfügbar sind, wenn das zu erstellende Programm implementiert ist und genutzt wird. Die Anforderungen betreffen im vorliegenden Fall drei gewünschte Prozesse, die sich durch den (betriebssystembedingten) Charakter des Datenträgers, auf dem sie ablaufen, auszeichnen: Ein Windows-nativer, ein betriebssystemunabhängiger und ein Linux-spezifischer Anforderungsteil.

6.3.1 Windows-native Anforderungen

Die auf einen vom Betriebssystemtyp Windows beschreibbaren Datenträger vorliegende Quelldatei soll vom Programm eingelesen werden.

Es soll eine Überprüfung des Dateiinhalts vorgenommen werden um festzustellen, ob er ein XML-Dokument im Sinne der in [IDS Scheer AG 2005c] beschriebenen Struktur darstellt. Falls das der Fall ist, so sollen diesem Dokument die für die Umsetzung relevanten Eigenschaften des darin gesicherten Modells, i. e. Elementplatzierung, -name, -parameter, verbindende Kanten und Modellname, entnommen und im Speicher vorgehalten werden.

6.3.2 Betriebssystemunabhängige Anforderungen

Die im Speicher vorliegenden XML-Daten, die spezielle Eigenschaften eines ARIS-Modells enthalten, welche nach der Modellanalyse (siehe Kapitel 5) ein prinzipiell korrektes und sinnvoll einsetzbares MLDesigner-Modell beschreiben können, sollen in ein anderes XML-Format, das MLDesigner-eigene Modellbeschreibungsformat MML, umgewandelt werden. Alle Eigenschaften des Modells, die dem XML-Dokument entnommen wurden, sollen dabei erhalten bleiben. Zum Ende des Umwandlungsprozesses sollen die nun im MML-Format vorliegenden Daten weiterhin im Speicher verfügbar sein.

Hiermit ist ein komplett automatischer Konvertierungsprozess umschrieben, der vom Nutzer keine Eingaben erfordert. Das resultierende Modell muss in jedem Falle im MLDesigner weiter bearbeitet werden (vgl. Abschnitt 6.4.1).

6.3.3 Linux-spezifische Anforderungen

Nach der Umwandlung der im Speicher enthaltenen MML-Daten soll im letzten Konvertierungsschritt eine wie in Abschnitt 6.1.2 beschriebene Verzeichnis- und Dateistruktur hergestellt werden. Die entstehende Ordnerstruktur, nun gefüllt mit den Daten eines ML-Designer-Modells, dessen Eigenschaften aus einem ursprünglich in ARIS erstellten Modell abgeleitet wurden, soll TAR-„komprimiert“ und umbenannt werden, so dass eine MAR-Datei entsteht, welche dann auf einem von Linux lesbaren Datenträger abgespeichert werden kann.

6.4 Ableitung der Spezifikationen

Die in diesem Abschnitt zu entwickelnden, aus den aufgestellten Anforderungen abzuleitenden Spezifikationen des Konverters sollen zur Bewahrung der Übersicht in eine Schichtenarchitektur gegliedert werden [Bass et al. 2002, S. 66–69], die in Abb. 6.4 wiedergegeben ist. Aufgrund einer aufsteigenden Spezialisierung der Aufgaben nach unten hin soll mit der obersten Schicht begonnen werden.

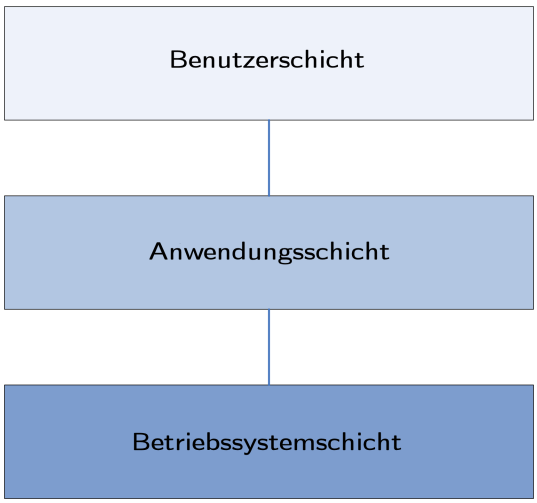

Abbildung 6.4: Konverterspezifikationen – Schichtenarchitektur

6.4.1 Spezifikationen der Benutzerschicht

Die Aufgaben aus der Benutzerschicht des Konverters betreffen die Interaktion des Programms mit dem Nutzer. Hierbei ist folgende Funktionalität zu schaffen:

- Der Nutzer soll zunächst die zu konvertierende Quelldatei auswählen können.
- Das Modell soll jeweils „auf Knopfdruck“ a) in ein MLDesigner-Modell umgewandelt und b) in einer Library abgelegt werden.
- Der Nutzer soll bei Bedarf weitere Modelle auf die genannte Art umwandeln können.
- Zum Beenden des Konvertierungsprozesses sollen alle bis dahin konvertierten Modelle in einer gemeinsamen importfähigen MLDesigner-Library abgelegt werden, deren Speicherort wiederum der Nutzer wählen kann.

Die hier beschriebene Einfachheit des Konverters und des Konvertierungsvorgangs selbst, d. h. ohne weitere Rückfragen etwa bei Konvertierungsoptionen, nicht notwendigen Parametern etc. stellt die Spezifikation einer Anforderung an den Konverter dar, die aus zwei Gründen resultiert: a) besteht die Möglichkeit, dass der Benutzer sich in das ARIS-Toolset-Modell noch nicht eingearbeitet hat und b) ist nicht automatisch davon auszugehen, dass es sich bei Konverternutzer und das Modell im MLDesigner Weiterverarbeitendem um ein und dieselbe Person handelt. Demzufolge erscheint eine möglichst automatische Konvertierung als sinnvollere Variante; jegliche Anpassungen des Modells im MLDesigner sollten nachträglich von der bearbeitenden Person durchgeführt werden.

6.4.2 Spezifikationen der Anwendungsschicht

Die eigentliche Funktionalität des Konverters wird in der Anwendungsschicht implementiert. Sie lässt sich folgendermaßen spezifizieren:

- Das Einlesen einer Quelldatei soll möglich sein.
- Die Quelldatei soll auf Gültigkeit hin geprüft werden.
- Die für die Konvertierung relevanten Inhalte einer gültigen Quelldatei sollen entnommen werden.
- Nach dem Konvertierungsvorgang sollen die Modelldaten entsprechend in Verzeichnisse eingeordnete Zieldateien transferiert werden.
- Der beschriebene Vorgang soll beliebig wiederholbar sein.

6.4.3 Spezifikationen der Betriebssystemschicht

Da der Konverter ein Programm sein wird, das Daten behandelt, welchen unterschiedliche Datenträgerformate zugrunde liegen, muss in der Betriebssystemschicht die folgende Funktionalität verfügbar sein:

- Einzelne XML-Dateien müssen von Windows-basierten Datenträgerformaten einlesbar sein.
- XML-Daten müssen außerdem auf Linux-lesbare Datenträger in verschiedene MML-Dateien schreibbar sein, zusätzlich muss hier eine Verzeichnisstruktur angelegt oder kopiert werden können.

Mit der Spezifikationsfestlegung in allen Schichten sind die Aufgaben und Eigenschaften des Konverters umfassend erläutert: Das zu realisierende Tool soll im Kern ein betriebssystemunabhängiges Werkzeug zur Transformation von XML-Daten werden.

Der Lösungsweg zur Realisierung eines derart spezifizierten Konverters wird im nächsten Abschnitt erarbeitet.

6.5 Entwurf und Lösungsweg

Der im Folgenden erläuterte Entwurf konkreter Algorithmen für den Konverter beschränkt sich auf zwei der unter Abschnitt 6.4 vorgestellten Schichten der Software-Architektur. Da der Umgang mit Dateien und deren Einlesen und Abspeichern auf beliebigen Datenträgern eine bereits implementierte Funktion der verwendeten Programmiersprache, mit welcher der Konverter realisiert wird, sein muss (siehe Abschnitt 6.5.2), kann auf die Erarbeitung von Algorithmen hierfür verzichtet werden. Es erfolgt daher die Konzentration auf die eigentliche Umwandlung der Modelle im XML-Format, welche dann die Arbeit auf Dateiebene gleich enthält, und auf das Design des Werkzeugs, also seine Schnittstelle zum Benutzer, der die Modellumwandlung vornimmt.

6.5.1 Modelltransformation

Für die Transformation von in der Auszeichnungssprache XML vorliegenden Daten in ein anderes XML-Format (oder weitere Datenformate, wie etwa HTML) existiert eine weitere

Sprache, welche selbst dem XML-Standard genügt: die *Extended Stylesheet Language for Transformations* (XSLT) [Bongers 2004; W3C 1999b]. Implementiert man ein Stylesheet, also eine XML-Datei mit Tags aus dem XSLT-Befehlssatz und mit der Endung `.XSL`, so kann man mit dessen Hilfe vorliegende XML-Daten durch die Verwendung eines XSLT-Prozessors in ein anderes Format transferieren. Dem XSLT-Prozessor werden also Originaldaten (XML) und Umwandlungsspezifikationen (XSLT) übergeben, woraus dieser das gewünschte Zielformat entwickeln kann (vgl. Abb. 6.5).

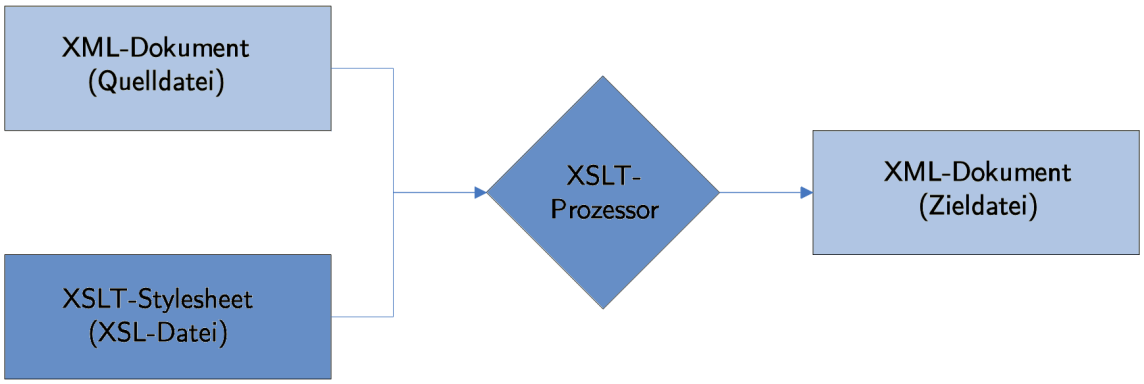

Abbildung 6.5: Prinzip der XSL-Transformation von XML-Daten

XSLT-Prozessoren liegen in zahlreicher Form und in diversen Programmiersprachen vor, so etwa Saxon und Xalan in Java, Xalan und Sablotron in C++ sowie MSXML (Microsoft XML- und XSLT-Prozessor) für die Windows-Plattform allgemein. Viele Softwaresysteme (nicht nur) zur Bearbeitung von XML-Daten bringen darüber hinaus ihren eigenen XSLT-Prozessor mit, wie das in dieser Arbeit unter anderem genutzte Altova XMLSpy [Altova GmbH 2005] oder etwa die Sprache PHP.

Für die Transformation einer Quelldatei im AML-Format, welches ein XML-Derivat darstellt, in die Zieldateien vom MML-Format (ebenfalls XML-zugehörig) müssen also mehrere Stylesheets entworfen und realisiert werden, die a) eine Library-Datei, b) eine MML-Datei für das erstellte System und c) eine Datei mit den Spezifikationen der einzelnen Module und Primitive generieren können (vgl. Abschnitt 6.1.2 und Abb. 6.2). Die MML-Datei mit den Modulspezifikationen muss später von dem Nutzerinteraktionsbestandteil des Konverters, der wie bereits erläutert außerdem die Dateizugriffsfunktionen realisiert, auf eine Verzeichnisstruktur und dort enthaltene Einzeldateien für jedes Modul bzw. Primitiv aufgeteilt werden. Aus diesen Betrachtungen folgt, dass drei separate Stylesheets benötigt werden,

deren algorithmische Inhalte anhand der entwickelten Spezifikationen im Folgenden entwickelt werden.

6.5.1.1 Stylesheet für Library-Datei

Das XSLT-Stylesheet, das aus einer Quelldatei die MML-Datei erstellt, welche für den MLDesigner die notwendigen Informationen zur gesamten neu erzeugten Library enthält, muss, als abstrakter Algorithmus formuliert, die folgende Funktionalität realisieren:

1. `Erstellung eines korrekten XML-(MML-)Headers`
2. `Erzeugen eines Wurzelknotens, der den Namen der Library enthält`
3. `Erzeugen von Metainformationen über die Library: Name, Version inklusive Erstellungsdatum, Autor und Hidden-Status`
4. `Erzeugen von Knoten für alle Elemente der Library, Modelle, Module und Primitive, sowie deren Namen und Speicherort, die innerhalb der Library-Struktur (aus dem Namen ableitbar, siehe Abschnitt 6.1.2) vorhanden sind`

Konkret müssen für die Library-Datei durch das Stylesheet der Quelldatei also die folgenden Informationen entnommen werden:

1. *Name des Systems*
2. *Name der einzelnen Objekte*

Dabei ist zu beachten, dass diese Namen im ARIS Toolset Sonderzeichen und Umlaute enthalten können, was für die Umwandlung in das MLDesigner-Format Probleme bereitet, wenn diese Namen als Verzeichnis- bzw. Dateinamen verwendet werden. Deshalb muss das Library-Stylesheet an diesen Stellen eine Funktion enthalten, die die Sonderzeichen aus dem Namen entfernt und Umlaute in für Dateisysteme verwendbare Buchstaben umwandelt.

6.5.1.2 Stylesheet für System-Datei

Das XSLT-Stylesheet, das aus einer Quelldatei die MML-Datei erstellt, welche für den MLDesigner die notwendigen Informationen zum eigentlichen Modell (System) enthält, muss, als abstrakter Algorithmus formuliert, die folgende Funktionalität realisieren:

1. `Erstellung eines korrekten XML-(MML-)Headers`
2. `Erzeugen eines Wurzelknotens, der den Namen des konvertierten Modells enthält`
3. `Erzeugen von Metainformationen über das Modell: Name, Version inklusive Erstellungsdatum, Autor und Hidden-Status`
4. `Erzeugen von Knoten für alle Elemente des Modells, Module und Primitive, die deren Namen und Speicherort, deren Position im Modell sowie deren Ein- und Ausgänge innerhalb der Modell-Struktur enthalten`
5. `Erzeugen von Knoten mit Informationen über alle Kanten innerhalb des Modells, deren Start- und Endpunkte sowie vorhandene Abbiegungen zur besseren Visualisierung`
6. `Erzeugen von Knoten mit Informationen über die Zusammenhänge zwischen Modellelementen und Kanten, also Verlinkungen von Kanten mit zwei Elementen, woraus hervorgeht, dass die Kante die Verbindung des Ausgang des einen Elements mit dem Eingang des anderen realisiert`

Konkret müssen für die System-Datei durch das Stylesheet der Quelldatei die folgenden Informationen entnommen werden:

1. *Name des Modells*
2. *Name und Position der einzelnen Objekte*
3. *Position und Eckpunkte der Modellkanten*

Auch im System-Stylesheet muss die bereits erläuterte Umwandlungsfunktion für Sonderzeichen und Umlaute Anwendung finden. Darüber hinaus sind nicht alle Informationen, welche der MLDesigner benötigt, in der Quelldatei enthalten. Dies betrifft im Einzelnen die Eingänge der Objekte (es werden nur von den Objekten ausgehende Kanten definiert und angegeben, zu welchem Objekt sie führen) sowie Informationen über Positionen der Ein- und Ausgänge am Objekt selbst. Diese Informationen müssen daher entweder mit Standardwerten versehen oder indirekt ermittelt werden. Näheres hierzu wird im Abschnitt 7.1.2, der Implementierung des System-Stylesheets, aufgeführt.

6.5.1.3 Stylesheet für Modellelement-Datei

Das XSLT-Stylesheet, das aus einer Quelldatei die MML-Datei erstellt, welche für den MLDesigner die notwendigen Informationen zu allen Modellelementen enthält, muss, als abstrakter Algorithmus formuliert, die folgende Funktionalität realisieren:

1. `Erstellung eines korrekten XML-(MML-)Headers`
2. `Erzeugen eines Wurzelknotens, der den Namen des konvertierten Moduls oder Primitivs enthält`
3. `Erzeugen von Metainformationen über das Modul/Primitiv: Name, Version inklusive Erstellungsdatum, Autor und Hidden-Status`
4. `Erzeugen von Knoten für alle Ein- und Ausgänge, deren Position und Ausrichtung, die dem Modul/Primitiv angehören`
5. `Erzeugen von Knoten für alle Parameter des Moduls/Primitivs, inklusive Namen und Wert`
6. `Erzeugen von Knoten mit Informationen über alle Kanten, die das Modul/Primitiv erreichen oder von ihm ausgehen, deren Start- und Endpunkte sowie vorhandene Abbiegungen zur besseren Visualisierung`
7. `Erzeugen von Knoten mit Informationen über die Zusammenhänge zwischen den Kanten und weiteren Modellelementen, also Verlinkungen von Kanten mit zwei Elementen, woraus hervorgeht, dass die Kante die Verbindung des Ausgangs des einen Elements mit dem Eingang des anderen realisiert`

Konkret müssen für die Modul-/Primitiv-Datei durch das Stylesheet der Quelldatei die folgenden Informationen entnommen werden:

1. *Name und Typ des Moduls/Primitivs*
2. *Name und Wert aller Parameter*
3. *Position und Ausrichtung der Ein- und Ausgänge*
4. *Position und Eckpunkte der Modellkanten*

Auch im Modul-/Primitiv-Stylesheet muss die bereits erläuterte Umwandlungsfunktion für Sonderzeichen und Umlaute Anwendung finden. Darüber hinaus sind auch hier nicht alle Informationen, welche der MLDesigner benötigt, in der Quelldatei enthalten, so wiederum Position und Ausrichtung der Ein- und Ausgänge sowie detaillierte Informationen über „Verlinkung“ durch Kanten und Modellelementeingänge. Näheres hierzu enthält Abschnitt 7.1.2.

6.5.2 Nutzerinteraktion

Neben der Realisierung der Anwendungsschicht, die mit den XSLT-Stylesheets zur Umwandlung der Quelldatei in die Zieldateien spezifiziert ist, bedarf es der Programmierung eines Interfaces für die Benutzerschicht, das zum einen das Nutzerinterface selbst erzeugt und zum anderen das Management der XSLT-Funktionen in Verbindung mit den korrekten Dateizugriffen übernimmt. In diesem Sinne lassen sich drei zentrale Aufgaben der Benutzerschicht ausmachen: die Erzeugung der Benutzeroberfläche, das Datenmanagement beim Konvertiervorgang und das Management des Datenexports. Diese Funktionalität soll möglichst – wie auch die XSLT-Stylesheets – betriebssystemübergreifend zur Verfügung stehen, weshalb sich als Programmiersprache für die spätere Implementierung Java anbietet.

6.5.2.1 Benutzeroberfläche

Die eigentliche Interaktion mit dem den Konverter Bedienenden sollte über ein grafisches Interface erfolgen, das die folgende Funktionalität bereit stellen muss:

1. `Erzeugung eines Hauptfensters, das ein Auslösen aller gewünschten Funktionen des Konverters ermöglicht: die Auswahl der Quelldatei, das Auslösen des Konvertierungsvorganges und den Datenexport`
2. `Die Auswahl der Quelldatei sollte über einen vom jeweiligen Betriebssystem her bekannten Dateiauswahldialog geschehen; gleiches gilt für den Datenexport`
3. `Im Idealfall enthält die Benutzeroberfläche noch eine Hilfefunktion und ein mögliches Schließen der Anwendung über eine separate Schaltfläche`

6.5.2.2 Konvertierung

Die eigentliche Datenkonvertierung geschieht durch die Anwendung der XSLT-Stylesheets auf die vom Nutzer ausgewählte Quelldatei. Dabei ist Folgendes durchzuführen:

1. `Anwendung der drei XSLT-Stylesheets auf die Quelldatei`
2. `Speichern der Transformationsresultate in einem temporären Verzeichnis`
3. `Anlegen eines Unterverzeichnisses mit dem Namen des eigentlichen Modells und Umbenennen der System-MML-Datei in ebendiesen Namen;`
   `gleichzeitig Verschieben dieser Datei in das neu erstellte Verzeichnis`
4. `Anlegen je eines Unterverzeichnisses mit dem Namen eines jeden Moduls oder Primitivs, das im Modell vorkommt`
5. `Entnahme der MML-Daten für je ein einzelnes Modul/Primitiv und Speichern dieser Daten in einer MML-Datei mit dem Namen des Moduls/Primitivs innerhalb des für es erstellten Unterverzeichnisses`

Mit dem Durchlaufen dieser Arbeitsschritte ist die MLDesigner-Dateistruktur, wie sie in Abschnitt 6.1.2 beschrieben wurde, hergestellt, mit Ausnahme aller anderen Dateiformate. Diese werden zum einen später beim Import vom MLDesigner ergänzt (etwa die HTML-Dateien), zum anderen müssen sie vom Nutzer selbst mit Code gefüllt werden, etwa was die konkrete Funktion der Primitive im MLDesigner betrifft (enthalten in den PL-Dateien), die im ARIS Toolset (noch) nicht vorhanden war.

6.5.2.3 Export

Die dritte Aufgabe der Benutzerschicht ist der vom Bedienenden anzustoßende Datenexport. Nachdem unter Umständen mehrere Quelldateien einer Konvertierung unterzogen worden sind, soll die dadurch entstandene MLDesigner-Library, welche alle Modelle enthält, nun abgespeichert werden. Dazu müssen folgende Schritte abgearbeitet werden:

1. `Erzeugen eines TAR-Containers, der alle aus der Konvertierung resultierenden Dateien und Ordner enthält, außerdem eine INF-Datei mit Informationen über die Dateistruktur`
2. `Anzeige eines Datei-Speichern-Dialogs für den Speicherort der fertigen Library`

3. `Verschieben der erzeugten TAR-Datei an den angegebenen Ort und Umbenennen in den angegebenen Namen inklusive vom MLDesigner geforderter Dateiendung .mar`
4. `Nach Beendigung des Exports oder des gesamten Konverters Löschen aller angelegten temporären Dateien`

Nach der Durchführung dieser Funktionen ist eine importfähige MAR-Datei aus der Quelldatei entstanden; die notwendige Funktionalität der Benutzerschicht des Konverters ist damit umfassend beschrieben. Im folgenden Kapitel kann daher auf die Implementierungsdetails zum einen der Stylesheets der Anwendungsschicht und zum anderen der Java-Klassen der Benutzerschicht (und implizit auch der Betriebssystemschicht) eingegangen werden.

7 Implementierung

Der Konverter zur Umwandlung von ARIS-Toolset-Modellen in das MLDesigner-Format ist ein betriebssystemunabhängiges Werkzeug zur Transformation von XML-Daten. Er besteht aus zwei hinsichtlich der verwendeten Programmiersprachen und -prinzipien disjunkten Komponenten, zum einen aus den transformationsspezifischen XSLT-Stylesheets als Realisierung der Anwendungsschicht (siehe Abschnitt 6.4.2) und zum anderen aus einem Mensch-Maschine-Interface für Nutzerinteraktion und Dateizugriffe in Form von Java-Klassen als Realisierung sowohl der Benutzerschicht (Abschnitt 6.4.1) als auch der Betriebssystemschicht (Abschnitt 6.4.3). Dieser Aufbau wird in Abb. 7.1 verdeutlicht.

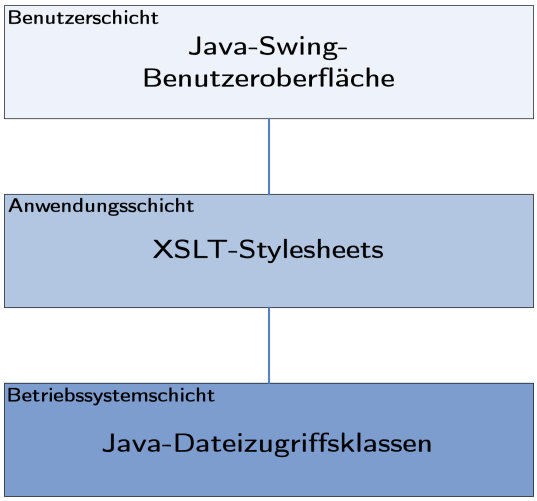

Abbildung 7.1: Software-Architektur des ARIS-MLDesigner-Konverters

Nachdem im vorangegangenen Kapitel über eine Anforderungsanalyse die Spezifikationen zu beiden Konverter-Komponenten entwickelt und daraus im Entwurf abstrakt-algorithmische Vorgehensweisen abgeleitet werden konnten, erfolgt in den nächsten Abschnitten die Erläuterung der konkreten Implementierungsdetails für jeden Bestandteil des Werkzeugs.

7.1 Modellkonvertierung durch XSLT

Da Quell- und Zieldateien im XML-Format vorliegen, wird für die Konvertierung XSLT verwendet [W3C 1999b; Bongers 2004; Tidwell 2002]. Um aus einer vorliegenden Quelldatei die vom MLDesigner geforderte Dateistruktur (siehe Abschnitt 6.1.2) etablieren zu können, sind drei verschiedene Stylesheets notwendig, die aus der Quelldatei die jeweils benötigten Informationen entnehmen und daraus Zieldateien generieren, welche die Library, das enthaltene Modell bzw. die enthaltenen Modellelemente beschreiben. An diese Aufgaben angelehnt, wurden die Stylesheets `library.xsl`, `system.xsl` und `module.xsl` benannt.

7.1.1 XSLT-Dokument `library.xsl`

Aufgabe des Stylesheets `library.xsl` ist es, der Quelldatei die notwendigen Informationen zu entnehmen, um eine MML-Datei zu generieren, welche die komplette entstehende Library inklusive Namen und Speicherort aller enthaltenen Modelle und Elemente beschreibt. Hierfür wird nach dem in Abschnitt 6.5.1.1 formulierten Algorithmus vorgegangen.

1. Erstellung eines korrekten XML-(MML-)Headers:
   ```
   <xsl:output method="xml" encoding="ISO-8859-1" indent="yes"/>
   ```
2. Erzeugen eines Wurzelknotens, der den Namen der Library enthält:
   ```
   <xsl:template match="/">
     <library name="converted_library" version="2.5.r00">
   ```
3. Erzeugen von Metainformationen über die Library: Name, Version inklusive Erstellungsdatum, Autor und Hidden-Status:

   ```
   <property class="String" name="Logical Name" value="converted_library"/>
   <property class="String" name="Version" value="{concat('0.0 ',translate
                          (/AML/Header-Info/@CreateDate,'-','/'))}"/>
   <property class="String" name="Author" value="ARIS-MLD-Konverter"/>
   <property class="String" name="hidden" value="no"/>
   ```

   Wie sich hier zeigt, werden für fast alle Library-Metainformationen fixe, vorher festgelegte Werte verwendet, so etwa der Library-Name `converted_library`. Einzig das Er-

stellungsdatum wird aus der Quelldatei übernommen und für das MLDesigner-Format angepasst (MM/TT/JJJJ statt MM-TT-JJJJ).

4. Erzeugen von Knoten für alle Elemente der Library, Modelle, Module und Primitive, sowie deren Namen und Speicherort:

```
    <xsl:apply-templates select="AML/Group/Model/AttrDef[@AttrDef.Type=
        'AT_NAME']"/>
    <xsl:apply-templates select="AML/Group/ObjDef/AttrDef[@AttrDef.Type=
        'AT_NAME']"/>
  </library>
</xsl:template>
```

Konkret müssen für die Library-Datei durch das Stylesheet der Quelldatei also die folgenden Informationen entnommen werden:

1. Name des Systems, zu finden unter
   `AML/Group/Model/AttrDef@AttrDef.Type='AT_NAME' Value='[Systemname]'`
2. Name der einzelnen Objekte, zu finden unter
   `AML/Group/ObjDef/AttrDef@AttrDef.Type='AT_NAME' Value='[Systemname]'`

Wie aus den oben angegebenen Codebestandteilen ersichtlich ist, werden in XSLT sogenannte Template-Regeln angewandt, um aus in der Quelldatei vorhandenen XML-Knoten notwendige Informationen zu extrahieren und in einen neuen XML-Ergebnisbaum, der im Stylesheet bereits vorimplementiert ist, einzufügen. Eine solche Template-Regel wird hier auch genutzt, um die Umwandlung von Sonderzeichen und Umlauten, wie unter Abschnitt 6.5.1.1 erläutert, zu realisieren. Dabei wird der aus der Quelldatei entnommene Name einer fünffachen Umwandlung (für Sonderzeichen, Leerzeichen und die drei deutschen Umlaute) unterzogen und in einer XPath-Variable gespeichert [W3C 1999a]. Diese Variable wird schließlich wiederholt verwendet, um die Verzeichnisstruktur der Zieldateien zu beschreiben:

```
<xsl:template match="AttrValue">
  <xsl:variable name="ObjectName" select="translate(translate(translate(translate
    (translate(.,'.,()/:{*}?\&quot;\&gt;\&lt;|\&amp;\&\#xA;',''),'\&\#x20;','\_'),
```

```
    'ä','a'),'ö','o'),'ü','u')"/>
  <ref name="{$ObjectName}" url="file:$MLD_USER/converted_library/{$ObjectName}/
    {$ObjectName}.mml"/>
</xsl:template>
```

7.1.2 XSLT-Dokument `system.xsl`

Das zweite XSLT-Stylesheet ist für die Erstellung der MML-Datei verantwortlich, welche das in der Quelldatei enthaltene Modell umfassend beschreiben soll. Da sich die Algorithmen dieses Stylesheets an einigen Stellen an diejenigen des Library-Transformationsdokumentes halten, das im letzten Abschnitt vollständig erläutert wurde, sollen hier nur diejenigen Implementierungsdetails betrachtet werden, die nicht in den bisher dargelegten Verfahrensweisen enthalten sind. Dazu zählen die folgenden Transformationsvorgänge:

1. Erzeugen von Knoten für alle Elemente des Modells, Module und Primitive, die deren Namen und Speicherort, deren Position im Modell sowie deren Ein- und Ausgänge innerhalb der Modell-Struktur enthalten:
   Die Position der Modelle, Ein-/Ausgänge etc. wird über eine Umrechnung ermittelt, da die Koordinatensysteme für die Modellerstellung in ARIS Toolset und MLDesigner, obwohl beide auf pixelgenauem Positionieren basieren, sich in ihren Ursprungslokalisationen unterscheiden (vgl. Abb. 7.2). Demzufolge gelten die Formeln $x_{MLD} = \lfloor \frac{x_{ARIS}-1300}{2} \rfloor$ und $y_{MLD} = \lfloor \frac{y_{ARIS}-1500}{2} \rfloor$, wobei x_{MLD}und y_{MLD} die Koordinaten eines Objekts im MLDesigner darstellen und x_{ARIS}und y_{ARIS} die Koordinaten desselben Objekts im ARIS Toolset; $\lfloor x \rfloor$bedeutet dabei ein auf die ihm vorangehende ganze Zahl abgerundetes x:

   ```
   <xsl:for-each select="Position">
     <xsl:value-of select="floor((@Pos.X - 1300) div 2)"/>
     <xsl:value-of select="concat(',',floor((@Pos.Y - 1500) div 2),' ')"/>
   </xsl:for-each>
   ```

   Die Ein- und Ausgänge eines jeden Objekts müssen auf indirekte Art ermittelt werden, da hierzu in der Quelldatei keine direkten Informationen vorliegen. Einziger Anhaltspunkt ist ein Attribut `ToObjOcc` der Kantendefinitionen `CxnOcc`, der auf eine ausgehende Kante in Richtung eines weiteren Objekts hinweist [IDS Scheer AG 2005c]. Aus diesem Grunde wurde jedem Objekt grundsätzlich ein Eingang zugeordnet (der

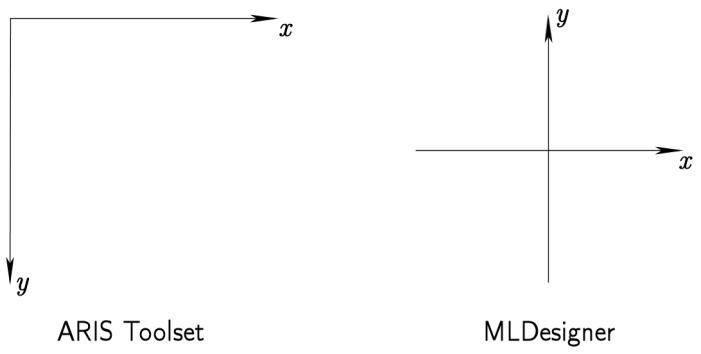

Abbildung 7.2: Koordinatensysteme von ARIS Toolset und MLDesigner

beim Startobjekt dann gegebenenfalls wieder entfernt werden kann), Position und Anzahl der Ausgänge aber anhand der Anzahl und Werte der `ToObjOcc`-Attribute ermittelt und eingebunden:

```
<xsl:template match="/AML/Group/Model/ObjOcc/CxnOcc" mode="Ports">
[...]
  <port class="datastruct" datastruct="Root" name="{concat('Output',
        position())}" type="output">
    <svg:svg font-style="italic" height="100%" width="100%">
      <property class="String" name="PortAlign" value="{$PortAlignment}"/>
      <property class="String" name="Rotation" value="0.0"/>
      <property class="String" name="ConnectPoint" value="{$PortPosition}"/>
      <property class="String" name="Position" value="{$PortPosition}"/>
    </svg:svg>
  </port>
```

Des Weiteren musste ein Fehler in der aktuellen ARIS-Toolset-Version 7.0 festgestellt werden: Sporadisch, aber reproduzierbar werden Modellkanten beim XML-Export „vergessen“. Für diesen Fall, also wenn ein Objekt scheinbar gar keinen Ausgang (und keine ausgehende Kante) besitzt, wurde ein XSLT-Konstrukt integriert, das solchen Objekten dennoch zumindest einen Ausgang zuordnet. Auf diese Weise besitzt auch jedes Objekt einen Ausgang, was im MLDesigner durchaus wünschenswert sein kann, wenn die Ergebnisdaten der „letzten“ Module oder Primitive eine Weiterverarbeitung oder grafische Darstellung erfahren sollen.

2. Erzeugen von Knoten mit Informationen über Verlinkungen von Kanten:
   Diese Teilaufgabe muss über ein komplexes Template mit verschachtelten Quelldatei-Abfragen gelöst werden, da sich die Informationen, dass eine Verlinkung besteht und diejenige, womit verlinkt wird, an unterschiedlichen Stellen in der AML-Datei befinden. Die Abfrage wird daher in einer größeren Anzahl von Variablen gespeichert, die aufeinander Bezug nehmen und deren finale Konstrukte schließlich in der eigentlichen Verlinkung (die nur zwei Zeilen XML-Code umfasst) Verwendung finden:

```
<xsl:template match="/AML/Group/Model/ObjOcc/CxnOcc" mode="Links">
  <xsl:variable name="OutputID" select="../@ObjDef.IdRef"/>
  <xsl:variable name="OutputName" select="translate(translate(translate
      (translate(translate(/AML/Group/ObjDef[@ObjDef.ID=$OutputID]/AttrDef
      [@AttrDef.Type='AT_NAME']/AttrValue,'.,()/:*?&amp;quot;&amp;gt;&amp;lt;|
      &amp;amp;&amp;#xA;',''),'&amp;#x20;','_'),'ä','a'),'ö','o'),'ü','u')"/>
  <xsl:variable name="InputOccIDRef" select="@ToObjOcc.IdRef"/>
  <xsl:variable name="InputOccID" select="/AML/Group/Model/ObjOcc
      [@ObjOcc.ID=$InputOccIDRef]/@ObjOcc.ID"/>
  <xsl:variable name="InputID" select="/AML/Group/Model/ObjOcc
      [@ObjOcc.ID=$InputOccID]/@ObjDef.IdRef"/>
  <xsl:variable name="InputName" select="translate(translate(translate
      (translate(translate(/AML/Group/ObjDef[@ObjDef.ID=$InputID]/AttrDef
      [@AttrDef.Type='AT_NAME']/AttrValue,'.,()/:*?&amp;quot;&amp;gt;&amp;lt;|
      &amp;amp;&amp;#xA;', ''),'&amp;#x20;','_'),'ä','a'),'ö','o'),'ü','u')"/>
  <xsl:variable name="OutputNumber">
    <xsl:number count="CxnOcc"/>
  </xsl:variable>
  <link port="{concat($OutputName,'#1.output',$OutputNumber)}" relation=
      "{concat('Relation',position())}"/>
  <link port="{concat($InputName,'#1.input')}" relation="{concat
      ('Relation',position())}"/>
</xsl:template>
```

3. Schließlich muss bei der Modellumwandlung geprüft werden, an welchen Stellen AND- bzw. XOR-Regeln verwendet wurden. Diese Regeln werden nicht in Module transferiert, sondern auf die beiden Primitive *ProbSwitch* bzw. *fork* abgebildet:

```
<xsl:when test="(/AML/Group/ObjDef[@ObjDef.ID=$ObjectID]/@TypeNum='OT_RULE')
                and (@SymbolNum='ST_OPR_OR_1')">
  <entity class="file:$MLD/MLD_Libraries/DE/Control/Fork/Fork.output=2.mml#
                Fork.output=2" name="{$ObjectFileName}#1">
    [...MML-Code des fork-Primitivs...]
  </entity>
</xsl:when>
<xsl:when test="(/AML/Group/ObjDef[@ObjDef.ID=$ObjectID]/@TypeNum='OT_RULE')
                and ((@SymbolNum='ST_OPR_XOR_1')
                or (@SymbolNum='ST_BPMN_RULE'))">
  <entity class="file:$MLD/MLD_Libraries/DE/Switches/ProbSwitch/ProbSwitch.
                mml#ProbSwitch" name="{$ObjectFileName}#1">
    [...MML-Code des ProbSwitch-Primitivs...]
  </entity>
</xsl:when>
```

Konkret müssen für die System-Datei durch das Stylesheet der Quelldatei die folgenden Informationen entnommen werden:

1. Name des Systems, zu finden unter
   `AML/Group/Model/AttrDef@AttrDef.Type='AT_NAME' Value='[Systemname]'`
2. Name der einzelnen Objekte, zu finden unter
   `AML/Group/ObjDef/AttrDef@AttrDef.Type='AT_NAME' Value='[Systemname]'`
3. Position der einzelnen Objekte, zu finden unter
   `/AML/Group/Model/ObjOcc/ObjOcc/Position@Pos.X` und `Pos.Y`
4. Typ der einzelnen Objekte, zu finden unter
   `AML/Group/ObjDef@TypeNum`

5. Position und Eckpunkte der Modellkanten, zu finden unter

   `/AML/Group/Model/ObjOcc/CxnOcc/Position@Pos.X` und `Pos.Y`

7.1.3 XSLT-Dokument `module.xsl`

Das dritte Stylesheet, das für die Transformation der Objektdaten in eine MML-Containerdatei verantwortlich ist, die zunächst die einzelnen Informationen über Module und Primitive gesammelt enthält, lehnt sich stark an die im vorigen Abschnitt beschriebene Datei `system.xsl` an. Es existieren nur geringe spezifische Unterschiede:

1. Es existiert nicht nur ein Wurzelknoten, sondern die Anzahl an diesen entspricht der Menge der aus der Quelldatei ausgelesenen und im Modell enthaltenen Objekte. Deshalb beginnt die erste Template-Regel, die immer durchlaufen wird, in diesem Dokument gleich mit dem Verweis auf die `ObjOcc`-Tags der AML-Datei und nicht, wie bisher, erst auf das Modell selbst:

   ```
   <xsl:template match="/">
     <xsl:apply-templates select="AML/Group/Model/ObjOcc"/>
   </xsl:template>
   ```

2. Das Erzeugen von Knoten für alle Parameter eines Moduls oder Primitivs, inklusive Namen und Wert, ist hier notwendig:

   ```
   <xsl:template match="/AML/Group/ObjDef/AttrDef" mode="Parameters">
     <xsl:for-each select="AttrValue">
       <parameter attributes="A_CONSTANT|A_SETTABLE" name="{../@AttrDef.Type}"
                  scope="External" type="string" value="{.}"/>
     </xsl:for-each>
   </xsl:template>
   ```

   Hierfür wird in einem separaten Template eine Schleife aufgerufen, die alle Modellparameter sammelt und im MML-Format ausgibt. Alle Parameter sind standardmäßig als Datentyp String (Zeichenkette) definiert; eventuelle Änderungen kann der Nutzer dann bei der Hinterlegung des Modells mit spezifischem Programmcode hier vornehmen.

Konkret müssen für die Modul-/Primitiv-Datei durch das Stylesheet der Quelldatei die folgenden Informationen entnommen werden:

1. Name des Moduls/Primitivs, zu finden unter
   `/AML/Group/ObjDef/AttrDef[@AttrDef.Type='AT_NAME']/AttrValue`
2. Name und Wert aller Parameter, zu finden unter
   `AML/Group/ObjDef/AttrDef`
3. Position und Ausrichtung der Ein- und Ausgänge, zu finden unter
   `/AML/Group/Model/ObjOcc/CxnOcc/Position@Pos.X` und `Pos.Y`
4. Position und Eckpunkte der Modellkanten, zu finden ebenfalls unter
   `/AML/Group/Model/ObjOcc/CxnOcc/Position@Pos.X` und `Pos.Y`

7.2 Programmrealisierung mit Java

Die Anwendung der drei bisher beschriebenen und implementierten XSLT-Transformationsanweisungen mit Hilfe eines XSLT-Prozessors setzt zum einen ein weiteres Werkzeug voraus, das einen solchen Prozessor beinhaltet und startet; zum anderen entspricht die Ergebnisstruktur, bestehend aus drei XML-Dateien, noch nicht der korrekten Verzeichnisstruktur der Zieldateien. Aus diesem Grunde muss die XSLT-Transformation in ein weiteres Programm eingebettet werden, das erstens eine Benutzerinteraktion erleichtert (Benutzerschicht) und zweitens die Zieldateistruktur etabliert (Betriebssystemschicht). Für diese Aufgabe wurde aus den folgenden Gründen die Programmiersprache Java gewählt:

- betriebssystemunabhängige Programmiersprache
- Dateizugriffe möglich [Ullenboom 2005, S. 621–663]
- Benutzerinterfaces mit der Swing-API generierbar [Jesse 2001]
- XML-Datenzugriffe und XSLT-Transformationen ausführbar [Burke 2002]

Mit Java ist also die Realisierung eines Benutzerinterfaces möglich, das die XSLT-Transformation, also die eigentliche Konvertierung, anstößt und weiterführt, mehrere Konvertierungen in einem Schritt durchführen lassen kann und – als kompilierte JAR-Datei – sowohl

unter Linux als auch unter Windows lauffähig ist (ein entsprechend installiertes Java Runtime Environment (JRE) vorausgesetzt), womit es exakt den formulierten Anforderungen entspricht (Abschnitt 6.3).

Für die Programmentwicklung wurde als Entwicklungsumgebung die Eclipse IDE in der Version 3.1.1 genutzt [Eclipse Foundation 2006; Assisi 2004]. Weiterhin wurde das aktuelle Java Software Development Kit (JDK) 1.5.0_07-b03, welches das JRE in der gleichen Version enthält, verwendet.

Im bereits ausgeführten Programmentwurf (siehe Abschnitt 6.5.2) wurden die Aufgabenbereiche des Java-Systems in drei Kategorien eingeteilt. In den nächsten Abschnitten wird erläutert, wie die Spezifikationen für diese Kategorien in konkrete Java-Klassen umgesetzt wurden.

7.2.1 Klassen für die Benutzeroberfläche

Im Entwurfsabschnitt für die Benutzeroberfläche (Abschnitt 6.5.2.1) wurden drei algorithmisch formulierte Spezifikationen für diesen Konverter-Bestandteil definiert, die in vier separaten Java-Klassen realisiert werden:

7.2.1.1 Klasse *Hauptfenster*

Das grundlegende GUI des Konverters wird als Hauptfenster bezeichnet. Das Hauptfenster wird mit Hilfe der Swing-API, die in Java betriebssystemspezifische Anwendungen, Menüs, Schaltflächen etc. ermöglicht, in der Mitte des Nutzerbildschirms präsentiert. Es enthält, in einem dreizeiligen sogenannten *GridLayout* dargestellt, die folgenden Komponenten:

1. Ein Texteingabefeld und eine Schaltfläche *Durchsuchen*, die beide die Spezifizierung einer Quelldatei ermöglichen:
   ```
   final JTextField dateiFeld =
     new JTextField("Bitte zu konvertierende AML-Datei auswählen!", 40);
   JButton durchsuchenButton = new JButton("Durchsuchen...");
   ```
2. Zwei Schaltflächen *Konvertieren* und *Export*, die das Konvertieren der Quelldatei bzw. den Export nach x Konvertiervorgängen auslösen sollen:
   ```
   JButton konvertierenButton = new JButton("Konvertieren");
   ```

```
JButton exportierenButton = new JButton("Exportieren");
```

3. Zwei Schaltflächen *Hilfe* und *Beenden*, welche einen kurzen Anleitungstext anzeigen bzw. das Programm terminieren sollen.

Damit hat das Hauptfenster, abhängig von den nutzerspezifischen Visualisierungseinstellungen, in etwa das in Abb. 7.3 dargestellte Aussehen.

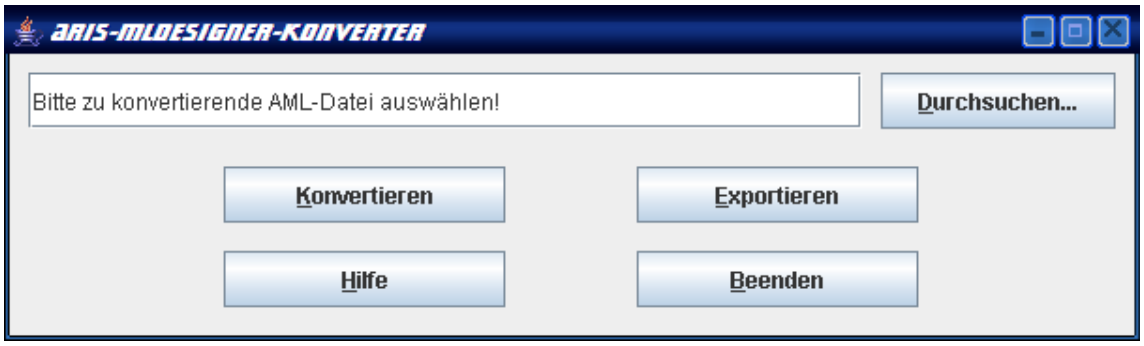

Abbildung 7.3: ARIS-MLDesigner-Konverter – Hauptfenster

Weiterhin enthält die Hauptfensterklasse die Funktionalität, bei den Ereignissen der jeweiligen Schaltflächenbetätigung Objekte der weiteren Java-Klassen zu generieren, um so das Durchsuchen, das Konvertieren, den Export, die Hilfe oder das Beenden zu ermöglichen. Dazu wird jede Schaltfläche mit einem sogenannten *ActionListener* hinterlegt, der bei ihrer Betätigung entsprechende Befehle ausführen kann. Dies wird hier am Beispiel des *ActionListener* für den Hilfe-Button dargestellt:

```
ActionListener hilfeGeklickt = new ActionListener() {
        public void actionPerformed(ActionEvent e) {
                new Hilfefenster();
        }
};
hilfeButton.addActionListener(hilfeGeklickt);
```

In den abschließenden Befehlen der *Hauptfenster*-Klasse wird das bis dahin konstruierte Fenster am Bildschirm präsentiert. Schließlich enthält die *Hauptfenster*-Klasse eine *Main*-Methode, womit sie zur beim Programmstart auszuführenden Klasse wird. Die *Main*-Methode ruft dabei die *Hauptfenster*-Methode auf. Diese Abfolge resultiert darin, dass zum

Programmstart das Hauptfenster am Bildschirm erscheint und der Benutzer seine Eingaben tätigen kann.

7.2.1.2 Klasse *Durchsuchenfenster*

Die Spezifizierung der Quelldatei, die konvertiert werden soll, kann auf zwei unterschiedliche Wege erfolgen. Zum einen kann der Nutzer den Dateinamen inklusive vollständigem Pfad in das Textfeld im Hauptfenster des Konverters eingeben. Damit wird ebenfalls ermöglicht, dass eventuell in der Zwischenablage vorhandene Pfade zur Quelldatei sofort durch ein Einfügen des Zwischenablageninhalts in das Textfeld zur Konvertierung der Datei in den Konverter übernommen werden können.

Die zweite Möglichkeit bietet die Betätigung der Schaltfläche *Durchsuchen*, bei der ein Objekt der Klasse *Durchsuchenfenster* generiert wird. Die implementierte Klasse Durchsuchenfenster ist dabei von der Java-Swing-Klasse *JFileChooser* abgeleitet. Diese Klasse dient zur Darstellung eines vom jeweiligen Betriebssystem her bekannten Datei-Öffnen-Dialogs. Angepasst wird dieser Dialog in dem Sinne, dass nur XML-Dateien ausgewählt werden können (da die Quelldatei beim ARIS-Export mit dieser Dateiendung generiert wird), als Dateiname wird außerdem „AML-Datei“ angezeigt, woraus hervorgeht, dass nur solche XML-Dateien ausgewählt werden sollten, die dem ARIS-Datenexport-Format entsprechen.

```
public class Durchsuchenfenster extends JFileChooser {

  public Durchsuchenfenster()
          throws Exception{
          setFileFilter(new FileFilter() {
              public boolean accept(File datei) {
                  return datei.isDirectory() ||
                      datei.getName().toLowerCase().endsWith(".xml");
                  }
                  public String getDescription() {
                          return "AML-Dateien";
                  }
          } );
```

```
        if (showOpenDialog(null) == JFileChooser.APPROVE_OPTION) {}
    }
```

Ein Beispiel für den Durchsuchen-Dialog findet sich in Abb. 7.4.

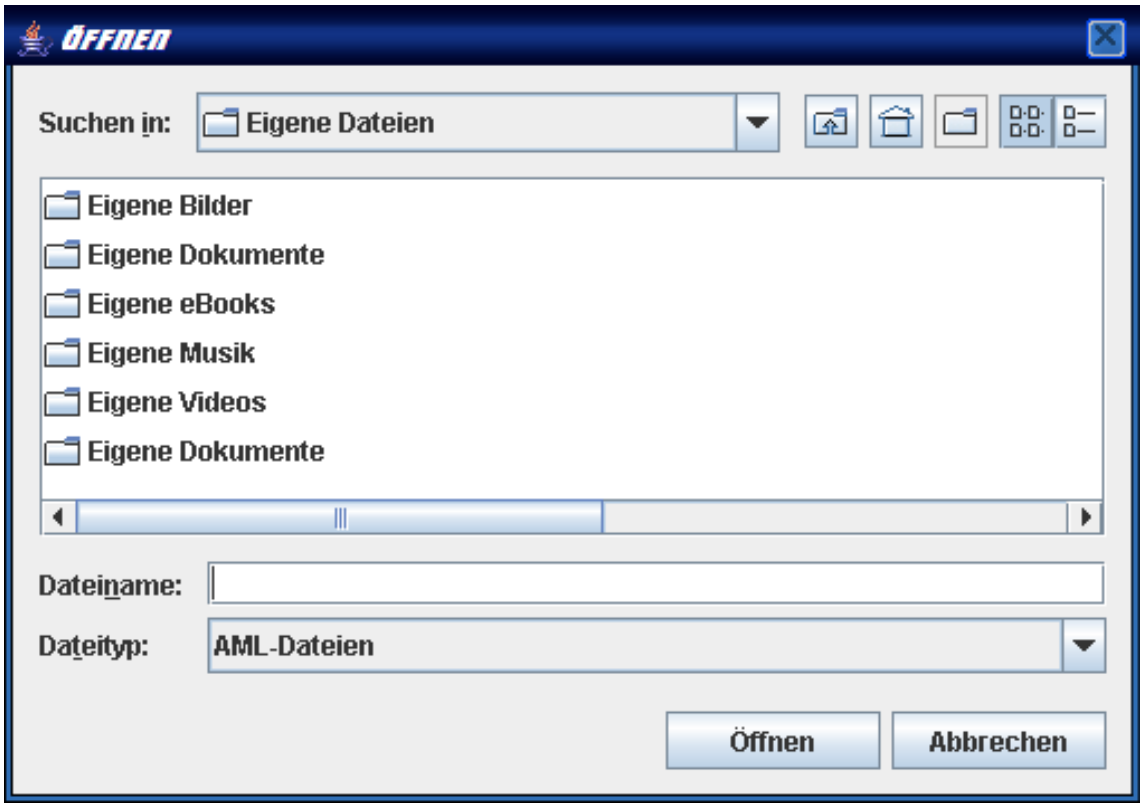

Abbildung 7.4: ARIS-MLDesigner-Konverter – Datei-Öffnen-Dialog

7.2.1.3 Klasse *Hilfefenster*

Den dritten Bestandteil der Benutzerschicht des Konverters bildet ein weiteres Fenster, das beim Betätigen der Schaltfläche *Hilfe* angezeigt wird und einen kurzen Anleitungstext zur Verwendung des Konverters enthält. Über einen *Schließen*-Button kann dieses Fenster wieder vom Bildschirm entfernt werden, es wird also neben einem *JLabel* mit dem Hilfetext und der Schaltfläche ein *ActionListener* für das Schließen implementiert, der bei Schaltflächenbetätigung das Hilfefenster entfernt. Auf Codebeispiele wird hier wegen der Analogie zu den beiden vorangehend beschriebenen Klassen verzichtet. Das Hilfefenster selbst ist in Abb. 7.5 dargestellt.

Abbildung 7.5: ARIS-MLDesigner-Konverter – Hilfefenster

7.2.1.4 Klasse *Beenden*

Eine letzte zur Benutzerschicht zählende Klasse ist für die Ereignisbehandlung beim Betätigen der Beenden-Schaltfläche zuständig. Hier muss geprüft werden, ob alle konvertierten Modelle auch bereits exportiert worden sind (siehe unten, Abschnitte 7.2.2 und 7.2.3). Falls nicht, so soll eine Warnung vor dem Beenden erfolgen, um das Beenden bei vergessenem Exportieren abbrechen zu können. Ist der Benutzer mit dem Beenden dennoch einverstanden oder existieren keine unbehandelten Konvertierungsdaten mehr, so kann das Programm direkt beendet werden, wobei vorher noch, wenn vorhanden, durch ein Objekt der Klasse *TempDateienLoeschen* (Abschnitt 7.2.3.2) alle temporären Dateien entfernt werden sollten. Insgesamt entsteht dadurch der folgende Code:

```
if (new File(System.getProperties().getProperty("java.io.tmpdir")
  + "/converted_library").exists()) {
    int ende = JOptionPane.showConfirmDialog(null,
```

```
      "Konvertierte Modelle wurden noch nicht exportiert.\n" +
      "Beenden, ohne zu exportieren?",
      "Warnung",
      JOptionPane.OK_CANCEL_OPTION,
      JOptionPane.WARNING_MESSAGE
    );
    if (ende == JOptionPane.YES_OPTION)
    {
        new TempDateienLoeschen();
        System.exit(0);
    }
} else {
    System.exit(0);
}
```

7.2.2 Klasse für die Konvertierung

Die eigentliche Datenkonvertierung wird von einer separaten Java-Klasse *Konverter* ausgeführt, welche die Transformation unter Anwendung der in den XSLT-Stylesheets festgelegten Regeln durchführt (vgl. Abschnitt 7.1) und außerdem das Transformationsergebnis von drei XML-Dateien in die korrekte MLDesigner-Struktur bringt. Dafür werden die folgenden Arbeitsschritte durchlaufen:

1. Prüfung, ob zunächst eine Quelldatei spezifiziert wurde – falls nicht, so wird eine Fehlermeldung ausgegeben und die Bearbeitung beendet:

   ```
   if (!(amlDateiStr.equals("Bitte zu konvertierende AML-Datei auswählen!")))
   {
       [...Code zur Konvertierung...]
   } else {
       JOptionPane.showMessageDialog(null,
       "Konvertieren nicht möglich.\nKeine Datei ausgewählt!",
   ```

```
    "Fehler",
    JOptionPane.ERROR_MESSAGE
    );
}
```

2. Dateien in Java initialisieren, also Variablen für sie bereitstellen; verwendet werden *convertedLibrary*, *libraryFile*, *oldLibraryFile*, *systemFile* und *modulesFile*.

3. Prüfen, ob das temporäre Verzeichnis `converted_library`, in das die Transformationsergebnisse gespeichert werden, schon existiert, falls nein, dieses Verzeichnis anlegen; falls ja, die dort enthaltene Datei `converted_library.mml` nach `converted_library.mml~` umbenennen.

4. Quell- und Zieldateien für die Transformation festlegen:

```
Source xmlSource = new StreamSource(new File(amlDateiStr));
Result library = new StreamResult();
library.setOutputStream(new FileOutputStream(libraryFile));
Result system = new StreamResult(systemFile);
system.setOutputStream(new FileOutputStream(systemFile));
Result modules = new StreamResult(modulesFile);
modules.setOutputStream(new FileOutputStream(modulesFile));
```

5. XSLT-Stylesheets für die Transformation spezifizieren (hier am Beispiel des Library-Stylesheets):

```
Transformer libraryTrans =
    TransformerFactory.newInstance().newTransformer(new StreamSource(
        getClass().getResource("library.xsl").toString()));
```

6. Transformation durchführen:

```
libraryTrans.transform(xmlSource, library);
systemTrans.transform(xmlSource, system);
moduleTrans.transform(xmlSource, modules);
```

7. Falls schon vorher Konvertierungsdaten vorhanden waren, die älteren Daten an die der neuen Datei anhängen:

```
RandomAccessFile libraryappend = new RandomAccessFile(libraryFile, "rw");
```

```
RandomAccessFile librarynew = new RandomAccessFile(oldLibraryFile, "r");
String zeile = "      \r\n";
int i;
for (i = 1; i <= 6; i++) {
    librarynew.readLine();
}
libraryappend.seek(libraryappend.length() - 12);
while (zeile != null) {
    zeile += "\r\n";
    libraryappend.writeBytes(zeile);
    zeile = librarynew.readLine();
    libraryappend.seek(libraryappend.length());
}
```

8. Systemnamen aus der konvertierten Datei *systemFile* auslesen, diese Datei in ein Verzeichnis dieses Namens verschieben und selbst in `<Name des Systems>.mml` umbenennen.

9. Modul-/Primitivnamen aus der konvertierten Datei *modulesFile* auslesen, diese Datei auf einzelne Moduldateien mit dem Modulnamen als Dateinamen aufteilen und Verzeichnisse mit den Modulnamen anlegen, wohin die aufgeteilten MML-Dateien verschoben werden. Dies geschieht über zwei ineinander verschachtelte Schleifen, wobei die äußere Schleife für die Erstellung der Moduldateien verantwortlich ist, während die innere Schleife genutzt wird, um diese Moduldateien zeilenweise mit dem korrekten Inhalt zu füllen.

10. Ausgabe einer Erfolgsmeldung; sollte an einer beliebigen Stelle in der vorangehenden Abarbeitung ein Fehler auftreten, so wird eine Fehlermeldung *„Fehler bei der Transformation!“* ausgegeben und die Bearbeitung abgebrochen. Folglich werden auch ungültige Quelldateien erkannt und nicht transformiert.

Für die oben ausgeführte Abarbeitung wurde in der Klasse *Konverter* außerdem eine zusätzliche private Methode *FileCopy* implementiert, die ein Byte-weises Kopieren von Dateien

ermöglicht. Sie ist notwendig, da im Konvertierungsverlauf Dateien kopiert werden müssen, eine Java-interne Methode für das Kopieren aber nicht existiert und daher stets selbst implementiert werden muss.

Wie aus den erläuterten Arbeitsschritten ersichtlich ist, kann die Konvertierung von Quelldateien mehrfach erfolgen. Die konvertierten Modelle werden dann alle in eine gemeinsame Library mit dem Namen `converted_library` transformiert. Sollten diese Modelle dann Module oder Primitive mit dem gleichen Namen enthalten, so werden diese – auf Grund der Abarbeitungsweise des Konverters und der Konventionen für Datei- und Verzeichnisnamen unter allen Betriebssystemen – nur einmal angelegt. Wenn die mit einem oder mehreren Modellen gefüllte konvertierte Library dann im MLDesigner genutzt werden soll, so muss sie im nächsten Schritt exportiert werden.

7.2.3 Klassen für den Export

7.2.3.1 Klasse *Export*

Um ein konvertiertes Modell exportieren zu können, muss dieses zunächst vorhanden sein. Deshalb wird – ähnlich wie im Falle des Konvertierens – beim Export zunächst geprüft, ob konvertierte Modelle in Form des Verzeichnisses `converted_library` vorliegen. Falls nicht, endet der Export mit einer Fehlermeldung. Verläuft die Prüfung positiv, werden die eigentlichen Export-Anweisungen ausgeführt. Diese bestehen aus den folgenden vier Abarbeitungsschritten:

1. Es wird ein Datei-Speichern-unter-Programmdialog angezeigt, der – ähnlich wie im Falle des Datei-Öffnen-Dialogs in der Klasse *Durchsuchenfenster* (vgl. Abschnitt 7.2.1.2) – ein betriebssystemabhängiges *JFileChooser*-Objekt darstellt und dabei auf ein Speichern mit der Dateiendung MAR, wie vom MLDesigner gefordert, beschränkt ist:

```
JFileChooser Speichern = new JFileChooser();
Speichern.setFileFilter(new FileFilter() {
    public boolean accept(File datei) {
        return datei.isDirectory() ||
            datei.getName().toLowerCase().endsWith(".mar");
```

```
    }
    public String getDescription() {
        return "MAR-Datei";
    }
} );
```

2. Wenn der Nutzer in diesem Dialog einen von ihm spezifizierten Speicherort und Dateinamen für die MAR-Datei angegeben hat, so beginnt der eigentliche Exportvorgang. Dieser muss aus der im Konvertiervorgang angelegten Verzeichnisstruktur eine einzelne MAR-Datei generieren. Zu diesem Zweck wird zunächst eine Variable für diese MAR-Datei initialisiert und danach eine Methode zur TAR-Komprimierung des Verzeichnisses `converted_library` aufgerufen:

```
File marDatei = new File(System.getProperties().getProperty("java.io.tmpdir")
            + File.separator + "converted_library.tar");
createTAR(convertedLibrary, marDatei);
```

3. Mit der Methode *createTAR* wird zunächst eine Datei mit dem Namen `.mar.inf` angelegt, die sich in jeder in den MLDesigner zu importierenden Library befindet und Angaben über sie enthält. Da hier nur solche Informationen enthalten sind, die beim Verwenden des ARIS-MLDesigner-Konverters in fest vorgegebenen Werten bestehen, müssen keine Informationen aus Quell- oder Zieldateien extrahiert werden:

```
RandomAccessFile marInf = new RandomAccessFile
    (System.getProperties().getProperty("java.io.tmpdir")
    + File.separator + ".mar.inf", "rw");
marInf.writeBytes("MLD_VERSION 2.4.r03\r\n");
marInf.writeBytes("EXPORTED_LIBRARY file:$MLD_USER/converted_library/" +
    "converted_library.mml converted_library\r\n");
marInf.writeBytes("FILE_EXTENSION htm\r\n");
marInf.writeBytes("FILE_EXTENSION pl\r\n");
marInf.writeBytes("FILE_EXTENSION mml\r\n");
marInf.close();
```

4. In diesem Schritt erfolgt die Zusammenführung der Verzeichnisstruktur, welche im Rahmen der Modellkonvertierung(en) entstanden ist, in einer TAR-Datei. Da das

TAR-„Komprimieren“ (es stellt kein eigentliches Komprimieren dar, denn der Komprimierungsfaktor beträgt Null) nicht von Java nativ unterstützt wird, muss an dieser Stelle auf eine frei verfügbare fertige Implementierung der TAR-Algorithmen zurückgegriffen werden. Diese existiert in Form des Paketes `org.apache.tools.tar`, ein Bestandteil des Ant-Projektes [Apache Software Foundation 2006], das in Form vorkompilierter JAR-Dateien in das ARIS-MLDesigner-Konverterprojekt eingebunden wurde (siehe auch Abschnitt 7.3.1). Alle in der Verzeichnisstruktur enthaltenen Zieldateien werden somit in einen TAR-Datenstrom integriert, der in Form einer fertigen TAR-Datei `converted_library.tar` dann im temporären Verzeichnis des Betriebssystems hinterlegt wird. Dies geschieht über zwei zusätzliche Methoden *fileTAR* und *addtoTAR*, wobei *fileTAR* eine Rekursion beinhaltet, um neben allen Dateien des zu komprimierenden Verzeichnisses auch dessen Unterverzeichnisse zu erfassen:

```
for (int i = 0; i < fileList.length; i++) {
  File file = fileList[i];
  if (file == null||!file.exists()) continue;
  if (file.isDirectory()) {
    fileTAR(file, out);
    continue;
  } else {
    addtoTAR(file, out);
}
```

Die wesentliche Arbeit übernimmt dann die Methode *addtoTAR*, welche die einzelnen TAR-Einträge erzeugt und per Stream in die resultierende TAR-Datei einfügt:

```
TarEntry tarAdd = new TarEntry(addThis);
tarAdd.setModTime(addThis.lastModified());
tarAdd.setName((addThis.getAbsolutePath().substring(System.getProperties().
               getProperty("java.io.tmpdir").length(),
               addThis.getAbsolutePath().length())).replace("\\","/"));
out.setLongFileMode(TarOutputStream.LONGFILE_GNU);
out.putNextEntry(tarAdd);
FileInputStream in = new FileInputStream(addThis);
while (true) {
  int nRead = in.read(buffer, 0, buffer.length);
```

```
    if (nRead <= 0) break;
    out.write(buffer, 0, nRead);
  }
```

5. Nun kann die im TMP-Verzeichnis vorliegende fertige Datei `converted_library.tar` in das im ersten Schritt vom Nutzer angegebene Verzeichnis verschoben und dort in `<gewünschterName>.mar` umbenannt werden:

```
String speicherort = new String(Speichern.getSelectedFile().
                     getAbsolutePath());
if (!(speicherort.substring(speicherort.length() - 4,
speicherort.length()).equals(".mar"))) {
  speicherort += ".mar";
}
marDatei.renameTo(new File(speicherort));
new TempDateienLoeschen();
```

   Mit dem am Ende ausgeführten Löschen des Verzeichnisses `converted_library` durch ein Objekt der Klasse *TempDateienLoeschen* (vgl. folgender Abschnitt) ist dann der Export von konvertierten Dateien abgeschlossen.

7.2.3.2 Klasse *TempDateienLoeschen*

In den Bereich des Exports fällt schließlich das Löschen der nunmehr in eine MAR-Datei zusammengefassten und gespeicherten `converted_library`, die noch im unkomprimierten Format als Verzeichnisstruktur im TMP-Verzeichnis des Betriebssystems vorliegt. Hierzu wird innerhalb der diese Aufgabe erfüllenden Klasse eine private Methode *deleteTree* definiert, die ein rekursives Löschen eines gesamten Verzeichnisses inklusive aller Unterverzeichnisse und eventuell enthaltenen Dateien ermöglicht:

```
private static void deleteTree(File pfad)
{
    for (File datei: pfad.listFiles())
    {
        if (datei.isDirectory()) {
```

```
            deleteTree(datei);
        }
        datei.delete();
    }
    pfad.delete();
}
```

In der auszuführenden Methode der Klasse wird dann lediglich das zu entfernende Verzeichnis `converted_library` spezifiziert und an die Methode *deleteTree* übergeben, womit das Löschen aller temporär angelegten Dateien durchgeführt ist:

```
public TempDateienLoeschen() {
    deleteTree(new File(System.getProperties().getProperty("java.io.tmpdir")
            + "/converted_library"));
}
```

Damit sind alle Klassen des Java-Bestandteils des ARIS-MLDesigner-Konverters erläutert worden. Im Folgenden kann jetzt die Gesamtstruktur und das allgemeine Verhalten des komplett implementierten Konverters beschrieben werden, um damit die Abhandlung und Dokumentation der Implementierung abzuschließen.

7.3 Konverterimplementierung

7.3.1 Konverterstruktur

Der in dieser Arbeit entwickelte und implementierte Konverter zur Übertragung von ARIS-Toolset-Modellen in das MLDesigner-Format stellt grundsätzlich ein Java-Paket mit der Bezeichnung `de.konverter.xml.aml2mml` dar. Dieses liegt als JAR-Datei vor, die auf jedem Betriebssystem, auf dem ein Java Runtime Environment installierbar ist, ausgeführt werden kann, in jedem Falle sowohl unter Windows als auch unter Linux. Die ausführlich, sowohl mit Hilfe von JavaDoc als auch mit separaten Code-Kommentaren dokumentierten Java-Klassen sind als Quellcode ebenfalls vorhanden.

Unabhängig von der vorkompilierten JAR-Version des Konverters besitzt das Programm die im Folgenden erläuterte Dateistruktur:

- Verzeichnis `ARIS-MLD-Konverter`: Hauptverzeichnis des Werkzeugs
- Unterverzeichnis `.settings`: Enthält Einstellungen für die Entwicklungsumgebung Eclipse. Damit kann der Konverter problemlos in Eclipse reintegriert und der Programmcode eingesehen bzw. bearbeitet werden.
- Unterverzeichnis `bin`: Enthält vorkompilierte Klassen mit der Dateiendung `.class` und alle zur Ausführung benötigen Dateien, also die verwendeten XSLT-Dokumente.
- Unterverzeichnis `doc`: Enthält eine HTML-Struktur (mit beliebiger Browser-Software einsehbar), die von JavaDoc erzeugt wurde und Teil der Programmdokumentation ist.
- Unterverzeichnis `jar`: Enthält den vorkompilierten Konverter als Datei `aml2mml.jar`, die unabhängig von der hier beschriebenen Verzeichnisstruktur verwendet und ausgeführt werden kann.
- Unterverzeichnis `src`: Enthält alle in Abschnitt 7.2 aufgeführten Java-Klassen des Konverters im einsehbaren Textformat.
- Unterverzeichnis `tar`: Enthält die vorkompilierte Datei `tar.jar`, die Bestandteil des Java-Paketes `org.apache.tools.tar` ist und zur TAR-„Komprimierung“ der konvertierten Modelle verwendet wird (vgl. Abschnitt 7.2.3.1).
- Unterverzeichnis `xslt`: Enthält die drei Transformations-Stylesheets `library.xsl`, `system.xsl` und `module.xsl`.

Eine grafische Veranschaulichung der Verzeichnis- und Dateistruktur des ARIS-MLDesigner-Konverters findet sich in Abb. 7.6.

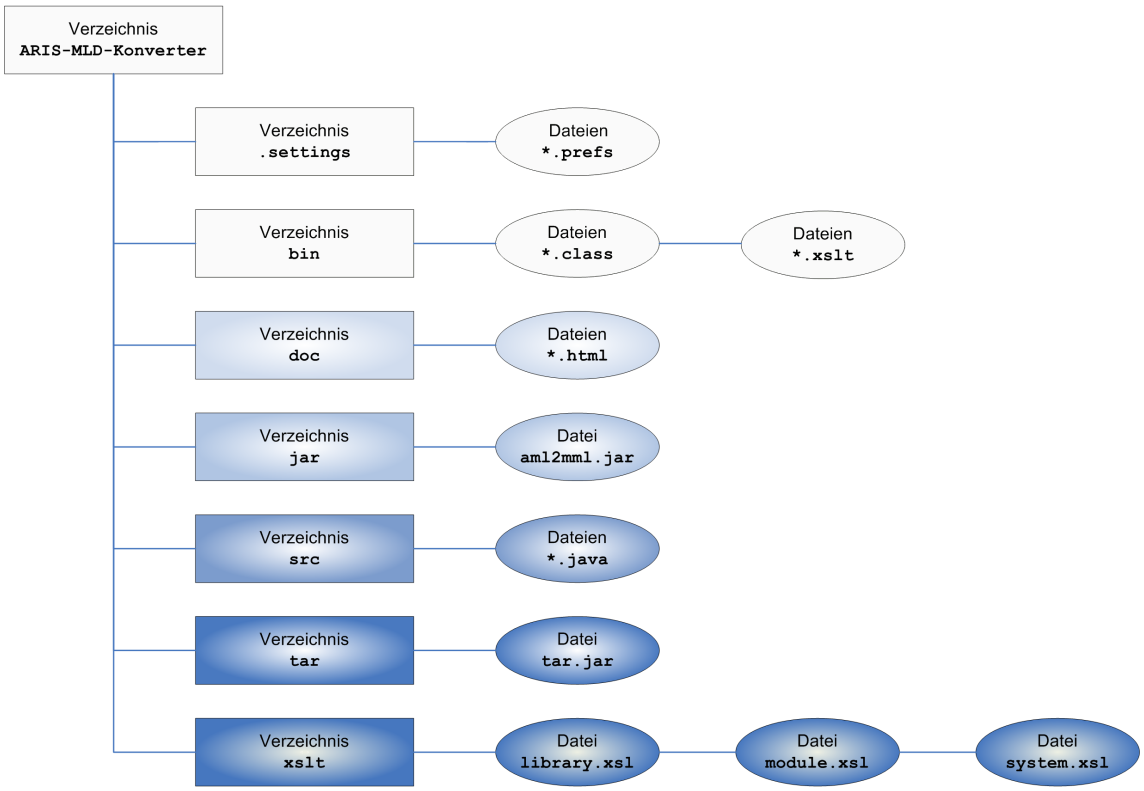

Abbildung 7.6: ARIS-MLDesigner-Konverter – Struktur der Programmdateien

7.3.2 Konverter-Ein- und Ausgabeverhalten

Der ARIS-MLDesigner-Konverter, wie er im vorherigen Kapitel definiert wurde, verwendet für die Konvertierung von ARIS-Toolset-Modellen in das MLDesigner-Library-Format AML-Dateien als Quelldateien (vgl. Abschnitt 6.2.1). Sie können im ARIS Toolset über die XML-Exportfunktion erzeugt werden und sind dann der Transformation durch XSLT-Stylesheets zugänglich. Dabei ist zu beachten, dass AML-Dateien durch eine spezielle, mit dem ARIS Toolset mitgelieferte *Document Type Definition* (DTD) validiert werden. DTDs dienen dazu, die Struktur eines XML-Dokuments festzulegen und damit eine Prüfung ihrer Gültigkeit zu ermöglichen. Da eine solche Prüfung vor dem Transformationsvorgang mit XSLT standardmäßig durchgeführt wird und jede AML-Datei in ihrem Header auf die notwendige Dokumenttypdefinition `ARIS-Export.dtd` verweist, muss diese Datei in dem Verzeichnis, aus dem die Quelldatei für den Konverter ausgewählt wird, ebenfalls vorhanden sein.

Wenn Quelldatei und DTD als Eingangsgrößen vorhanden sind und vom Nutzer spezifiziert wurden, produziert der Konverter nach dem angegebenen Verfahren eine Anzahl von Ausgabedaten im Dateiformat. Diese Ausgabedateien werden im TMP-Verzeichnis des Betriebs-

systems gespeichert, also unter Windows etwa unter dem Verzeichnispfad `C:\Dokumente und Einstellungen\<Benutzername>\Lokale Einstellungen\Temp\` oder unter Linux in `/TEMP/`.

Zu den Ausgabedaten zählt das Verzeichnis `converted_library`, in das zunächst die Ergebnisdateien der XSLT-Transformation `converted_library.mml`, `converted_system.mml` und `converted_module.mml` gespeichert werden. Im nächsten Konvertierungsschritt werden in diesem Verzeichnis dann Unterverzeichnisse mit dem Namen des Modells und mit den Namen aller Module und Primitive angelegt. Die Datei `converted_system.mml` wird in `<Name des Modells>.mml` umbenannt und in das gleichnamige Verzeichnis verschoben. Die Datei `converted_module.mml` wird auf die Anzahl der enthaltenen Module bzw. Primitive aufgeteilt, mit deren Namen versehen und ebenfalls in die gleichnamigen Verzeichnisse verschoben. Der bis hierhin beschriebene Konvertierungsablauf kann mehrfach durchlaufen werden, dabei wird dann eine Sicherungskopie der Library-Datei mit dem Namen `converted_library.mml~` angelegt.

Nach dem Anstoßen des Exportvorgangs wird das Verzeichnis `converted_library` in einen TAR-Container kopiert, der ebenfalls im TMP-Verzeichnis unter dem Namen `converted_library.tar` abgelegt wird. Nach dem Spezifizieren des Speicherortes durch den Benutzer wird diese Datei verschoben und gemäß den Wünschen des Bedienenden umbenannt.

Beim Beenden des Programms oder beim (fehlerhaften oder erfolgreichen) Export wird das Verzeichnis `converted_library` inklusive Inhalt aus dem TMP-Verzeichnis gelöscht.

Damit ist das Ein-/Ausgabeverhalten des Konverters umfassend beschrieben. Eine grafische Darstellung mit Ein- und Ausgangsdateien des ARIS-MLDesigner-Konverters bietet Abb. 7.7.

Im Folgenden kann der implementierte Prototyp einer Verifikation und Validierung unterzogen werden.

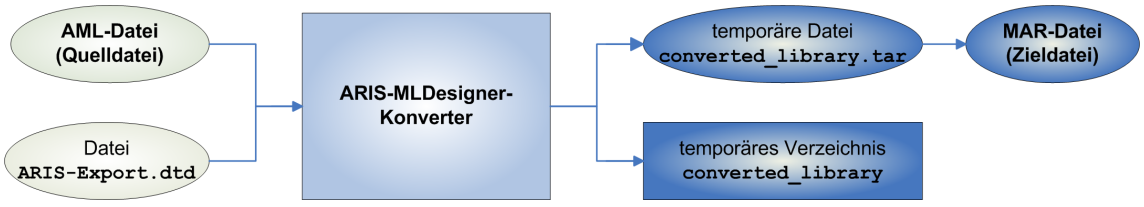

Abbildung 7.7: ARIS-MLDesigner-Konverter – Ein-/Ausgabeverhalten

8 Validierung und Leistungsbewertung

In einem letzten Arbeitsschritt soll der aus den theoretischen und praktischen Überlegungen dieser Arbeit entstandene Prototyp eines ARIS-MLDesigner-Konverters auf Gültigkeit und formale Korrektheit geprüft sowie sein derzeitiger Leistungsumfang ermittelt werden. Begonnen werden soll mit den auf der Softwaretechnik basierenden Prüfungen auf Validität und Korrektheit, wobei die Validierung ermittelt, inwieweit die Implementierung eine Umsetzung der Anforderungen ist, also ob der Konverter die gewünschte Funktion erfüllt, während die Verifikation eine Prüfung auf Übereinstimmung mit den Spezifikationen ist, also die Frage beantwortet, ob der Konverter seine Funktion korrekt erfüllt. Verbunden wird die Verifikation hier mit einer Leistungsbewertung, die den Grad der Vollständigkeit eruiert, mit welcher der Konverter die gestellte Aufgabe lösen kann. Im Rahmen dieser Leistungsbewertung werden außerdem konkrete Angaben über dem MLDesigner-Benutzer verbleibende Anpassungen konvertierter Modelle an die über den Funktionsumfang des ARIS Toolset hinaus gehenden Möglichkeiten dieses Werkzeugs, allen voran dessen Simulationsfähigkeit, geliefert. Die Kombination der hier genannten Prüfungen führt dann zu einer abschließenden Beurteilung des Prototypen.

8.1 Validierung der Elementumsetzung

In einem vorangehenden Validierungsprozess soll geprüft werden, inwieweit der realisierte Konverter einzelne Elemente eines ARIS-Toolset-Modells nach den in Abschnitt 5.2 erarbeiteten Prinzipien umzusetzen in der Lage ist. Hierzu können im ersten Schritt einfache, aus einer Funktion oder einem Ereignis bestehende ARIS-Modelle verwendet und konvertiert werden. Nach dem Import in das MLDesigner-Format liegt dann die gleiche Anzahl von Modulen vor, die entsprechende Parameter wie Name etc. der Originale besitzen. Mit dieser Überprüfung ist die korrekte Umsetzung von Ereignissen und Funktionen nachweisbar.

In einem nächsten Schritt kann ein Modell, in dem zwei Objekte (Ereignis und Funktion) mit einer Verbindung versehen werden, im ARIS Toolset erzeugt, konvertiert und in

den MLDesigner übertragen werden. Dabei wird ersichtlich, dass das konvertierte Resultat aus zwei Modulen besteht, die durch eine Kante miteinander verbunden sind. Dieser Sachverhalt weist die korrekte Umsetzung von einzelnen Modellkanten nach.

Der dritte Validierungsschritt bei der Elementüberprüfung beinhaltet die Konvertierung eines Modells, in dem einmalig eine (AND- bzw. XOR-) Regel enthalten ist. Im erzeugten MLDesigner-Modell wird an Stelle dieser Regel ein Bibliotheksprimitiv (*fork* bzw. *ProbSwitch*) eingefügt. Dieser letzte Elementarvalidierungsschritt belegt die gültige Konvertierung von ARIS-Regeln in MLDesigner-Beispielprimitive nach dem in dieser Arbeit formulierten Prinzip (vgl. Abb. 8.1).

Die grundlegende Elementvalidierung zeigt auf, dass einzelne ARIS-Modellelemente vom Konverter korrekt in das MLDesigner-Format überführt werden. Damit kann die Validierung zu komplexen Modellen, welche eine größere Anzahl an Objekten in vielfältigen Verbindungen miteinander enthalten, übergehen.

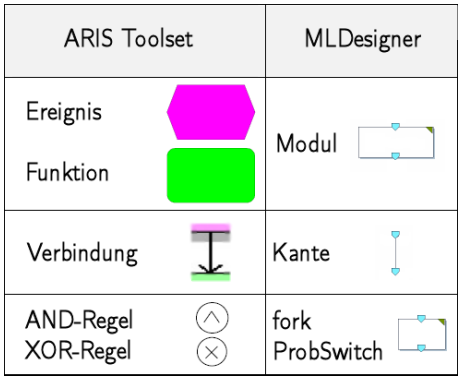

Abbildung 8.1: ARIS-MLDesigner-Konverter – Elementumsetzung

8.2 Modelle aus der klinischen Praxis als Validierungsinstrument

Für die weitere Validierung des ARIS-MLDesigner-Konverters werden bestehende Modelle aus der klinischen Praxis herangezogen, die mit dem ARIS Toolset modelliert wurden und in das MLDesigner-Format transferiert werden sollen. Entspricht das erhaltene MLDesigner-Modell der Struktur des ARIS-Toolset-Modells, so kann die Konvertierung ganzer Modelle als valide betrachtet werden.

8.2.1 Modell „Siemens Process Framework Level 2 – Treat“

Das erste Modell, das einer Umwandlung in das MLDesigner-Format unterzogen werden soll, repräsentiert eine Schicht des *Siemens Process Frameworks* zur formalen Abbildung von Krankenhausprozessen [Holzner u. Fleischer 2004], namentlich das Level 2 „Treat“, welches die konkrete Behandlung eines Patienten auf einer abstrakten Ebene beschreibt. Dieses eher einfache Modell hat, mit dem ARIS Toolset modelliert, die in Abb. 8.2 wiedergegebene Form. Erstellt wurde dieses Modell im Rahmen einer Diplomarbeit [Detschewa 2005].

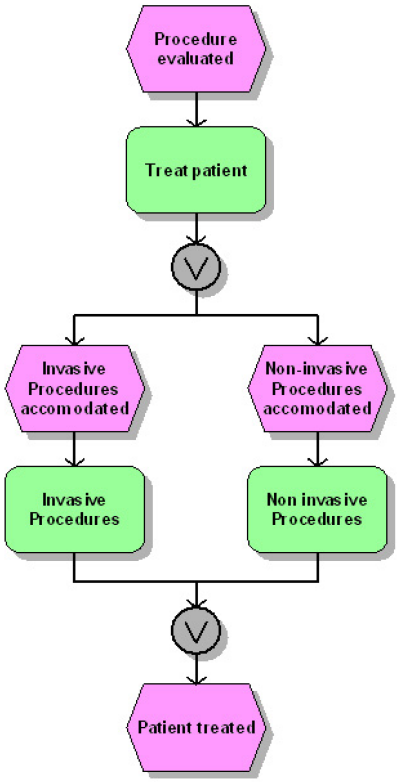

Abbildung 8.2: Siemens Process Framework Level 2 – ARIS-Toolset-Modell

Dieser modellierte und den Materialien der Diplomarbeit beiliegende Framework-Abschnitt wurde als Datenbankrücksicherung in das ARIS Toolset importiert, in das AML-Format exportiert und unter Zuhilfenahme des hier implementierten ARIS-MLDesigner-Konverters in eine Library überführt, die dann in den MLDesigner importiert werden konnte.

Nach dem Import zeigten sich drei nicht antizipierte Fehler bei der Konvertierung, für die jeweils die Ursache ermittelt werden konnte. Diese drei Fehler äußern sich folgendermaßen:

- Zusätzliche Module, die nicht im Originalmodell vorhanden waren, wurden erzeugt. Nach einer Inspektion der exportierten AML-Dateien zeigte sich, dass diese Objekte hier bereits vorhanden waren, der Konverter also korrekt arbeitete. Es scheint sich um früher verwendete, jedoch dann aus dem Modell entfernte Objekte zu handeln, die im ARIS Toolset nicht komplett gelöscht werden. Der Fehler liegt in der Datenrücksicherung des ARIS-Toolset-Modells, das ursprünglich in einem ARIS Toolset der Version 6.23 erzeugt wurde, in Verbindung mit dem sofortigen XML-Export des Toolset 7.0. (Neue, im ARIS Toolset 7 erzeugte Modelle sind nicht betroffen.)

- Sporadisch, aber reproduzierbar, wurden Kanten des Modells bei der Konvertierung ausgelassen. Eine Überprüfung ergab hier, dass diese Kanten ebenfalls bereits im AML-Format des Modells fehlten. Hier ist ebenso der XML-Export des ARIS Toolset 7.0 bei Modellen älterer Versionen fehlerbehaftet. (Wiederum sind neu erzeugte Modelle nicht vom Exportfehler betroffen.) Es bleibt zu hoffen, dass dieser und der oben genannte Fehler in zukünftigen Versionen des ARIS Toolset behoben werden können. (Hinweis: Die beiden genannten Fehler wurden bei der Verwendung eines anderen Konverters, der AML-Dateien in das EPML-Format umwandelt [Mendling u. Nüttgens 2004], ebenfalls beobachtet. Diese zusätzliche Prüfung liefert einen weiteren Beleg für die hier ursächliche Fehlerhaftigkeit des ARIS-XML-Exports älterer, rückgesicherter Modelle in Kontrast zur korrekten Umsetzung der fehlerhaften AML-Daten durch den ARIS-MLDesigner-Konverter.)

- Für die bessere Übersicht und das Vermeiden von Überdeckungen können Kanten sowohl im ARIS Toolset als auch im MLDesigner mit Eckpunkten versehen werden. Die Konzepte der Speicherstrukturen, die ein solches „Abknicken“ der Kanten ermöglichen, unterscheiden sich in den beiden Werkzeugen jedoch so grundlegend, dass im Rahmen dieser Diplomarbeit keine Möglichkeit zur Umsetzung dieser Option gefunden werden konnte. Das knotenweise „Einsammeln“ der Eigenschaften aller Modellbestandteile im AML-Format durch das XSLT-Stylesheet verhindert eine korrekte Zuordnung Kante–Eckpunkt. Versuchsweise konvertierte Eckpunkte wurden den falschen Kanten zugeordnet. Es besteht nachweislich – bedingt durch die Konvertierungsart – keine Möglichkeit, eine solche Zuordnung durchzuführen. Hierfür wären andere Konvertierungsformen anstelle von XSLT notwendig (etwa zeichenkettenbasier-

te Manipulationen), welche wiederum die Modellkonvertierung erheblich erschweren würden. Aus diesem Grunde wurde dieser dritte Fehler bei der Konvertierung, dessen Ursache klar beim Konverter selbst liegt, zugunsten der Validität aller anderen Strukturen belassen – Eckpunkte in Kanten eines ARIS-Modells fehlen daher stets nach der Konvertierung in das MLDesigner-Format.

Die drei beschriebenen Fehler treten grundsätzlich bei jedem Konvertierungsvorgang auf. Die Position der Module und Primitive, ihre Namen und Parameter und damit die allgemeine Struktur des Modells wird hingegen korrekt übertragen.

Nach einer Überarbeitung und Behebung der beiden ersten Fehler der Konvertierung, die wie erläutert nicht vom Konverter selbst verursacht werden, kann nach dem Import in den MLDesigner das in Abb. 8.3 wiedergegebene System aufgerufen und der Weiterverarbeitung zugeführt werden.

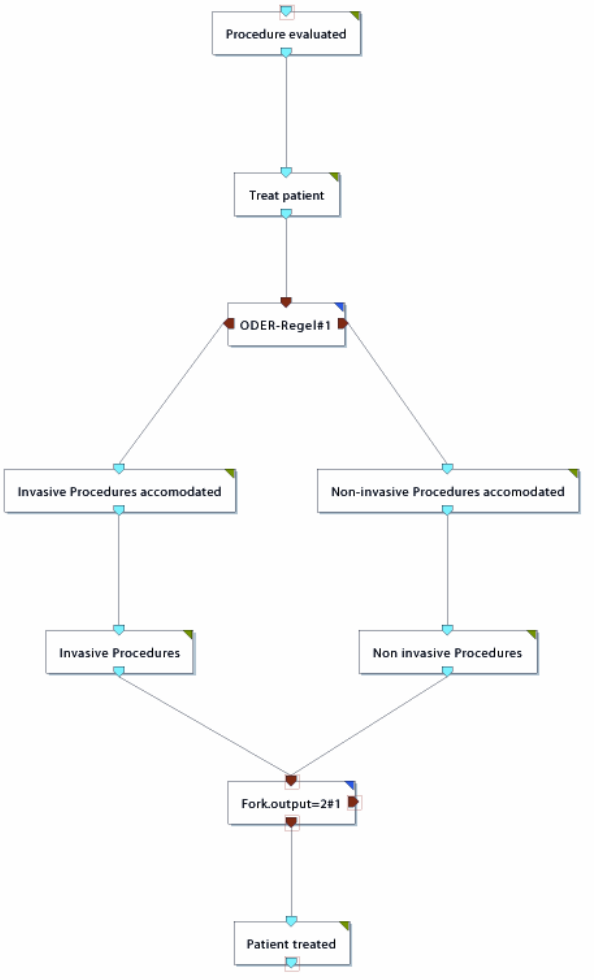

Abbildung 8.3: Siemens Process Framework Level 2 – MLDesigner-Modell

8.2.2 Modell „Therapiestandard Akutes Koronarsyndrom“

Im zweiten Validierungsschritt soll ein komplexeres Modell vom ARIS-Toolset-Format in eine MLDesigner-Library übertragen werden. Hierfür wurde ebenfalls ein Modell aus [Detschewa 2005] ausgewählt, das die Therapie des akuten Koronarsyndroms abbilden soll. Das Modell hat nach dem Importieren und Öffnen im ARIS Toolset das in Abb. 8.4 zu erkennende Erscheinungsbild.

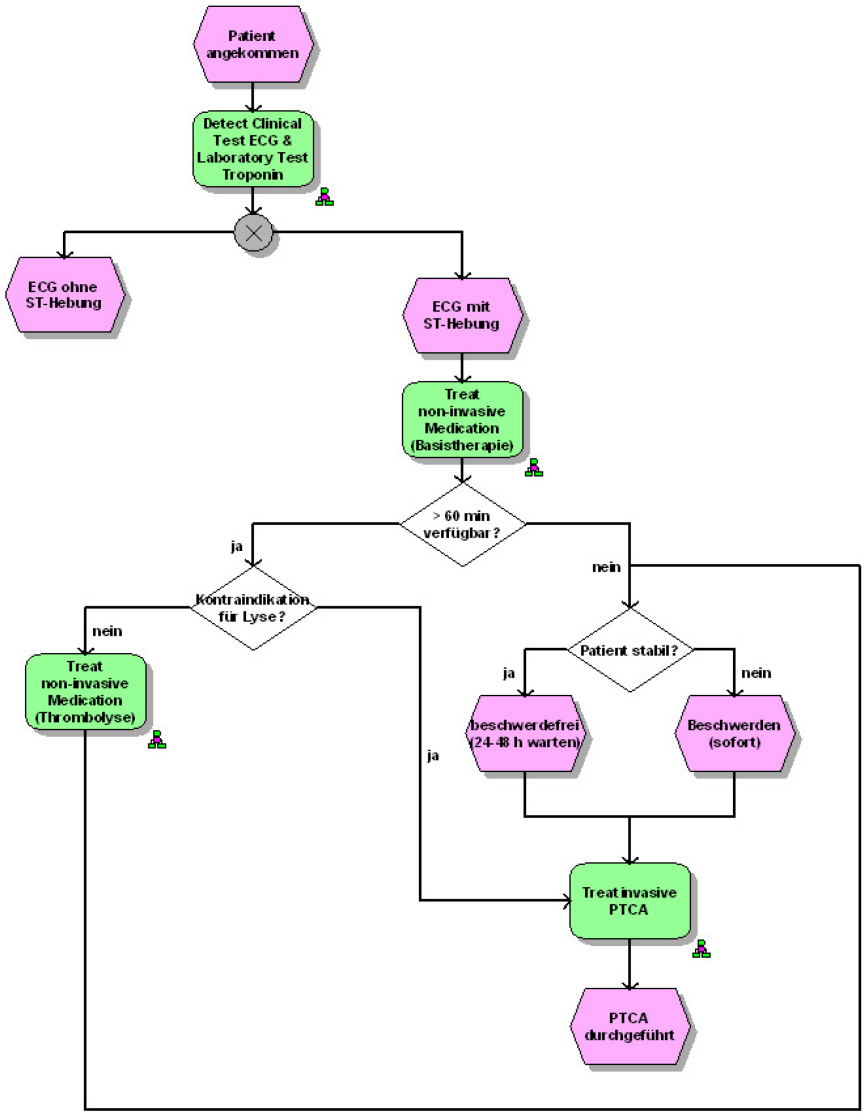

Abbildung 8.4: Therapiestandard Akutes Koronarsyndrom – ARIS-Toolset-Modell

Nach dem Export als AML-Datei und unter Anwendung des ARIS-MLDesigner-Konverters entsteht das in Abb. 8.5 dargestellte MLDesigner-Modell als Bestandteil einer converted-Library (XML-Exportfehler wurden abermals bereinigt).

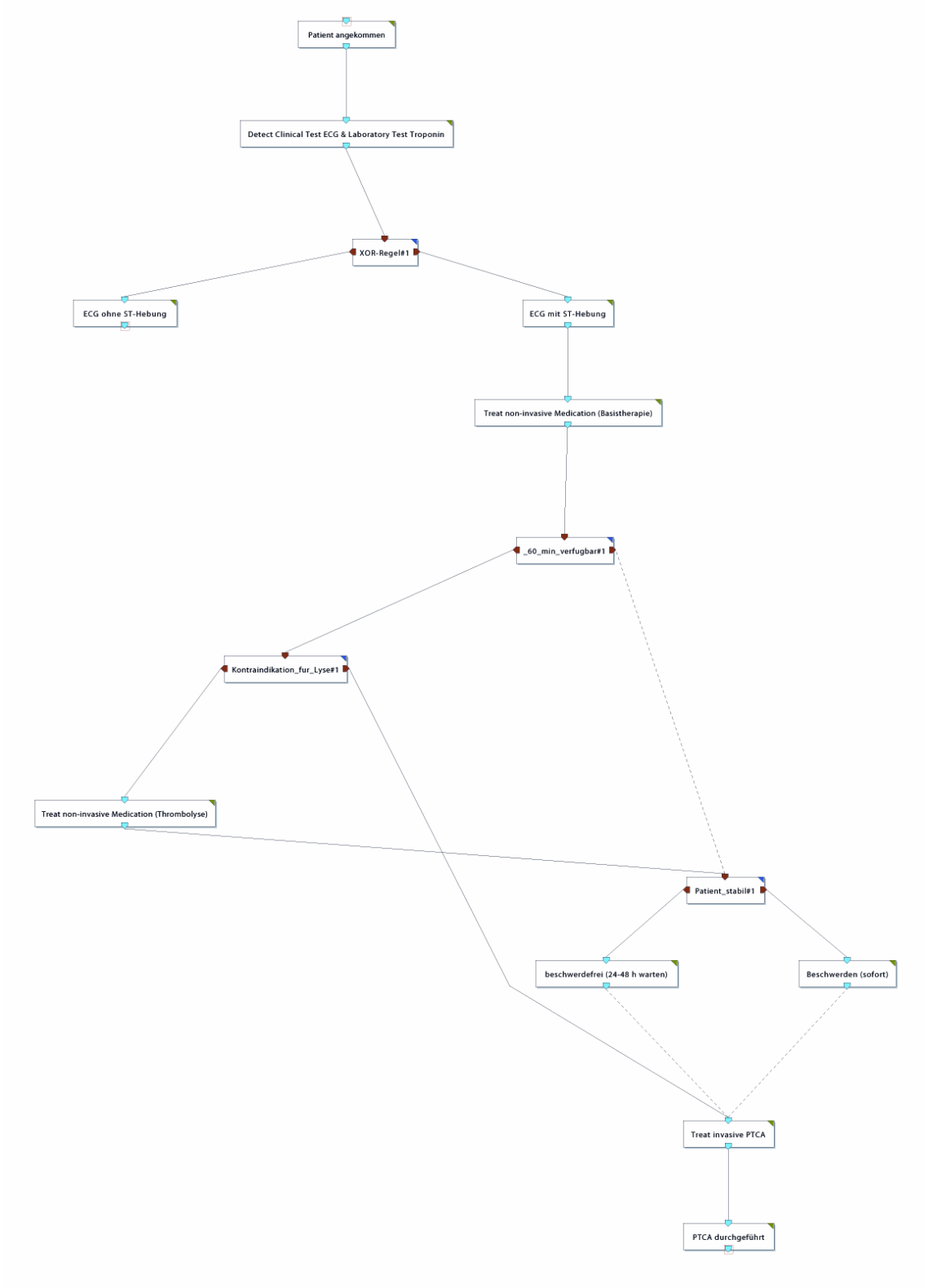

Abbildung 8.5: Therapiestandard Akutes Koronarsyndrom – MLDesigner-Modell

Es zeigt sich nach der Prüfung durch die Konvertierung der beiden bestehenden Modelle aus der klinischen Praxis, dass der Konverter abzüglich der fehlenden Eckpunkt-Umsetzung

der geforderten Validität entspricht.

Nach der Konvertierung müssen vom Benutzer noch die folgenden Schritte unternommen werden, um das erhaltene MLDesigner-Modell in das Format eines gültigen Systems zu versetzen:

1. Prüfung der fehlerhaften AML-Daten: Überschüssige Objekte müssen entfernt und fehlende Kanten ergänzt werden.

2. Prüfung aller vorhandenen In- und Outputs. Im ARIS Toolset existieren diese Strukturen nicht und können daher nicht konvertiert werden. Der Konverter erstellt für jedes Objekt genau einen Input und einen Output, Regeln erhalten zwei Outputs. Da diese Struktur nur für den Großteil aller konvertierten Objekte auch im MLDesigner zutrifft, muss der Nutzer gegebenenfalls überzählige Ports entfernen und zusätzlich notwendige Ports einfügen.

3. Prüfung der Primitivfunktion. Entsprechend dem gewünschten Verhalten des Systems bei der Simulation muss der Benutzer testen, ob die eingesetzten Beispielprimitive die angedachten Funktionen erfüllen. Gegebenenfalls sind Primitive auszutauschen.

4. Implementierung. Das konvertierte Modell ist, wie bereits erläutert, nicht simulationsfähig, da das ARIS Toolset eine Simulation im Sinne des MLDesigner nicht unterstützt und die Verwendung von eigenem Programmcode hier nicht möglich ist. Die Hinterlegung aller erzeugten Module und Primitive mit dem gewünschten Code stellt daher die Hauptarbeit des Benutzers nach dem Modellimport konvertierter ARIS-Modelle dar.

Die hier beschriebenen manuell durchzuführenden Arbeiten am konvertierten MLDesigner-Modell stellen aber keine Einschränkung der Validität des Konverters dar, sondern begründen sich in den formalen Unterschieden der Modellierungsarten der beiden zugrunde liegenden Werkzeuge. Eine automatisierte Behandlung dieser Aufgaben ist daher nicht möglich.

8.3 Verifikation und Leistungsbewertung

Im Folgenden soll der grundsätzlich als valide zu beurteilende Konverter-Prototyp einer Verifikation und Bewertung seiner Leistung unterzogen werden. Im definitorischen Sinne bezeichnet die Verifikation dabei die Nutzung einer formalen Semantik, um mathematisch die Korrektheit des Codes zu zeigen. Aufgrund der Vorgehensweise der Softwaretechnik, bei der aus Anforderungen Spezifikationen ermittelt und als Implementierung umgesetzt werden, ist eine strenge Verifikation hier nicht notwendig, da die Korrektheit aus der Erfüllung der Spezifikationen hervorgeht. Die Äquivalenz von in abstraktem Code formulierten Spezifikationen und realem Programmcode wird in dieser Arbeit in den Kapiteln 6.4, 6.5 und 7 deutlich. Die weitere Korrektheit des Codes lässt sich nun informal durch die Anwendung einer Systembewertung überprüfen, die in [Ammenwerth u. Haux 2005, S. 114–116] vorgeschlagen wird und gleichzeitig die Leistungsbewertung enthält. Es wird hierbei ein Vergleich zwischen dem Endzustand eines implementierten Werkzeugs mit der eigentlichen Aufgabenstellung verglichen. Anhand dieses konkreten Vergleichs kann dann ersichtlich werden, inwieweit die Implementierung das ursprüngliche Problem auf korrekte Art und Weise löst.

8.3.1 Analyse des Ist-Zustandes

Im ersten Schritt der Systembewertung erfolgt eine Analyse des fertigen ARIS-MLDesigner-Konverters auf seine Eigenschaften und Funktionen hin. In diesem Falle sind folgende Angaben zu tätigen:

1. Der ARIS-MLDesigner-Konverter ist eine Java-Anwendung mit XSLT-Datenzugriff.
2. Der Konverter kann aus dem ARIS Toolset exportierte AML-Dateien einlesen und in das MML-Format konvertierten.
3. Aus konvertierten Modellen ist zusammengefasst eine in den MLDesigner importierbare Library im MAR-Format generierbar.
4. Das Werkzeug ist betriebssystemunabhängig nutzbar.

8.3.2 Vergleich mit dem Soll-Zustand

Nach der Charakterisierung des fertigen Werkzeugs soll dessen Ist-Zustand mit der ursprünglichen Problemstellung verglichen werden, um eine Leistungsbewertung zu ermöglichen. Dies geschieht hier in den folgenden Punkten:

1. Das zu erstellende Werkzeug sollte in der Lage sein, bestehende ARIS-Toolset-Modelle und dessen Objekte aus diesem System zu extrahieren: Diese Funktion wird vom ARIS Toolset selbst bereit gestellt. Mit dem XML-Export des Modellierungswerkzeugs können Modelle separat abgespeichert werden. Der ARIS-MLDesigner-Konverter besitzt eine Schnittstelle zum Einlesen der aus dem XML-Export resultierenden AML-Dateien und hat auf diese Weise die Möglichkeit, auf ARIS-Toolset-Modelle zuzugreifen.

2. Die ARIS-Toolset-Modelle sollen korrekt und vollständig in das MLDesigner-Format konvertiert werden können: Dies wird durch die Modelltransformation mit XSLT erreicht, die aus den AML-Dateien eine MML-Library-Struktur generieren kann. Damit ist nicht nur die Konvertierungsmöglichkeit an sich gegeben, es besteht auch die Option, gleich mehrere Modelle in einer neuen MLDesigner-Library zusammengefasst zu exportieren. Der ARIS-MLDesigner-Konverter erfüllt die Konvertierungsaufgabe aus dieser Sicht. Anzumerken ist mit Blick auf die Korrektheit und Vollständigkeit, dass der Konverter zum einen Fehler des XML-Export-Prozesses nicht beheben kann und zum andern eine Konvertierung von mehrfach „abgeknickten“ Kanten nicht möglich ist. Die allgemeine Struktur eines jeden ARIS-Toolset-Modells wird aber korrekt und vollständig übertragen.

3. Die Funktionsfähigkeit des Konverters soll prinzipiell gegeben sein: Zu diesem Zweck wurde auf betriebssystemunabhängige Programmiersprachen zurückgegriffen, die es ermöglichen, den Konverter sowohl unter Linux oder Solaris als auch unter Windows auszuführen. Hintergrund ist hierbei die ARIS-Toolset-Bindung an Windows, während der MLDesigner nur auf UNIX-Betriebssystemen lauffähig ist.

Anhand dieser drei Punkte zeigt sich, dass der hier entworfene und realisierte Prototyp eines ARIS-MLDesigner-Konverters die gestellten Anforderungen vollständig umzusetzen vermag. Einziges Manko bleiben die fehlerhaften AML-Eingabedaten und die nicht übertragbaren Modelleigenschaften Portanzahl und Kantenabbiegungen.

8.4 Abschließende Beurteilung

Im vorliegenden Kapitel wurde die Validität und Korrektheit des implementierten ARIS-MLDesigner-Konverters geprüft und der Prototyp einer Leistungsbewertung unterzogen. Dabei wurde ersichtlich, dass der Konverter sowohl die gestellten Anforderungen erfüllte als auch korrekt implementiert war. Die auftretenden Konvertierungsfehler wurden mit Ausnahme der Eckpunkte als Ursache einer inkorrekten XML-Exportfunktion, die sich nur bei Rücksicherung älterer Modelldaten bemerkbar macht, des ARIS Toolset identifiziert.

Für Weiterentwicklungen des ARIS-MLDesigner-Konverters sind daher verschiedene Verbesserungen denkbar. Zum einen kann die noch fehlende Konvertierung der „abgeknickten“ Kanten realisiert werden. Darüber hinaus ist in späteren MLDesigner-Versionen vielleicht ein Beispiel-Primitiv vorhanden, das sich zur Abbildung der OR-Regel aus dem ARIS-Modelltyp eignet (vgl. Abschnitt 5.2.3).

Ein weiteres Verbesserungsfeld liegt in einer möglichen Analyse von AML-Daten auf Fehlerhaftigkeit. Überzählige Module und fehlende Kanten und Ports sind so zu vermeiden. Diese Aufgabe ist hinfällig, sobald der XML-Export im ARIS Toolset in einer neueren Version von seiner Fehlerträchtigkeit entbunden wird.

Schließlich besteht die Möglichkeit der Anbindung einer in der Entwicklung befindlichen MLDesigner-Modulbibliothek an den Konverter. Diese und weitere Aufgaben im Zusammenhang mit der Erstellung eines ARIS-MLDesigner-Konverters werden im Kapitel 10 behandelt. Zuvor sollen die hier geleisteten Arbeiten im folgenden Kapitel zusammenfassend wiedergegeben werden.

9 Zusammenfassung

Die vorliegende Arbeit leistet einen Beitrag zum informationstechnisch unterstützten Qualitäts- und Workflow-Management in der Medizin. Zu diesem Aufgabenbereich zählt die computergestützte Modellierung und Simulation klinisch-administrativer Prozesse. Hierfür existieren diverse Softwaresysteme, von denen im vorliegenden Falle das ARIS Toolset und der MLDesigner von Interesse sind. Aufgabe dieser Arbeit war es, mit dem ARIS Toolset erstellte Prozessmodelle automatisiert und konsistent in den MLDesigner zu übertragen.

Zu diesem Zweck werden in einem ersten Arbeitsschritt das ARIS Toolset als Modellierungswerkzeug, die von ihm angebotenen Modellformen und Modellierungselemente näher analysiert. Als betriebswirtschaftlich-informationstechnisches Werkzeug zum Unternehmens- und Prozessmanagement ist das ARIS Toolset ein langjährig etabliertes Softwaresystem mit einer Vielzahl an angebotenen Modelltypen zur Beschreibung von Sachverhalten und zeitlichen Abläufen im Unternehmensumfeld. Als für die hier zu lösende Aufgabe relevant stellt sich bei der Analyse dieser Modelltypen die Gruppe aus der Steuerungssicht heraus, welche alle anderen Sichten auf das Unternehmen mit Hilfe von Modellen für zeitliche Abläufe in einer prozessbasierten Modellierungsform zusammenführt. Aus dieser Gruppe lassen sich explizit die Vorgangskettendiagramme, Wertschöpfungskettendiagramme und die erweiterten ereignisgesteuerten Prozessketten (eEPK) als für eine Konvertierung in das MLDesigner-Format grundlegend identifizieren.

In der Analyse der Elemente dieser Modelltypen stellen sich Ereignisse, Funktionen und Regeln als wichtigste Objekttypen, die für die Konvertierung separat zu betrachten sind, heraus. Nach der Identifikation der relevanten ARIS-Modelltypen und Modellierungselemente erfolgt als nächster Schritt die Analyse des MLDesigner-Modellierungsverfahrens und seiner Modellelemente.

Der MLDesigner ist ein auf systemtheoretischen Prinzipien und Methoden basierendes Modellierungs- und Simulationswerkzeug. Damit weist der MLDesigner gegenüber dem ARIS Toolset zum einen den weitaus höheren Abstraktionsgrad und die generischeren Modellierungselemente, jedoch auch einen erweiterten Funktionsumfang mit einer umfassenden

Parametrisierungs- und Simulationsumgebung für erstellte Modelle auf. Diese Eigenschaften spiegeln sich in der Analyse der Modellelemente wider, welche Module als abstrakt-rekursiv definierte Systembestandteile und Primitive als konkrete datenverarbeitende Codeabschnitte enthalten. Die hier ermittelten Aspekte sind bei einer Analyse der möglichen Umsetzung von ARIS-Modellen als MLDesigner-Derivate zugrunde zu legen.

Der Vergleich von ARIS-Toolset-Modellen und hier speziell von eEPK mit dem MLDesigner-Modelltyp erbringt eine vergleichbare Abstraktion der Modellelemente Ereignis, Funktion (ARIS) und Modul (MLDesigner) sowie eine vergleichbare Konkretisierung in den Elementen (Boole'sche) Regel (ARIS) und Primitiv (*fork* bzw. *ProbSwitch*, MLDesigner). Daraus kann die allgemeine Umsetzungsmethode von Modellen, die mit dem ARIS Toolset erstellt wurden, in das MLDesigner-Format entwickelt werden.

Auf Basis der vorangehenden Untersuchungen folgt eine softwaretechnisch getriebene Analyse der Einzelsysteme, um ein Datentransformationsmodell zu etablieren, das die oben beschriebene Konvertierungsmethode auf der Betriebssystemebene ermöglicht. Dabei sind die unterschiedlichen Betriebssysteme zu beachten, unter denen die zu verbindenden Modellierungswerkzeuge lauffähig sind (ARIS Toolset – Windows, MLDesigner – Linux). Als Lösungsmöglichkeit für das Extrahieren von Modelldaten aus dem ARIS Toolset und für das Einbringen dieser Daten in den MLDesigner zeichneten sich auf der einen Seite der ARIS-XML-Export und auf der anderen Seite die Möglichkeit zum Import von Librarys in den MLDesigner, die stets ebenfalls in einem XML-basierten Format vorliegen, ab.

Mit der Ermittlung dieser Anforderungen an einen Konverter, der ARIS-Modelle in das MLDesigner-Format transferiert, können im nächsten Schritt eindeutige Spezifikationen für ein solches Werkzeug abgeleitet werden: Es muss sich hauptsächlich um ein betriebssystemunabhängiges Werkzeug zur Transformation von XML-Daten handeln. Dem sich anschließenden Entwurf eines solchen Werkzeugs zufolge sollen die XML-Transformationssprache XSLT und die betriebssystemunabhängige Programmiersprache Java verwendet werden, um Funktionalität bzw. Benutzerschnittstelle des Konverters anforderungsgetreu zu implementieren. Für die Strukturierung des Systementwurfs und der Realisierung wird der Konverter durch eine Schichtenarchitektur mit den drei Ausführungen Betriebssystemschicht, Anwendungsschicht und Benutzerschicht charakterisiert.

In der Implementierungsphase wird die Modelltransformation mit Hilfe dreier XSLT-Style-

sheets für die separate Behandlung von Library, Modell und enthaltenen Modellelementen als Realisierung der Anwendungsschicht durchgeführt. Außerdem wird die Betriebssystemschicht, welche Ein- und Ausgabe der Modelldaten sowie zusätzliche Datenkonvertierungen ermöglichen soll, mit der Implementierung in vier Java-Klassen realisiert. Schließlich werden drei weitere Java-Klassen, welche für die Präsentation eines Nutzerinterfaces am Bildschirm verantwortlich sind, verwendet.

Mit dem Entwurf und der Implementierung eines Werkzeugs, das aus dem ARIS Toolset exportierte Modelle in das MLDesigner-Format übertragen kann, ist die zentrale Aufgabe der vorliegenden Arbeit gelöst. In einem letzten Arbeitsschritt wird der fertige Prototyp, eine Java-Applikation mit XSLT-Anbindung, einem Leistungstest unterzogen, indem er bestehende Modelle aus der klinischen Praxis einer suffizienten Transformation unterzieht, womit seine Funktionsfähigkeit und die Lösungsmöglichkeit der gestellten Aufgabe bestätigt werden können.

Mit der Rekapitulation des angewandten Vorgehens, das der etablierten Top-Down-Methode (vom Allgemeinen zum Speziellen) folgt und von der Betrachtung des Aufgabengebietes „Klinisches Prozessmanagement durch Modellierung“ über die Analyse zweier Modellierungswerkzeuge sowie die Schaffung einer Umsetzungsmethode ihrer spezifischen Modellelemente und daraus ableitend dem Entwurf eines Modelltransferwerkzeugs und der Beschreibung seiner Anforderungen und Spezifikationen hin zur konkreten Implementierung eines ARIS-MLDesigner-Konverters führt, sowie einem im nächsten Kapitel vorgenommenen Ausblick auf das weitere Aufgabenfeld im hier betrachteten Bereich erfolgt der Abschluss der vorliegenden Arbeit.

10 Ausblick

Der in dieser Arbeit prototypisch implementierte Konverter zur Übertragung von ARIS-Toolset-Modellen in den MLDesigner bietet eine Grundlage für weitere Entwicklungen auf dem Gebiet zur Modelltransformation für klinische Prozesse. Eine Anzahl sich angrenzender Aufgabenfelder ist mit der ersten Realisierung eines solchen Konverters verbunden.

So ist derzeit an der Technischen Universität Ilmenau ein Projekt in der Entwurfsphase, das die Zusammenstellung und Realisierung einer vorgefertigten Modulbibliothek speziell für die Modellierung klinischer Prozesse mit dem MLDesigner zum Inhalt hat. Für den Funktionsumfang des ARIS-MLDesigner-Konverters wäre daher eine Erweiterung denkbar, die bei der Modellumsetzung prüft, ob und inwieweit die Ersetzung von bestimmten ARIS-Modellobjekten durch spezifische Module der klinischen Modulbibliothek möglich ist und eine solche dann entsprechend durchführt. Eine solche Funktionalität würde sowohl auf die Konsistenz der Konvertierung als auch auf den Erweiterungsaufwand des resultierenden MLDesigner-Modells (der auf Grund der fehlenden Simulationsmöglichkeit im ARIS Toolset und des systemnäheren, abstrakteren Modellierungsverfahrens im MLDesigner stets notwendig sein wird) positiven Einfluss nehmen.

Eine weitere Aufgabe besteht in der Rücktransformation von MLDesigner-Modellen in das ARIS-Format. Basierend auf den in dieser Arbeit vorgenommenen Analysen wären für einen solchen Konverter die gleichen Prinzipien – Modelltransformation mit XSLT, eingebettet in eine Java-Applikation – anwendbar. Eine Rücktransformation leistet darüber hinaus einen Beitrag zur Re-Abstraktion von einer konkreten Modellerstellung mit Hilfe einer speziellen Software, die meist mit dem Vorliegen einer weitaus breiter gefächerten Anwendung zugänglicher Daten in einem vorher proprietären und damit prinzipiell für viele Arbeitsbereiche unzugänglichen Format einhergeht.

Die Konsequenz aus dem Problem von in herstellerspezifischen Formaten vorliegenden Prozessmodellen ist die alternative Kodierung solcher Modelle in einem standardisierten Format. Zu diesem Zweck wurde das auf XML basierende Datenformat EPML entwickelt, das speziell den Modelltyp der ereignisgesteuerten Prozesskette einem offenen und damit für

alle Zwecke zugänglichen Datenformat zuführt (vgl. Abschnitt 4.3.1). Ein Konverter, der ARIS-Toolset-Modelle in dieses standardisierte Format überführt, liegt bereits vor [Mendling u. Nüttgens 2004]. Mit der Implementierung einer Transformationsmöglichkeit von in EPML vorliegenden Modellen in das MLDesigner-Format und umgekehrt würde zum einen ein zusätzlicher Beitrag zur Standardisierung von Modelldatenformaten geleistet; zum anderen würde sich durch die Existenz weiterer Konvertierungssoftware, welche Daten aus anderen Modellierungswerkzeugen in das EPML-Format überträgt, die Anzahl der in das MLDesigner-Format konvertierbaren Modelltypen und Modelle zusätzlich um ein Vielfaches erhöhen.

Es zeigt sich, dass es sich bei der Dichotomie zwischen abstrakten und allgemein anwendbaren Modelltypen, die für die Unternehmensprozessdarstellung spezifiziert wurden, und der proprietären Datenspeicherung, welche in den meisten etablierten Modellierungswerkzeugen angewandt wird, um ein generelles Problem handelt, für das die vorliegende Arbeit nur eine geringe Teillösung bieten kann. Alle Standardisierungsbestrebungen in diesem Bereich als auch bereits vorgeschlagene offene Datenformate für die Modellspeicherung sind daher zu begrüßen und in zukünftige Arbeiten auf dem Gebiet der klinischen Prozessmodellierung und -simulation zu integrieren.

Anhang: Inhalt der beiliegenden CD-ROM

Auf der dieser Arbeit beiliegenden CD-ROM sind folgende Materialien abgelegt:

- Prototyp des ARIS-MLDesigner-Konverters
  - Verzeichnis `ARIS-MLDesigner-Konverter`
- Quellcode des ARIS-MLDesigner-Konverters
  - Verzeichnis `ARIS-MLDesigner-Konverter`, Unterverzeichnis `SRC`
  - Dateien `Beenden.java`
    `Durchsuchenfenster.java`
    `Export.java`
    `Hauptfenster.java`
    `Hilfefenster.java`
    `Konverter.java`
    `TempDateienLoeschen.java`
- ausführbare JAR-Datei des ARIS-MLDesigner-Konverters
  - Verzeichnis `ARIS-MLDesigner-Konverter`, Unterverzeichnis `JAR`
  - Datei `aml2mml.jar`
- XSLT-Dokumente des ARIS-MLDesigner-Konverters
  - Verzeichnis `ARIS-MLDesigner-Konverter`, Unterverzeichnis `XSLT`
  - Dateien `library.xsl`
    `module.xsl`
    `system.xsl`
- schriftliche Ausarbeitung im PDF-Format
  - Datei `D.Ammon - Diplomarbeit.pdf`
- Abschlussvortragsfolien im PPT-Format
  - Datei `D.Ammon - Diplomarbeit.ppt`

Literaturverzeichnis

Altova GmbH 2005

Altova GmbH: *Altova XMLSpy 2005*. `http://www.altova.com/de/products_ide.html`. – Online–Ressource, Abruf: 22. Mai 2006

Ammenwerth u. Haux 2005

Ammenwerth, Elske (Hrsg.) ; Haux, Reinhold (Hrsg.): *IT-Projektmanagement in Krankenhaus und Gesundheitswesen*. Stuttgart : Schattauer GmbH, 2005. – ISBN 3–7945–2416–0

Apache Software Foundation 2006

Apache Software Foundation: *The Apache Ant Project*. `http://ant.apache.org`. – Online–Ressource, Abruf: 16. April 2006

Assisi 2004

Assisi, Ramin: *Eclipse : Einführung und Referenz*. München : Carl Hanser Verl., 2004. – ISBN 3–446–22620–6

Bass et al. 2002

Bass, Len ; Clements, Paul ; Kazman, Rick: *Software Architecture In Practice*. 9. Aufl. Boston : Addison-Wesley Verl., 2002. – ISBN 0–201–19930–0

von Bertalanffy 1969

Bertalanffy, Ludwig von: *General System Theory : Foundations, Development, Applications*. 2. Aufl. New York : Braziller Verl., 1969

Bongers 2004

Bongers, Frank: *XSLT 2.0 : Das umfassende Handbuch*. Bonn : Galileo Press GmbH, 2004. – ISBN 3–89842–361–1

Bott 2001

Bott, Oliver J.: *Simulation medizinischer Prozesse: Der Braunschweiger Kran-*

kenhaussimulator. http://dwarf.umi.cs.tu-bs.de/full/research/mis/mosaikm/mosaikm_kurzdarstellung.html. – Online–Ressource, Abruf: 22. Mai 2006

Burke 2002

Burke, Eric M.: *Java und XSLT*. Köln : O'Reilly Verl., 2002. – ISBN 3–89721–295–1

Büyükyilmaz u. Forbrig 2003

Büyükyilmaz, Ahmet ; Forbrig, Peter: *Unternehmensportale in Verbindung mit ARIS*. http://e-lib.informatik.uni-rostock.de/fulltext/2003/misc/Forbrig-EM-2003.pdf. – Online–Ressource, Abruf: 22. Mai 2006

Chen 1976

Chen, Peter Pin-Shan: The Entity-Relationship Model : Toward a Unified View of Data. In: *ACM Transactions on Database Systems* 1 (1976), Nr. 1, S. 9–36

Chen u. Scheer 1994

Chen, Rong ; Scheer, August-Wilhelm: *Modellierung von Prozessketten mittels Petri-Netz-Theorie*. Saarbrücken : IWi, 1994 (Veröffentlichungen des Instituts für Wirtschaftsinformatik (IWi) im Institut für Empirische Wirtschaftsforschung an der Universität des Saarlandes 107)

Davis 2001

Davis, Rob: *Business Process Modelling with ARIS : A Practical Guide*. London : Springer-Verl., 2001. – ISBN 1–85233–434–7

Detschewa 2005

Detschewa, Diana: *Modellierung ausgewählter klinischer Behandlungspfade auf Basis des Siemens Process Framework*, Ilmenau, Techn. Univ., Diplomarbeit, 2005

Donabedian 1986

Donabedian, Avedis: Criteria and Standards for Quality Assessment and Monitoring. In: *Quality Review Bulletin* 12 (1986), S. 9–100

Eclipse Foundation 2006

Eclipse Foundation: *Eclipse*. http://www.eclipse.org. – Online–Ressource, Abruf: 22. Mai 2006

von Eiff u. Ziegenbein 2001a

Eiff, Wilfried von ; Ziegenbein, Ralf: Entwicklung von Prozessmodellen im Krankenhaus. In: Eiff, Wilfried von (Hrsg.) ; Ziegenbein, Ralf (Hrsg.): *Geschäftsprozessmanagement : Methoden und Techniken für das Management von Leistungsprozessen im Krankenhaus.* Gütersloh : Verl. Bertelsmann Stiftung, 2001 (Leistungsorientierte Führung und Organisation im Gesundheitswesen 4), S. 55–81. – ISBN 3–89204–597–6

von Eiff u. Ziegenbein 2001b

Eiff, Wilfried von ; Ziegenbein, Ralf: Prozessmanagement im Krankenhaus auf Basis Klinischer Prozessbibliotheken. In: Eiff, Wilfried von (Hrsg.) ; Ziegenbein, Ralf (Hrsg.): *Geschäftsprozessmanagement : Methoden und Techniken für das Management von Leistungsprozessen im Krankenhaus.* Gütersloh : Verl. Bertelsmann Stiftung, 2001 (Leistungsorientierte Führung und Organisation im Gesundheitswesen 4), S. 135–160. – ISBN 3–89204–597–6

Eisentraut 2004

Eisentraut, Katja: *Abbildung und Simulation eines klinischen Prozesses mit dem MLDesigner am Beispiel des akuten Koronarsyndroms*, Ilmenau, Techn. Univ., Diplomarbeit, 2004

Franklin et al. 2002

Franklin, Gene F. ; Powell, J. D. ; Emami-Naeini, Abbas: *Feedback Control of Dynamic Systems.* 4. Aufl. New Jersey : Prentice Hall, Inc., 2002. – ISBN 0–13–098041–2

Gadatsch 2003

Gadatsch, Andreas: *Grundkurs Geschäftsprozess-Management : Methoden und Werkzeuge für die IT-Praxis.* 3., verb. und erw. Aufl. Wiesbaden : Vieweg Verl., 2003. – ISBN 3–528–25759–8

Girod et al. 2005

Girod, Bernd ; Rabenstein, Rudolf ; Stenger, Alexander: *Einführung in die Systemtheorie : Signale und Systeme in der Elektrotechnik und Informationstechnik.* 3., korr. Aufl. Wiesbaden : B. G. Teubner Verl., 2005. – ISBN 3–519–26194–4

Gumbel et al. 2005

Gumbel, Markus ; Grebe, Reinhard ; Knapp-Mohammady, Michaela ; Ullmann, Matthias ; Langowski, Jörg: Modellierung biologischer Prozesse. In: Lehmann, Thomas M. (Hrsg.): *Handbuch der Medizinischen Informatik.* München : Carl Hanser Verl., 2005, S. 197–251. – ISBN 3–446–22701–6

Haas 2005

Haas, Peter: *Medizinische Informationssysteme und Elektronische Krankenakten.* Berlin : Springer-Verl., 2005. – ISBN 3–540–20425–3

Hauguth 2000

Hauguth, Maik: *Entwurf eines XML-basierten Dokumentenformates für Blockdiagramme*, Ilmenau, Techn. Univ., Diplomarbeit, 2000

Heidenberger 1990

Heidenberger, Kurt: Simulation. In: Seelos, Hans-Jürgen (Hrsg.): *Wörterbuch der Medizinischen Informatik.* Berlin : Walter de Gruyter Verl., 1990, S. 452 f. – ISBN 3–11–011224–8

Holzner u. Fleischer 2004

Holzner, Andreas ; Fleischer, Maik: *Health Care Enterprise / Organization Business Process Framework V 1.00.* 2004. – Foliensammlung (unveröff.)

HRW Consulting Factory AG 2006

HRW Consulting Factory AG: *BPM-Converter.* `http://www.hrw-consulting.com/produkte/bpm-converter.html`. – Online–Ressource, Abruf: 22. Mai 2006

IDS Scheer AG 2000

IDS Scheer AG: *ATS1 ARIS Grundlagenschulung : Kurs für Einsteiger ARIS 5.0.* IDS Scheer AG, 2000

IDS Scheer AG 2005a

IDS Scheer AG: *ARIS Platform : Methodenhandbuch ARIS 7.0.* Juni 2005. IDS Scheer AG, 2005

IDS Scheer AG 2005b

IDS Scheer AG: *ARIS Platform : Produktbroschüre.* Juni 2005. IDS Scheer AG, 2005

IDS Scheer AG 2005c

IDS Scheer AG: *ARIS Platform : Schnittstellenbeschreibung XML-Export/Import ARIS 7.0.* Juni 2005. IDS Scheer AG, 2005

Jackson 1995

Jackson, Michael: *Software Requirements & Specifications : A Lexicon of Practice, Principles and Prejudices.* Harlow : Addison-Wesley Verl., 1995 (ACM Press Books). – ISBN 0–201–87712–0

Jesse 2001

Jesse, Ralf: *Swing : Swing-Komponenten, Layout-Manager, Ereignisse, Threads.* Kaarst : bhv Verl., 2001 (bhv programmierung). – ISBN 3–8287–2055–2

Keller et al. 1992

Keller, Gerhard ; Nüttgens, Markus ; Scheer, August-Wilhelm: *Semantische Prozessmodellierung auf der Grundlage „Ereignisgesteuerter Prozessketten (EPK)“.* Saarbrücken : IWi, 1992 (Veröffentlichungen des Instituts für Wirtschaftsinformatik (IWi) im Institut für Empirische Wirtschaftsforschung an der Universität des Saarlandes 89)

Kühn 2006

Kühn, Matthias: *Simulation und Optimierung einer Tagesklinik : Simulationsstudie mit dem MLDesigner*, Ilmenau, Techn. Univ., Diplomarbeit, 2006

Kunhardt et al. 2005

Kunhardt, Horst ; Dannert, Elisabeth ; Porzsolt, Franz ; Sigle, Jörg: Medizinisches Qualitätsmanagement. In: Lehmann, Thomas M. (Hrsg.): *Handbuch der Medizinischen Informatik.* München : Carl Hanser Verl., 2005, S. 773–813. – ISBN 3–446–22701–6

Mendling u. Nüttgens 2004

Mendling, Jan ; Nüttgens, Markus: Transformation of ARIS Markup Language to EPML. In: Nüttgens, Markus (Hrsg.) ; Rump, Frank J. (Hrsg.): *Proceedings of the 3rd GI Workshop on Event-Driven Process Chains (EPK 2004).* Luxemburg : Gesellschaft für Informatik e. V. (GI), Oktober 2004, S. 27–38

MLDesign 2004a

MLDesign: *Introduction to MLDesigner.* MLDesign Technologies, Inc., 2004

MLDesign 2004b

MLDesign: *Managing Models.* MLDesign Technologies, Inc., 2004

MLDesign 2004c

MLDesign: *MLDesigner Documentation (Draft).* Version 2.5. MLDesign Technologies, Inc., 2004

Mosa 2001

Mosa, Gabriele: Standardprozessorientierter Krankenhausvergleich. In: Eiff, Wilfried von (Hrsg.) ; Ziegenbein, Ralf (Hrsg.): *Geschäftsprozessmanagement : Methoden und Techniken für das Management von Leistungsprozessen im Krankenhaus.* Gütersloh : Verl. Bertelsmann Stiftung, 2001 (Leistungsorientierte Führung und Organisation im Gesundheitswesen 4), S. 83–134. – ISBN 3–89204–597–6

Măruşter u. Jorna 2005

Măruşter, Laura ; Jorna, René J.: From Data to Knowledge : A Method for Modeling Hospital Logistic Processes. In: *IEEE Transactions on Information Technology in Biomedicine* 9 (2005), Nr. 2, S. 248–255

Pietsch-Breitfeld u. Selbmann 1997

Pietsch-Breitfeld, Barbara ; Selbmann, Hans-Konrad: Qualitätssicherung in der Medizin. In: Seelos, Hans-Jürgen (Hrsg.): *Medizinische Informatik, Biometrie und Epidemiologie.* Berlin : Walter de Gruyter Verl., 1997, S. 151–176. – ISBN 3–11–014317–8

Reischmann Informatik GmbH 2006

Reischmann Informatik GmbH: *TOOLBUS Interfaces for the ARIS Toolset.* `http://www.reischmann.com/Products/TOOLBUS_Interfaces_for_ARIS_Toolset.htm`. – Online–Ressource, Abruf: 22. Mai 2006

Reiter 2004

Reiter, Christian: *Business Process Model Converter (BPMC) : Unternehmensübergreifende BPM Plattform für ein integriertes Geschäftsprozessmanagement.* `http://download.microsoft.com/download/e/f/9/ef961d71-9570-449e-905a-87d49ec28c01/HRW_BPMC_2004011.ppt`. – Online–Ressource, Abruf: 22. Mai 2006

Reiter 2005

Reiter, Christian: *Ganzheitliches Management von Geschäftsprozessen mit Microsoft-Produkten.* http://download.microsoft.com/download/e/f/9/ef961d71-9570-449e-905a-87d49ec28c01/HRW-Visio-InfoTag.pdf. – Online–Ressource, Abruf: 22. Mai 2006

Rosemann 1996

Rosemann, Michael: *Komplexitätsmanagement in Prozessmodellen : Methodenspezifische Gestaltungsempfehlungen für die Informationsmodellierung.* Wiesbaden : Verl. Dr. Th. Gabler GmbH, 1996 (Schriften zur EDV-orientierten Betriebswirtschaft). – ISBN 3–409–12172–2. – Zugl.: Münster/Westf. Univ., Diss., 1995

Rothfuss u. Ried 2003

Rothfuss, Gunther (Hrsg.) ; Ried, Christian (Hrsg.): *Content Management mit XML : Grundlagen und Anwendungen.* 2., überarb. Aufl. Berlin : Springer Verl., 2003 (Xpert.press). – ISBN 3–540–43844–0

Schäfer et al. 1998

Schäfer, Torsten ; Bott, Oliver J. ; Dresing, Klaus ; Pretschner, Dietrich ; Stürmer, Klaus-Michael: Anforderungen an Workflow-Management-Systeme zur Unterstützung klinischer Abläufe : Erfahrungen aus einem Projekt zur Spezifikation eines rechnergestützten Informationssystems für unfallchirurgische Abteilungen. In: Greiser, Eberhard (Hrsg.) ; Wischnewsky, Manfred (Hrsg.): *Medizinische Informatik, Biometrie und Epidemiologie GMDS '98 : 43. Jahrestagung der GMDS in Bremen.* Basel : MMV Medien & Medizin Verl., September 1998, S. 162–167

Scheer 1991

Scheer, August-Wilhelm: *Architektur integrierter Informationssysteme : Grundlagen der Unternehmensmodellierung.* Berlin : Springer-Verl., 1991. – ISBN 3–540–53984–0

Scheer 1996

Scheer, August-Wilhelm: *ARIS-House of Business Engineering.* Saarbrücken : IWi, 1996 (Veröffentlichungen des Instituts für Wirtschaftsinformatik (IWi) im Institut für Empirische Wirtschaftsforschung an der Universität des Saarlandes 133)

Scheer 1997

Scheer, August-Wilhelm: *Wirtschaftsinformatik : Referenzmodelle für industrielle Geschäftsprozesse.* 7., durchges. Aufl. Berlin : Springer-Verl., 1997. – ISBN 3–540–62967–X

Scheer 1998

Scheer, August-Wilhelm: *ARIS : Vom Geschäftsprozess zum Anwendungssystem.* 3., völlig neu bearb. und erw. Aufl. Berlin : Springer-Verl., 1998. – ISBN 3–540–63835–0

Scheer 2001

Scheer, August-Wilhelm: *ARIS : Modellierungsmethoden, Metamodelle, Anwendungen.* 4. Aufl. Berlin : Springer-Verl., 2001. – ISBN 3–540–41601–3

Scheer et al. 1996a

Scheer, August-Wilhelm ; Chen, Rong ; Zimmermann, Volker: *Geschäftsprozesse und integrierte Informationssysteme im Krankenhaus.* Saarbrücken : IWi, 1996 (Veröffentlichungen des Instituts für Wirtschaftsinformatik (IWi) im Institut für Empirische Wirtschaftsforschung an der Universität des Saarlandes 130)

Scheer et al. 1996b

Scheer, August-Wilhelm ; Chen, Rong ; Zimmermann, Volker: Prozessmanagement im Krankenhaus. In: Adam, Dietrich (Hrsg.): *Krankenhausmanagement.* Lengerich/Westf. : Verl. Dr. Th. Gabler GmbH, 1996 (Schriften zur Unternehmensführung 59), S. 75–96. – ISBN 3–409–13595–2

Schorcht et al. 2003

Schorcht, Gunar ; Unger, Peter ; George, Alan ; Troxel, Ian ; Salzwedel, Horst ; Zinn, Daniel ; Farhangian, Keyvan ; Mick, Colin K.: System-Level Simulation Modeling with MLDesigner. In: *IEEE / ACM MASCOT 2003 : 11th ACM / IEEE International Symposium on Modeling, Analysis and Simulation of Computer and Telecommunication Systems.* Orlando, Oktober 2003

Stoy 2002

Stoy, Markus: *AML Interpreter.* `http://www.informatik.uni-rostock.de/~masto/aris/index.html`. – Online–Ressource, Abruf: 22. Mai 2006

Tidwell 2002

TIDWELL, Doug: *XSLT : XML-Dokumente transformieren.* Köln : O'Reilly Verl., 2002. – ISBN 3–89721–292–7

Tröbs 2006

TRÖBS, Manuela: *Analyse, Optimierung und Simulation des Simis IS Entwicklungsprozesses*, Ilmenau, Techn. Univ., Diplomarbeit, 2006

Ullenboom 2005

ULLENBOOM, Christian: *Java ist auch eine Insel.* 4. Aufl. Bonn : Galileo Press GmbH, 2005. – ISBN 3–89842–526–6

Unbehauen 2002

UNBEHAUEN, Rolf: *Systemtheorie.* Bd. 1 : Allgemeine Grundlagen, Signale und lineare Systeme im Zeit und Frequenzbereich. 8., korr. Aufl. München : Oldenbourg Verl., 2002. – ISBN 3–486–25999–7

Vad et al. 2002

VAD, Janos S. ; BEER, Daniel G. ; HEINRICH, Timo ; HUHNT, Wolfgang: Werkzeuge zur Planung der Planung. In: *14. Forum Bauinformatik.* Düsseldorf : VDI, September 2002, S. 197–204

Vonhoegen 2005

VONHOEGEN, Helmut: *Einstieg in XML.* 2., überarb. Aufl. Bonn : Galileo Press GmbH, 2005. – ISBN 3–89842–630–0

W3C 1999a

W3C: *XML Path Language (XPath) Version 1.0.* `http://www.w3.org/TR/xpath`. – Online–Ressource, Abruf: 16. April 2006. – W3C Recommendation

W3C 1999b

W3C: *XSL Transformations (XSLT) Version 1.0.* `http://www.w3.org/TR/xslt`. – Online–Ressource, Abruf: 16. April 2006. – W3C Recommendation

W3C 2004

W3C: *Extensible Markup Language (XML) 1.0 (Third Edition).* `http://www.w3.org/TR/REC-xml`. – Online–Ressource, Abruf: 16. April 2006. – W3C Recommendation

Wollnik 1990

Wollnik, Michael: Sozio-technisches System. In: Seelos, Hans-Jürgen (Hrsg.): *Wörterbuch der Medizinischen Informatik.* Berlin : Walter de Gruyter Verl., 1990, S. 462 f. – ISBN 3–11–011224–8

Ziegenbein 2001

Ziegenbein, Ralf: *Klinisches Prozessmanagement : Implikationen, Konzepte und Instrumente einer ablauforientierten Krankenhausführung.* Gütersloh : Verl. Bertelsmann Stiftung, 2001 (Leistungsorientierte Führung und Organisation im Gesundheitswesen 3). – ISBN 3–89204–594–1. – Zugl.: Münster/Westf. Univ., Diss., 2001

Thesen zur Diplomarbeit

1. Mit der Unternehmens- und Geschäftsprozess-Modellierungssoftware ARIS Toolset erstellte Modelle sind in das Format von MLDesigner-Modellen konvertierbar.

2. Klinische Prozesse werden mit dem ARIS Toolset im Wesentlichen durch ereignisgesteuerte Prozessketten (EPK) modelliert.

3. Ereignisgesteuerte Prozessketten sind auf MLDesigner-Modelle abbildbar.

4. Die EPK-Objekte Ereignis und Funktion lassen sich als MLDesigner-Module, das EPK-Objekt Regel als Primitive umsetzen.

5. Für das hier zu realisierende Konvertierungswerkzeug wurden alle notwendigen Anforderungen ermittelt und beschrieben; die Umsetzung dieser Anforderungen in Spezifikationen und die daraus abgeleitete Implementierung wurden korrekt vollzogen.

6. Für die Umwandlung von exportierten ARIS-Modellen in importierbare MLDesigner-Librarys eignet sich die Datentransformation mit XSLT.

7. Für das Benutzerinterface und die Dateizugriffe beim Konvertierungsvorgang eignet sich die betriebssystemunabhängige Programmiersprache Java.

8. Der in dieser Arbeit entworfene und implementierte ARIS-MLDesigner-Konverter als Java-Applikation mit XSLT-Einbettung ist hinsichtlich seiner gewünschten Funktion vollständig, verifiziert und validiert.

9. Vom ARIS-MLDesigner-Konverter aus ARIS-Modellen erzeugte MLDesigner-Bibliotheken bilden die ursprünglichen Modelle in gültiger Form ab.

10. Die zukünftige Implementierung und Konvertierung von Prozessmodellen sollte unter Verwendung offener, allgemein zugänglicher Standardspeicherformate geschehen.

Eidesstattliche Erklärung

Die vorliegende Arbeit habe ich selbstständig und ohne Benutzung anderer als der angegebenen Quellen angefertigt. Alle Stellen, die wörtlich oder sinngemäß aus veröffentlichten und nicht veröffentlichten Quellen entnommen wurden, sind als solche kenntlich gemacht. Die Arbeit ist in gleicher oder ähnlicher Form oder auszugsweise im Rahmen einer anderen Prüfung noch nicht vorgelegt worden.

Ilmenau, 17.08.2006 _____