Eine Komponentenarchitektur zur Integration heterogener Modellierungswerkzeuge

zur Erlangung des akademischen Grades eines

Doktors der Ingenieurwissenschaften

der Fakultät für Informatik
der Universität Fridericiana zu Karlsruhe (TH)

genehmigte

Dissertation

von

Elisabeth Syrjakow

aus Karlsruhe

Tag der mündlichen Prüfung:	28. Juni 2005
Erster Gutachter:	Prof. em. Dr. Detlef Schmid
Zweiter Gutachter:	Prof. Dr. Wolffried Stucky

Elisabeth Syrjakow

Eine Komponentenarchitektur zur Integration heterogener Modellierungswerkzeuge

Kurzfassung

Zur Entwicklung technischer Systeme sind heute auf Grund der in den letzten Jahren stark angestiegenen Komplexität und Heterogenität rechnergestützte Entwurfsverfahren unabdingbar geworden. Diese bieten die Möglichkeit, den Großteil der Entwicklung in frühe Entwurfsphasen zu verlagern und damit kostspielige und zeitintensive Änderungen an realen Prototypen in späteren Entwicklungsschritten zu vermeiden. Die hohen Anforderungen an die Qualität und Effizienz der zu entwickelnden Systeme bedingen den Einsatz von Methoden aus den Bereichen Modellierung, Simulation und Optimierung. Dazu existiert heutzutage eine Vielfalt an unterschiedlichen Modellierungskonzepten, Beschreibungsmitteln und Analysemöglichkeiten, die als Resultat eine heterogene Werkzeuglandschaft hervorgebracht haben. Dies lässt sich am Beispiel von Petrinetz-basierten Modellierungswerkzeugen, die sich hinsichtlich ihres Funktionsumfangs stark voneinander unterscheiden können, besonders gut verdeutlichen. Aufgrund der Komplexität der Aufgabenstellung beim Entwurf technischer Systeme wäre es wünschenswert, unterschiedliche Werkzeuge für die verschiedenen Abschnitte des Entwurfsprozesses gemeinsam nutzen zu können. Von Nachteil bei den heute etablierten Modellierungswerkzeugen ist allerdings ihr monolithischer Werkzeugaufbau sowie die Vielfalt unterschiedlicher Modellbeschreibungsformate. Dadurch wird die gleichzeitige Nutzung des Funktionsumfangs mehrerer Werkzeuge erschwert bzw. ganz verhindert.

Aus ökonomischen Gründen, aber auch zur Beherrschung und Pflege von entstandenen Modellen sowie zu Wartungs- und Vernetzungszwecken wird in dieser Arbeit eine Konvergenz von Methoden, Beschreibungsmitteln und bereits existierenden Werkzeugen angestrebt, die einen innovativen Integrationsansatz erfordert. Diesem Ansatz liegt ein komponentenbasiertes Architekturkonzept zu Grunde, das es ermöglicht, durch Bereitstellung geeigneter Schnittstellen, individuell benötigte Komponenten zur Bearbeitung einer Modellierungsaufgabe gemeinsam verwenden zu können. Dabei kann es sich sowohl um bewährte Komponenten aus bereits existierenden Altwerkzeugen als auch um Neuentwicklungen handeln. Zur Unterstützung des Zugangs zu Daten sowie des Umgangs mit ihnen in einer offenen und verteilten Umgebung wurden generische Schnittstellen entwickelt, denen aktuelle internationale Standards zu Grunde liegen. Die Vorteile dieser Vorgehensweise liegen darin, dass durch den einfachen Austausch einzelner Komponenten veränderte Anforderungen schneller umgesetzt und neue Merkmale oder verbesserte Funktionen schneller eingesetzt werden können, da der Austausch nur lokale Änderungen erfordert. Dies führt zu einer Beschleunigung der Entwicklung und zu einer einfacheren Wartung der Einzelfunktionalität. Die Validation der entwickelten Konzepte erfolgte im Rahmen des Forschungsprojekts Informationslogistik anhand eines komplexen Beispielszenarios aus dem Bereich der Ingenieurwissenschaften.

Inhaltsverzeichnis

Abkürzungen

AIML	Astronomical Instrument Markup Language
API	Application Programming Interface
CDSPN	Concurrent DSPN
CERN	Centre Européen de Recherche Nucléaires
CGI	Common Gateway Interface
CML	Chemical Markup Language
CORBA	Common Object Request Broker Architecture
DARPA	Defense Advanced Research Projects Agency
DDN	Defense Data Network
DIN	Deutsche Industrie-Norm
DoD	Department of Defense
DOM	Document Object Model
DSPN	Deterministic and SPN
DTD	Document Type Definition
EDSPN	Extended DSPN
FOM	Federation Object Model
GA	Genetischer Algorithmus
GUI	Graphical User Interface
GSPN	Generalized SPN
HC	Hill-Climbing-Verfahren
HLA	High Level Architecture
HTML	Hyper Text Markup Language
HTTP	Hyper Text Transfer Protocol
IDL	Interface Definition Language
ISO	International Standardization Organization
MathML	Mathematical Markup Language
NSF	National Science Foundation
OMG	Object Management Group
OMT	Object Model Template
ORB	Object Request Broker
RTI	Runtime Infrastructure
SAX	Simple API for XML
SOAP	Simple Object Access Protocol
SOM	Simulation Object Model
SPN	Stochastisches Petrinetz
SVG	Scalable Vector Graphics

TCP/IP	Transmission Control Protocol/Internet Protocol
UDDI	Universal Description, Discovery, and Integration of Web Services
UML	Unified Modeling Language
URL	Uniform Resource Locator
VRML	Virtual Reality Modeling Language
W3C	World Wide Web Consortium
WAP	Wireless Application Protocol
WML	Wireless Markup Language
WSDL	Web Service Description Language
WWW	World Wide Web
XML	eXtensible Markup Language
XSL	eXtensible Stylesheet Language
XSLT	XSL-Transformations

1 Einleitung

1.1 Motivation

Die Entwicklung technischer Systeme wird heutzutage in zunehmendem Maße durch das Zusammenwachsen ehemals getrennter Fachgebiete wie Mechanik, Elektrotechnik und Informationsverarbeitung geprägt. Durch die Integration dieser Disziplinen entstehen neue, innovative Produkte, die als *mechatronische* bzw. als *eingebettete Systeme* bezeichnet werden. Ihre Leistungsfähigkeit beziehen sie zum einen durch die verschiedenen Disziplinen, zum anderen durch die Verwendung der jeweils für eine Teilaufgabe idealen Komponente. Auf Grund der gestiegenen *Komplexität* und *Heterogenität* kann die Entwicklung und Optimierung größerer technischer Systeme nur noch unter Verwendung rechnergestützter Entwurfsverfahren erfolgen. Diese bieten die Möglichkeit, den Großteil der Entwicklung in frühe Entwurfsphasen zu verlagern und damit kostspielige und zeitintensive Änderungen des realen Prototypen in späteren Phasen zu vermeiden /Sche-01/. Generell bedingen die sehr hohen Anforderungen an die Qualität und Effizienz der zu entwickelnden Systeme den Einsatz von Methoden aus der Modellierung, Simulation und Optimierung.

Simulieren heißt Probieren. Bei real existierenden Systemen sind Manipulationen oftmals nur sehr eingeschränkt möglich, bei nicht existenten, sich noch im Planungsstadium befindenden Systemen sogar ganz unmöglich. Hier bietet die Simulation einen Ausweg, indem zunächst ein realistisches *Modell* des zu untersuchenden Systems oder Prozesses erstellt wird, mit dem systematisch experimentiert werden kann /SzUt-00/. Die anhand des Modells gewonnenen Erkenntnisse können anschließend in die Realität übertragen werden, um damit beispielsweise Planungs- und Entwicklungsfehler zu vermeiden. Bild 1.1.1 zeigt eine schematische Darstellung der Modellbildung und Simulation wie sie in /Page-91/ zu finden ist.

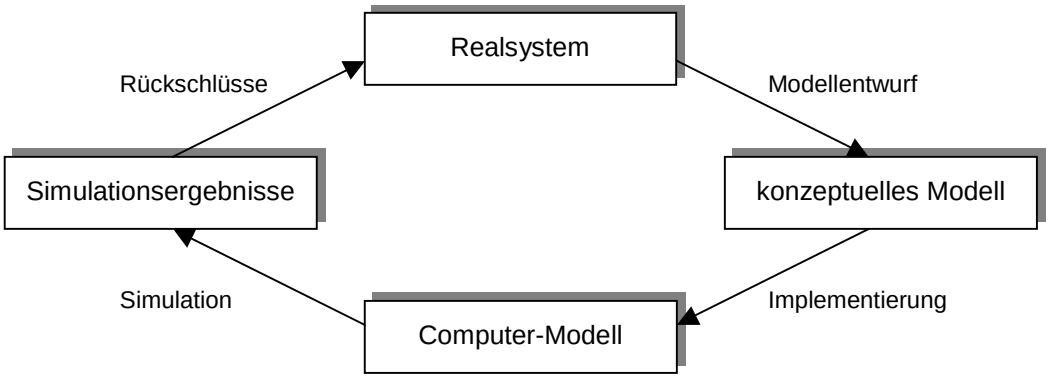


Bild 1.1.1 Schematische Darstellung der Modellbildung und Simulation

In Kombination mit geeigneten Optimierungsmethoden lassen sich die modellierten und analysierten Systeme und Prozesse zudem optimal auslegen, beispielsweise hinsichtlich der Ressourcenauslastung, des Durchsatzes oder der Zuverlässigkeit. Insbesondere bei der Konzeption und Realisierung komplexer Anlagen und Produktionsprozesse sollte die Simulation ein fester Bestandteil des Entwicklungsprozesses sein und so früh wie möglich eingesetzt werden. Mit ihrer Hilfe können alle Entwicklungsstadien systematisch unterstützt, Alternativen bewertet und dadurch von vornherein Fehler vermieden werden. Damit lässt sich der Entwicklungsprozess in der Regel erheblich beschleunigen und verkürzen, wobei die Simulationstechnik den Entwickler nicht ersetzt, sondern Entwicklungssicherheit gibt.

Petrinetze werden in der Literatur /Ober-96/ als durchgängig, das heißt als von der Planung bis hin zur Ausführung einsetzbare Beschreibungssprache für betriebliche Abläufe bzw. für das Systemverhalten vorgeschlagen. Der Einsatz von Petrinetzen zur Modellierung von Planungs- und Entwicklungsprozessen bietet folgende Vorteile:

- Formale Darstellung
- Frühzeitige Erkennung von Verklemmungen und Inkonsistenzen durch *Strukturanalyse*
- Verhaltensanalyse und –prognose durch *Simulation*
- Steigerung der Effektivität und Effizienz sowie Verbesserung der Ressourcenausnutzung durch *Optimierung*

Für eine praktische Nutzung von Petrinetzen ist die Verfügbarkeit entsprechender Werkzeuge unabdingbare Voraussetzung. Heutzutage existiert bereits eine Vielzahl an Petrinetz-Werkzeugen, von denen jedes bestimmte Vorzüge besitzt. Aufgrund der Komplexität der Aufgabenstellung bei der Entwicklung technischer Systeme wäre es wünschenswert, verschiedene Werkzeuge, basierend auf unterschiedlichen Analyse-, Simulations- und Optimierungsmethoden, für die verschiedenen Abschnitte des Entwicklungsprozesses nutzen zu können. Des Weiteren wäre aus Anwendersicht eine Integration von sowohl Methoden als auch von Werkzeugen in bereits existierende Systementwicklungsumgebungen von Vorteil. Aus all diesen Betrachtungen wird ersichtlich, dass neben leistungsfähigen Modellierungswerkzeugen auch geeignete Integrationskonzepte zur Unterstützung von Planungs- und Entwicklungsprozessen notwendig sind.

1.2 Problemstellung

Die Erstellung einer leistungsfähigen Modellierungs- und Simulationsumgebung erfordert heutzutage einen immensen Programmieraufwand, da die Vielzahl an Anforderungen zu immer größeren, unüberschaubaren und vor allem immer schlechter wartbaren Programmen führt. Am Beispiel moderner Textverarbeitungssysteme wird deutlich, welche Ausmaße dieser Trend hervorbringen kann. Diese Programme versuchen, jede nur erdenkliche Funktion zur Verfügung zu stellen, und überfordern dabei oft den Anwender, der nur einen Bruchteil dieser Funktionalität nutzen will. Statt dieser monolithischen Programmform ist ein offenes Konzept anzustreben, das dem Anwender die Möglichkeit gibt, sich genau die Funktionalität abzurufen, die er für die zu bearbeitende Aufgabe tatsächlich benötigt. Bei so genannten offenen Umgebungen ist es möglich, Softwarewerkzeuge unter Nutzung eines definierten Proto-

kolls je nach Erfordernis hinzuzufügen bzw. zu entfernen und Werkzeuge innerhalb der Umgebung für jedes spezifische Anwendungsgebiet zu modifizieren /StCe-94/. Neben Offenheit wird heute von Modellierungs- und Simulationsumgebungen vor allem die Nutzbarkeit der zur Verfügung gestellten Funktionalität über das Internet gefordert. Beispiele für solche innovativen Web-basierten Modellierungs- und Simulationsumgebungen finden sich in /KaZY-97/, /SeSB-98/, /Wied-99/ und /SoMa-02/.

Bei den heute etablierten Petrinetz-basierten Modellierungswerkzeugen handelt es sich meist um hochkomplexe monolithische Softwaresysteme, die in vielen Fällen einen langen evolutionären Entwicklungsprozess hinter sich haben. Beispielhaft seien hier die Systeme TimeNET /Zimm-03/, DSPNexpress /Lind-03/ und GreatSPN /GSPN-03/ aufgeführt. Diese Werkzeuge weisen hinsichtlich ihres Funktionsumfangs eine große Implementierungsvielfalt bezüglich graphischer Modell-Editoren, Tokenspiel-Animationen sowie Struktur- und Verhaltensanalysen auf. Das breite Spektrum vorhandener Werkzeuge bietet dem Modellierer zwar vielfältige Möglichkeiten zum Experimentieren mit den verschiedensten Petrinetzarten. Von Nachteil ist allerdings der monolithische Werkzeugaufbau sowie die Vielfalt unterschiedlicher Modellbeschreibungsformate. Dadurch wird die gleichzeitige Nutzung des Funktionsumfangs mehrerer Petrinetz-Werkzeuge erschwert bzw. ganz verhindert. Auch der Austausch oder die Kombination unterschiedlicher Werkzeugteile ist aufgrund uneinheitlicher Schnittstellen und unzureichender Dokumentation oft ein sehr schwieriges Unterfangen. Dies verzögert vor allem Weiterentwicklungen sowie die Integration innovativer Neuerungen. Des Weiteren ist unter diesen Voraussetzungen die heute zunehmend geforderte Nutzbarkeit der Werkzeugfunktionalität über das World Wide Web nur schwer zu erfüllen.

1.3 Lösungsansatz

Wesentliche Voraussetzung für kooperative, effiziente und möglichst fehlertolerante Realisierungen verteilter Anwendungen sind geeignete Softwarekomponenten. So werden in verteilten Umgebungen Anwendungen meist nicht komplett neu geschrieben, sondern in der Regel aus bereits vorhandenen Bausteinen - so genannten *Komponenten* - in geeigneter Weise zusammengesetzt. In vielen Ingenieursbereichen erfolgt heute die Produktentwicklung auf der Basis von Komponenten. Komponenten mit definierten oder genormten Eigenschaften bilden die Grundbausteine bei der Konstruktion komplexer Systeme. Beispiele hierzu finden sich in der Automobilbranche, im Bereich elektronischer Systeme oder auch im Hausbau. Komponentenbasierte Architekturen helfen den Herstellungsvorgang zu beschleunigen und sind teilweise sogar unabdingbar, um auch die Konstruktion komplexer Produkte überhaupt sinnvoll handhaben zu können. Durch komponentenbasierte Entwicklungstechniken lässt sich grundsätzlich ein hoher Grad an Arbeitsteilung erreichen, da die Herstellung einzelner Komponenten im allgemeinen von der Herstellung des Gesamtprodukts, in das sie eingehen, getrennt werden kann. Ein weiterer Vorteil der komponentenbasierten Entwicklung ist die Möglichkeit, einzelne Komponenten - innerhalb gewisser Spezifikationsgrenzen - auch nachträglich noch auszutauschen, ohne dass die Stabilität oder Gebrauchsqualität des Gesamtprodukts davon negativ beeinflusst würde. Beispielsweise kann Verschleiß oder veränderte Anforderungen an ein Produkt den Austausch von Komponenten erforderlich machen.

Die Hauptziele dieser Arbeit liegen in der Konzeption und Entwicklung von Komponenten aus dem Bereich der Modellierung und Simulation, die Einzelfunktionalitäten darstellen und über geeignete Schnittstellen jeweils nach den aktuellen Anforderungen zusammengebunden werden können. Die Vorteile dieser Vorgehensweise liegen darin, dass durch den einfachen Austausch einzelner Komponenten veränderte Anforderungen schneller umgesetzt und neue Merkmale oder verbesserte Funktionen schneller eingesetzt werden können, da der Austausch nur lokale Änderungen erfordert. Dies führt zu einer Beschleunigung der Entwicklung und zu einer einfacheren Wartung der Einzelfunktionalitäten. Für die Implementierung der entwickelten Komponenten soll die moderne Programmiersprache Java eingesetzt werden, um eine plattformunabhängige Ausführung sowie Verfügbarkeit über das World Wide Web zu ermöglichen.

Neben der Zergliederung eines Systems in Komponenten spielt die Art und Weise ihres Zusammenspiels eine große Rolle. Dazu wird in dieser Arbeit zwischen einer losen und einer engen Kopplung von Komponenten unterschieden. Die lose Kopplung basiert auf Dokumenten, die zwischen den Komponenten ausgetauscht werden. Zur engen Komponentenkopplung werden (entfernte) Prozedur- bzw. Methodenaufrufe verwendet. Insgesamt geht es um die Unterstützung des Zugangs zu Daten und Funktionen in einer offenen verteilten Modellierungs- und Simulationsumgebung durch dafür geeignete möglichst *generische Schnittstellen*. Die Verwendung von gemeinsamen Schnittstellen statt jeweils einer eigenen Schnittstelle von bzw. zu allen Komponenten hat den Vorteil, dass die Anzahl der benötigten Schnittstellen reduziert wird. Den in dieser Arbeit entwickelten Schnittstellen liegen aktuelle internationale Standards zu Grunde.

1.4 Gliederung der Arbeit

Die vorliegende Dissertation umfasst insgesamt sechs Kapitel. Nachdem die Arbeit im ersten Kapitel motiviert wurde, werden in *Kapitel 2* die Grundlagen dieser Arbeit vorgestellt. Dazu wird zunächst der Stand der Technik auf dem Gebiet der Systemmodellierung dargelegt, indem neben der Einführung grundlegender Begriffe die Vorgehensweise bei der Modellierung sowie unterschiedliche Arten von Modellierungskonzepten erläutert werden. Anschließend wird auf das Modellierungskonzept der Petrinetze eingegangen, und es werden Möglichkeiten zur Modellanalyse und –optimierung aufgezeigt. Einen weiteren Schwerpunkt dieses Kapitels bildet der Forschungsbereich Web- und Komponententechnologien in der Modellierung und Simulation. Dazu werden die wichtigsten Begriffe sowie die für diese Arbeit relevanten Technologien erläutert. Das Kapitel endet mit einer Analyse klassischer Modellierungswerkzeuge, aus der die Ziele der vorliegenden Arbeit abgeleitet werden.

In *Kapitel 3* wird aufbauend auf den in Kapitel 2 diskutierten Problemen klassischer Modellierungswerkzeuge eine Architektur vorgeschlagen, der ein Komponenten-basierter Ansatz zu Grunde liegt. Zuvor werden die Entwurfsentscheidungen für die interne Organisation dieser Architektur erläutert. Einen wichtigen Aspekt der entwickelten Komponenten-basierten Architektur stellen die Schnittstellen zwischen den einzelnen Komponenten sowie die Schnittstellen zu externen Systemen dar. Hierzu werden verschiedene Realisierungsmöglichkeiten aufgezeigt und Lösungsansätze diskutiert. Im Anschluss daran wird ein Integrations-

konzept auf Basis standardisierter Dokumente vorgestellt. Abschließend wird eine mögliche Form der technischen Umsetzung für diese Architektur beschrieben.

In *Kapitel 4* werden verschiedene Konzepte zur automatischen Modelltransformation vorgestellt wie sie im Kontext einer Integration heterogener Komponenten aus dem Bereich der Modellierung und Simulation benötigt werden. Dabei handelt es sich um Modelltransformationen zwischen unterschiedlichen Modellformaten und Modellformalismen. Der Einsatz und die Bewertung der in dieser Arbeit entwickelten Konzepte erfolgt in *Kapitel 5* anhand eines konkreten Beispiels aus dem Bereich der Ingenieurwissenschaften. Die Arbeit schließt mit einer Zusammenfassung in *Kapitel 6*.

2 Grundlagen

In diesem Kapitel werden die für diese Arbeit relevanten Grundlagen dargelegt. Abschnitt 2.1 gibt zunächst eine Einführung in die Systemmodellierung, wobei grundlegende Begriffe und Vorgehensweisen sowie Modellierungskonzepte erläutert werden. Im Anschluss daran wird in Abschnitt 2.2 die Modellbildung, Analyse und Optimierung dynamischer Systeme mit Hilfe von Petrinetzen erläutert. Der darauf folgende Abschnitt 2.3 befasst sich mit dem Einsatz von Web- und Komponententechnologien in der Modellierung und Simulation (M&S). Anschließend wird in Abschnitt 2.4 der Aufbau heute weit verbreiteter, dem aktuellen Stand der Technik entsprechender Modellierungswerkzeuge erläutert. In Abschnitt 2.5 werden schließlich die Ziele der vorliegenden Arbeit formuliert.

2.1 Einführung in die Systemmodellierung

Dieser Abschnitt führt zunächst die grundlegenden Begriffe der Systemmodellierung ein. Anschließend wird die Vorgehensweise bei der Modellierung aufgezeigt sowie eine Klassifikation von Modellierungskonzepten hinsichtlich verschiedener Kriterien durchgeführt.

2.1.1 Grundlegende Begriffe

In /Schn-97/ wird ein System wie folgt definiert:

> *In einem betrachteten Zusammenhang ist ein System eine Gesamtheit von Objekten, die sich aufgrund der Beziehungen der Objekte untereinander von ihrer Umwelt abhebt, davon abgegrenzt erscheint und daher ein als Einheit anzusehendes und gegliedertes Ganzes bildet.*

Somit ist zur Festlegung eines Systems eine Abgrenzung zu seiner Umgebung notwendig, bzw. die Grenzen müssen genau spezifiziert sein. Jede Interaktion zwischen dem System und seiner Umgebung wird über Schnittstellen nach vereinbarten Regeln durchgeführt. Zur Untersuchung der Systemstruktur bzw. des dynamischen Systemverhaltens kann entweder das System selbst oder ein *Modell* verwendet werden. Ein Modell ist eine Abbildung des Systems mit gleichem oder ähnlichem Verhalten hinsichtlich der zu untersuchenden Eigenschaften. Da nur bestimmte Eigenschaften konkret modelliert werden, lässt sich ein Modell in der Regel einfacher handhaben als das reale System. Das bedeutet aber auch, dass bei einem späteren Einsatz eines Modells dessen Modellierungsintention beachtet werden muss /Sche-01/. Der Begriff Modell wird in /Schn-97/ folgendermaßen definiert:

> *Idealisierte, vereinfachte, in gewisser Hinsicht ähnliche Darstellung eines Gegenstandes, Systems oder sonstigen Weltausschnittes mit dem Ziel, daran bestimmte Eigenschaften des Vorbilds besser studieren zu können.*

Des Weiteren wird zwischen materiellen und immateriellen Modellen unterschieden. Die in dieser Arbeit nicht betrachteten materiellen Modelle sind physikalisch existent. Findet eine mathematische bzw. abstrakte Formulierung der dem jeweiligen System zugrunde liegenden Eigenschaften statt, spricht man von immateriellen Modellen.

Um Beobachtungen machen zu können und damit das Verhalten des Systems zu studieren, werden *Experimente* durchgeführt. Dabei können Parameter des Systems, die Systemstruktur oder Randbedingungen des Systems verändert werden. Ein Experiment kann sowohl mit Hilfe des realen Systems als auch mit Hilfe des abstrakten Modells durchgeführt werden. Wird anhand eines Modells experimentiert, so spricht man auch von *Simulation*. Erfolgt die Simulation mit Hilfe eines Rechners, wird der Begriff rechnergestützte Simulation verwendet. Die rechnergestützte Simulation bietet eine Reihe von Vorteilen gegenüber dem Experimentieren mit dem realen System, die im Folgenden näher erläutert werden:

- *Das System befindet sich in einem frühen Entwurfsstadium*: Simulationen werden häufig durchgeführt, um die Konsequenzen von Entwurfsentscheidungen besser abschätzen zu können.
- *Experimente am System sind zu gefährlich*: Vielfach werden mit der Simulation Grenzwerte bestimmt, die in der Realität nicht erreicht werden dürfen.
- *Die Kosten eines Experiments sind zu hoch*: Eine rechnergestützte Simulation ist in vielen Fällen wesentlich preiswerter als die Durchführung eines Experiments am realen System.
- *Die Beobachtung interner Zustände ist im realen Experiment in der Regel nicht oder nur sehr schwer möglich*: In der Simulation können alle Zustände, Variablen und Parameter des Systems untersucht werden. Wesentlich ist dabei auch, dass bei der Simulation das funktionale und zeitliche Verhalten durch die Beobachtung nicht beeinflusst wird.
- *Die Durchführung eines Experiments ist zu zeitaufwendig*: Zum Beispiel erfordert die Durchführung eines Populationsexperiments in der Tierwelt mehrere Monate oder Jahre. Mit Hilfe der Simulation kann diese Zeit auf einen vertretbaren Wert verringert werden. Auch eine Dehnung der Zeit, zum Beispiel bei der Untersuchung einer chemischen Reaktion, ist möglich.

Somit können eine Vielzahl von Experimenten nur am Modell durchgeführt werden. Dabei muss jedoch beachtet werden, dass ein Modell eine Vereinfachung des Systems darstellt und insbesondere bei einer hohen Abstraktionsebene große Unterschiede zwischen dem simulierten und dem realen Systemverhalten auftreten können. Die Ergebnisse einer Simulation sollten deshalb immer auf Plausibilität untersucht und nur mit Bezug auf das verwendete Modell betrachtet werden. Üblicherweise erhält man erst durch eine Reihe von unterschiedlichen Simulationen einen umfassenden Einblick in das Systemverhalten. Ein einzelner Simulationslauf zeigt nur das Verhalten bei einer bestimmten Parametereinstellung.

2.1.2 Vorgehensweise bei der Modellierung

Bild 2.1.1 zeigt schematisch die einzelnen Arbeitsschritte, die bei einer Modellstudie durchzuführen sind. Im Einzelnen handelt es sich dabei nach /Saue-99/ um:

- *Schritt 1: Formulieren des Problems*

  Am Anfang jeder Modellstudie sollte eine Formulierung der Problemstellung stattfinden, in der die zu untersuchenden Fragestellungen und zu erreichenden Ziele festgelegt werden. Dieser Arbeitsschritt ist ein wichtiger Teil, da hierbei eine Abgrenzung des zu untersuchenden Realitätsausschnitts und die Bestimmung des Detaillierungsgrades des Modells vorgenommen werden muss. Fehler, die in dieser frühen Phase gemacht werden, sind später nur mit großem Aufwand zu korrigieren.

- *Schritt 2a: Entwickeln des konzeptionellen Modells*

  Ausgehend von der Problemstellung wird das Modell zunächst konzeptionell entwickelt. Hierzu müssen die wesentlichen Eigenschaften des Systems und deren Wechselwirkungen erkannt und beschrieben werden. Des Weiteren ist für die Umsetzung in ein Simulationsmodell die Art der Simulation festzulegen. Dies beinhaltet die Frage, ob eine kontinuierliche oder diskrete Simulation gewählt werden soll, sowie die Festlegung der gewünschten zeitlichen Auflösung.

- *Schritt 2b: Erheben und Generieren von Daten*

  Zur Durchführung der Simulation werden Ströme von Eingangsdaten benötigt. Eine Möglichkeit der Datenerhebung besteht darin, diese empirisch vorzunehmen. Allerdings lassen sich damit nur Abläufe aus der Vergangenheit reproduzieren. Allgemeine Aussagen oder Prognosen für künftiges Verhalten sind nur bedingt zu gewinnen. Deshalb ist es sinnvoll, die gefundenen Größen zu untersuchen und ihre Zufallsverteilung zu ermitteln, damit die Datenströme durch einen Zufallsgenerator erzeugt werden können.

- *Schritt 3: Implementieren des Modells*

  Nach der Spezifikation des Modells wird dieses in eine rechnerlesbare, ausführbare Form überführt. Das Modell kann zum einen vollständig in einer herkömmlichen Programmiersprache realisiert werden. Eine Alternative dazu sind spezielle Simulationssysteme, bei denen die Eingabe in einer speziellen Modellierungssprache oder in grafischer Form erfolgen kann. Das Ergebnis der Modellimplementierung ist ein Computerprogramm bzw. eine Modellbeschreibung, die auf einem Rechner ablauffähig ist.

- *Schritt 4: Verifizieren und Validieren des Modells*

  In einem nächsten Arbeitsschritt erfolgt die Verifikation und Validation des Modells. Bei der Verifikation wird überprüft, ob die einzelnen Schritte vom konzeptionellen Modell zum Simulationsmodell korrekt durchgeführt wurden. Dies wird als „building the model right“ bezeichnet. Demgegenüber behandelt die Validierung die Frage, ob das Modell die Realität im Hinblick auf die Zielsetzung der Simulation in geeigneter Weise abbildet. Entsprechend wird dies mit „building the right model“ umschrieben.

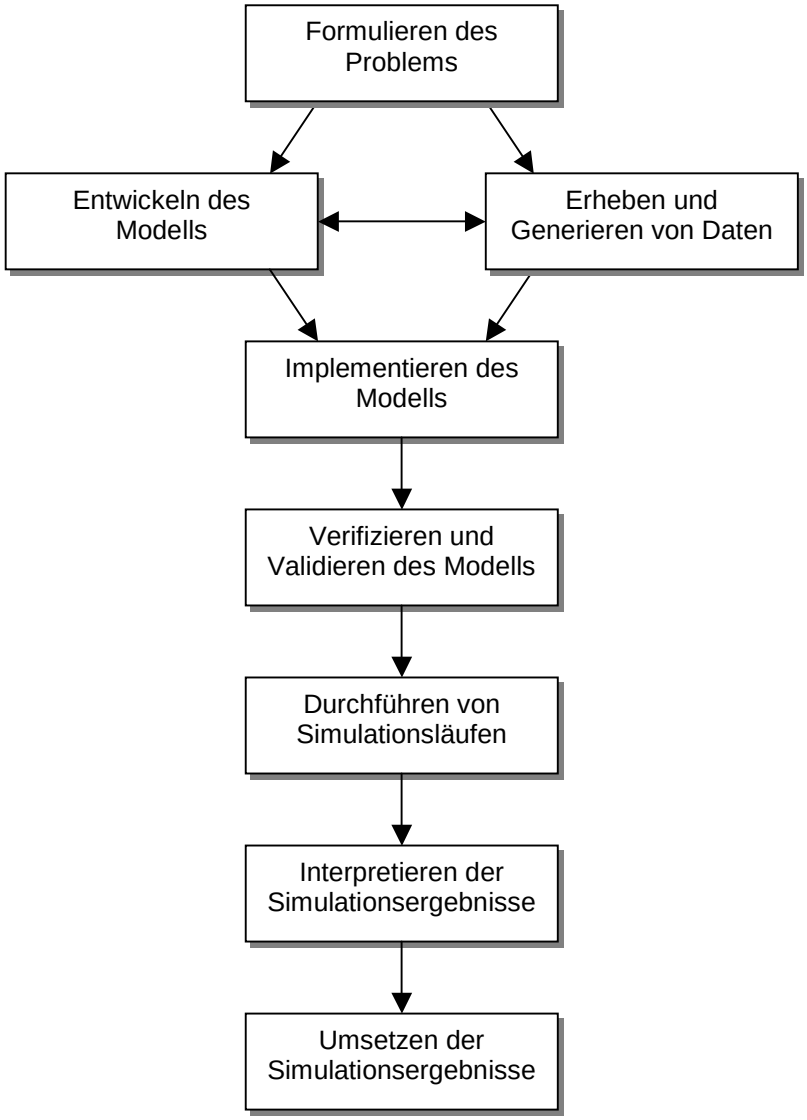


Bild 2.1.1 Ablaufmodell einer Modellstudie

- *Schritt 5: Durchführen von Simulationsläufen*

  Nachdem das Modell erstellt und einer Qualitätsprüfung unterzogen wurde, erfolgt das Durchführen und Auswerten von Simulationsläufen. In diesem Schritt ist zu berücksichtigen, ob ein stationärer oder nichtstationärer Prozess vorliegt. Nichtstationäre Prozesse können beispielsweise durch Einschwingvorgänge bedingt sein. Diese sollten bei stationären Untersuchungen unberücksichtigt bleiben. Für den Fall einer stochastischen Simulation ergibt sich bei jedem Simulationslauf in der Regel ein anderer Mittelwert, der um den theoretischen Erwartungswert streut. Um vertrauenswürdige Aussagen zur Genauigkeit bzw. Zuverlässigkeit von Simulationsergebnissen machen zu können, müssen daher möglichst viele unabhängige Simulationsläufe durchgeführt werden, so dass sich neben Mittelwerten auch Konfidenzintervalle schätzen bzw. berechnen lassen. Um das Modellverhalten auch bei etwas abweichenden Bedingungen abschätzen zu können, führt man zusätzlich Sensitivitätsanalysen durch. Dabei wird untersucht, wie sich Änderungen bestimmter Modelleingabeparameter an den Modellausgängen auswirken.

- *Schritt 6: Interpretieren der Simulationsergebnisse*

  Zum Abschluss des Simulationsexperiments findet eine Interpretation der Simulationsergebnisse statt, da die Ergebnisse zunächst einmal für das simulierte Modell gelten, nicht aber notwendigerweise auch für die Realität. Wurde eine Stichprobe simuliert, sind die Ergebnisse mit geeigneten Methoden auf die Grundgesamtheit hochzurechnen. Des Weiteren ist abzuschätzen, welchen Einfluss nicht modellierte Teile des realen Systems besitzen.

- *Schritt 7: Umsetzen der Simulationsergebnisse*

  Der letzte Arbeitsschritt einer Modellstudie befasst sich mit der Umsetzung der Simulationsergebnisse in die Realität. Simulationen, die innerhalb wissenschaftlicher Studien angewandt werden, dienen meist dem Erkenntnisgewinn. Im Gegensatz dazu stellen in der betrieblichen Praxis durchgeführte Simulationen in der Regel eine Entscheidungshilfe dar.

2.1.3 Klassifikation von Modellierungskonzepten

Modelle können nach verschiedenen Kriterien klassifiziert werden. In /Page-91/ findet eine Unterscheidung nach folgenden Kriterien statt:

- Art der Untersuchungsmethode
- Abbildungsmedium
- Art der Zustandsübergänge
- Verwendungszweck

Werden Untersuchungen an einem Modell vorgenommen, so können Erkenntnisse entweder durch analytische Berechnung oder durch Nachspielen der Abläufe im Modell gewonnen werden[1]. Bei analytischen Modellen wird mit Hilfe eines Gleichungssystems der zu ermittelnde Systemzustand direkt bestimmt. Im Gegensatz dazu wird bei Simulationsmodellen der Modellzustand Schritt für Schritt fortgeschrieben, weshalb sich solche Modelle besonders zur Veranschaulichung des Systemverhaltens eignen.

In dieser Arbeit werden ausschließlich *immaterielle* Modelle betrachtet. Diese müssen nicht notwendigerweise *formal* als *mathematische* oder *grafisch-mathematische* Modelle dargestellt werden. Auch *informale* Beschreibungen wie *verbale* und *graphisch-deskriptive* Modelle können wichtige Zwischenstufen bei der Entwicklung formaler Modelle sein. Bild 2.1.2 veranschaulicht die Klassifikation nach Abbildungsmedium und Untersuchungsmethode.

[1] Modelle, denen eine bestimmte mathematische Berechnungsmethode zugeordnet werden kann, werden im Folgenden als *analytische* Modelle bezeichnet. Modelle, bei denen dies nicht zutrifft werden *Simulationsmodelle* genannt.

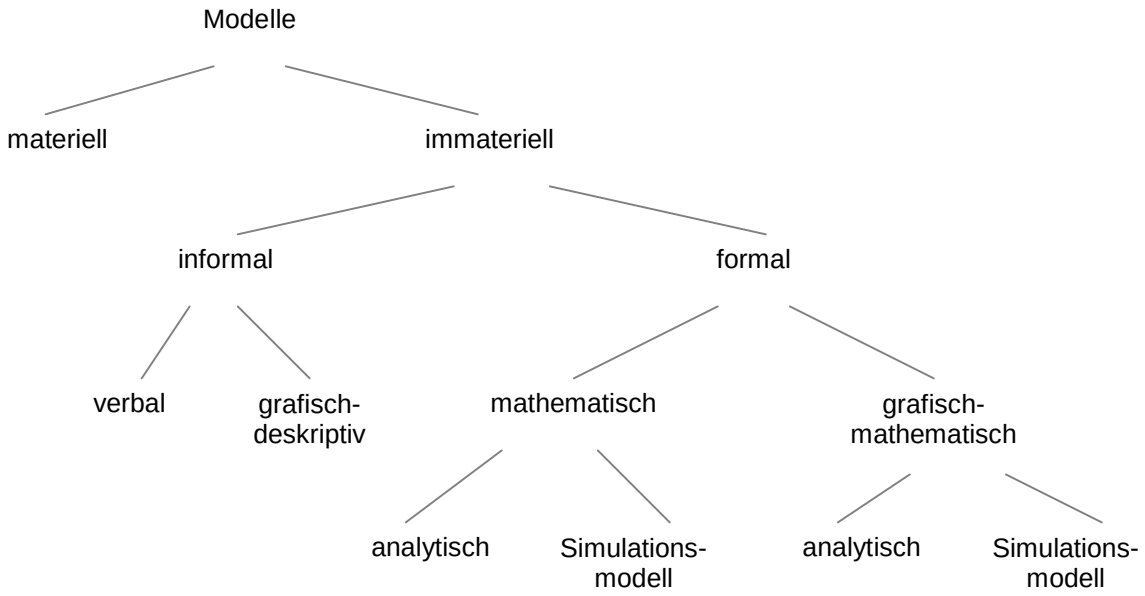


Bild 2.1.2 Klassifikation nach Abbildungsmedium und Untersuchungsmethode

Bild 2.1.3 zeigt eine Klassifikation von Modellen nach der Art der Zustandsübergänge. Treten keine Zustandsänderungen auf, wird von einem *statischen* Modell gesprochen. Ein *dynamisches* Modell zeichnet sich dagegen durch die Zeitabhängigkeit des Modellzustandes aus. Lassen sich die Änderungen der Zustandsvariablen über der Zeit durch stetige Funktionen beschreiben, handelt es sich dabei um ein *zeit- und zustandskontinuierliches* Modell. Ändern sich die Werte der Zustandsvariablen sprunghaft zu bestimmten, auf der Zeitachse diskret verteilten Zeitpunkten, spricht man von einem *zeitdiskreten* Modell. Diskrete Modelle können weiter unterteilt werden in *zeitgesteuerte* und *ereignisgesteuerte* Modelle. Zustandsänderungen werden durch das Auftreten von Ereignissen verursacht. Jedes Ereignis beinhaltet einen Aktivierungszeitpunkt, der häufig auch als Zeitstempel t_e bezeichnet wird, zu dem das Ereignis verarbeitet wird. Bei zeitgesteuerten Modellen wird die Zeit t in festgelegten konstanten Zeitschritten (Δt=const) erhöht. Nach jedem Zeitschritt werden alle Ereignisse mit dem Zeitstempel $t_e = t$ in einer willkürlichen Reihenfolge abgearbeitet. Falls ein Ereignis im Inneren des Zeitintervalls $<t_{i-1}, t_i>$ stattfindet, wird es auf den Zeitpunkt t_i gesetzt. Bei ereignisgesteuerten Modellen werden Ereignisse nach ihren Zeitstempeln, die beliebig auf der Zeitachse verteilt sein können, in eine zentrale Ereignisliste eingeordnet. Die Zeit t wird auf den Zeitstempel t_e des zeitlich nächsten Ereignisses gesetzt und die Aktion dieses Ereignisses wird abgearbeitet. In einer Aktion können beispielsweise Berechnungen stattfinden oder neue Ereignisse erzeugt werden. Nach der Abarbeitung wird das Ereignis aus der Liste gestrichen. Da bei ereignisgesteuerten Modellen jedes Ereignis zum exakten Zeitpunkt seines Auftretens abgearbeitet wird, sind diese besonders geeignet, wenn sehr unterschiedliche Ereignisdichten (Anzahl von Ereignissen pro Zeiteinheit) im Zeitverlauf auftreten. Modelle lassen sich weiter unterscheiden in *deterministische* und *stochastische* Modelle. Ein Modell heißt deterministisch, wenn seine Reaktion auf eine bestimmte Eingabe, ausgehend von einem bestimmten Zustand, eindeutig festgelegt ist. Bei einem stochastischen Modell dagegen lassen sich die Reaktionen nur durch Wahrscheinlichkeitsverteilungen beschreiben.

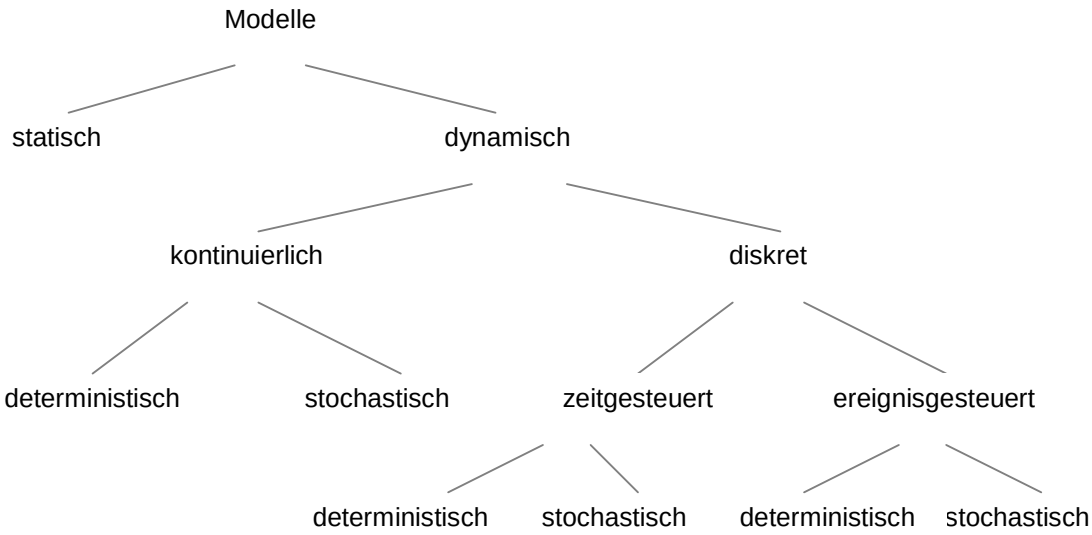


Bild 2.1.3 Klassifikation nach Art der Zustandsübergänge

Die in Bild 2.1.4 dargestellte Möglichkeit zur Klassifikation von Modellen basiert auf dem Verwendungszweck. Je nach Fragestellung und Zielsetzung werden unterschiedliche Anforderungen an Modelle gestellt. Ein *Erklärungsmodell* wird zur Erklärung des beobachteten Systemverhaltens herangezogen. Ein *Prognosemodell* dient dazu, das Systemverhalten unter gegebenen Annahmen abzuschätzen. *Gestaltungsmodelle* werden als Entscheidungshilfe beim Entwurf von Systemen und der Maßnahmenauswahl in der Projektplanung verwendet. *Optimierungsmodelle* schließlich dienen zur Ermittlung optimalen Systemverhaltens.

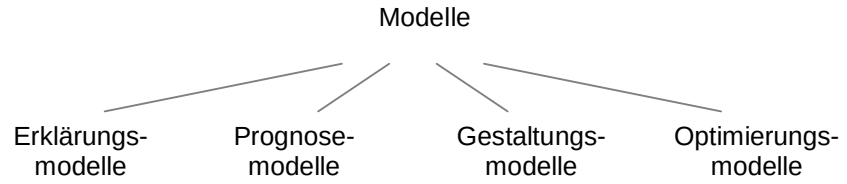


Bild 2.1.4 Klassifikation nach Verwendungszweck

Die Komplexität eines dynamischen Systems steigt, wenn zeitlich parallele, das heißt nebenläufige Prozesse auftreten. Werden durch solche Prozesse gemeinsame Betriebsmittel genutzt, so besteht grundsätzlich die Gefahr von Verklemmungssituationen, die im Allgemeinen nicht einfach vorherzusehen sind. Der folgende Abschnitt handelt von Petrinetzen, einem formalen Modellierungskonzept, mit dem nebenläufige Abläufe in einem System so dargestellt werden können, dass sie analysiert und im Anschluss daran optimiert werden können.

2.2 Modellbildung, Analyse und Optimierung dynamischer Systeme mit Petrinetzen

In diesem Abschnitt werden Möglichkeiten aufgezeigt, dynamische Systeme mit Hilfe von Petrinetzen zu beschreiben, zu analysieren und zu optimieren. Dazu werden im Folgenden zunächst die vorteilhaften Eigenschaften von Petrinetzen herausgestellt, die zur weiten Verbreitung dieser Modellierungsmethode beigetragen haben. Im Anschluss daran wird näher auf die Merkmale einer speziellen Petrinetz-Variante, die so genannten stochastischen Petrinetze, eingegangen. Der Abschnitt endet mit einer Beschreibung unterschiedlicher Methoden zur Modellanalyse und -optimierung.

2.2.1 Eigenschaften von Petrinetzen

Petrinetze gehören heute zu den klassischen Modellierungstechniken für nebenläufige und verteilte Systeme. Sie ermöglichen es, verteilte Systeme anschaulich und mathematisch fundiert zu spezifizieren. Dies ist hauptsächlich auf folgende Eigenschaften von Petrinetzen zurückzuführen /PGEE-00/:

- *Einfache Beschreibungssprache*: Petrinetze bestehen aus Stellen und Transitionen, einer Anfangsmarkierung und einfachen Schaltregeln und stellen damit einen einfachen und leicht verständlichen Beschreibungsformalismus dar.
- *Anschaulichkeit:* Die graphische Repräsentation des spezifizierten Systems als Petrinetz erlaubt es, die statischen und dynamischen Aspekte des Systems in natürlicher und damit leicht verständlicher Weise darzustellen.
- *Universalität:* Petrinetze werden zur Modellierung von Systemen aus den verschiedensten Bereichen eingesetzt, von ingenieurwissenschaftlichen Anwendungen bis hin zur Entwicklung von Softwaresystemen.
- *Wohlfundierte Theorie:* Die mathematische Fundierung der Petrinetztheorie erlaubt eine formale Verifikation von Netzeigenschaften (z.B. durch Invarianten- oder Erreichbarkeitsanalyse).
- *Simulationsfähigkeit:* Die Möglichkeit der Simulation des Prozessablaufs in einer Petrinetzspezifikation erlaubt die Validierung bezüglich der Anforderungen an das System. Die formale Analyse von Petrinetzen erlaubt darüber hinaus die Verifikation von Systemeigenschaften.
- *Existenz von Werkzeugen:* Es gibt eine reichhaltige Auswahl von Werkzeugen für die verschiedensten Petrinetz-Klassen, welche die Spezifikation, Simulation und Analyse von Petrinetzen unterstützen.

Petrinetze sind gekennzeichnet durch eine strenge Trennung von passiven und aktiven Systemkomponenten, dargestellt durch passive und aktive Netzelemente, die Stellen bzw. Transitionen genannt werden. Die passiven Stellen repräsentieren Zustände des Systems, die während des Ablaufs eines Systems angenommen werden können. Transitionen stellen Aktionen oder Ereignisse dar, für deren Ausführung bzw. Eintreten bestimmte Zustände Voraussetzung sind und nach deren Ausführung bzw. Auftreten bestimmte Zustände erreicht werden. In Be-

zug auf Transitionen kann man die Zustände also auch als Bedingungen auffassen. Die Veränderung der Zustände des Systems durch eine Transition wird ihre Wirkung genannt und durch die so genannte Flussrelation kodiert.

Petrinetze wurden von Carl Adam Petri in seiner Dissertation /Petr-62/ eingeführt und werden seitdem in unzähligen Artikeln und Büchern aus theoretischen und praktischen Gesichtspunkten heraus untersucht. In den ersten zwanzig Jahren wurden hauptsächlich Low-Level-Netze beschrieben, wie beispielsweise Bedingungs-Ereignis-Netze, Elementare Netze und Stellen-Transitions-Netze /Reis-85/. High-Level Petrinetze beinhalten Erweiterungen, die es ermöglichen, Zeit, Daten und Hierarchien zu modellieren. Die ersten High-Level-Netze, so genannte Prädikat-Transitions-Netze, wurden von Genrich und Lautenbach /GeLa-81/ entwickelt. In den letzten Jahren sind viele weitere Klassen von High-Level-Netzen eingeführt und untersucht worden, unter anderem Gefärbte Petrinetze von Jensen /Jens-95/, Petrinetze mit individuellen Marken und Kombination von Netzen mit algebraischen Spezifikationen. Eine weitere Petrinetz-Variante stellen stochastische Petrinetze /Baus-96/ dar, auf die im folgenden Abschnitt detaillierter eingegangen wird. Diese beinhalten die Möglichkeit, zeitliche Abläufe sowie zufällige Ereignisse innerhalb eines Systems zu modellieren.

2.2.2 Stochastische Petrinetze

Auf stochastische Petrinetze wird häufig zurückgegriffen, um Leistungsabschätzungen von komplexen technischen Systemen vorzunehmen. Dieser Abschnitt gibt zunächst eine formale Definition von stochastischen Petrinetzen. Im Anschluss daran wird aufgezeigt, wie stochastische Petrinetze quantitativ analysiert werden können, und zum Schluss werden einige Varianten von stochastischen Petrinetzen vorgestellt.

2.2.2.1 Definition

Stochastische Petrinetze (SPN) stellen eine Erweiterung von Standard-Petrinetzen dar. Ein Standard-Petrinetz, auch Stellen-Transitions-Netz oder kurz (S/T)-Netz genannt, ist ein gerichteter, bipartiter Graph. Dieser lässt sich nach /MaBC-86/ durch das 4-Tupel PN = (P,T,A,M_0) beschreiben, wobei

- $P=\{p_1,p_2,...,p_n\}$ eine endliche Menge von Plätzen (Stellen),
- $T=\{t_1,t_2,...,t_m\}$ eine endliche Menge von Transitionen,
- $A=\{a_1,a_2,...,a_k\}$ eine endliche Menge von Kanten,
- $M_0=\{m_{01},m_{02},...,m_{0n}\}$ eine Anfangsmarkierung

darstellt. Ein SPN beinhaltet zusätzlich die Möglichkeit, zeitliche Aspekte zu berücksichtigen. Formal lässt sich ein SPN durch das 5-Tupel SPN = (P,T,A,M_0,L) definieren, wobei P, T, A und M_0 die gleiche Bedeutung haben wie bei Standard-Petrinetzen und

- $L=\{l_1, l_2, \ldots,l_m\}$

die Menge von Schaltraten darstellt, die mit den Petrinetz-Transitionen assoziiert werden. Diese Schaltraten können markierungsabhängig sein.

In der graphischen Darstellung (siehe Bild 2.2.1) werden Plätze als Kreise, Transitionen als dünne Balken und Kanten als Pfeile dargestellt. Eine hemmende Kante endet mit einem schmalen Kreis, welcher mit einer Transition verbunden ist. Plätze können nicht unterscheidbare Marken enthalten, die als ausgefüllte Punkte dargestellt werden. Der Vektor, der die Anzahl der Marken in jedem Platz repräsentiert, entspricht dem Zustand des SPN und wird als Markierung bezeichnet. Mit jeder Kante kann eine markierungsabhängige Multiplizität assoziiert werden. Plätze, die mit einer Transition durch eine Kante verbunden sind, heißen je nach Pfeiltyp Eingangs-, Ausgangs- und hemmende Plätze der Transition. Eine Transition heißt *aktiviert* in einer Markierung, falls jeder Eingangsplatz mindestens so viele Marken enthält wie die Multiplizität der Eingangskante angibt und wenn jeder hemmende Platz weniger Marken enthält als die Multiplizität der hemmenden Kante. Eine Transition *feuert*, indem Marken aus Eingangsplätzen entfernt und je nach Multiplizität der entsprechenden Kanten zu Ausgangsplätzen hinzuaddiert werden.

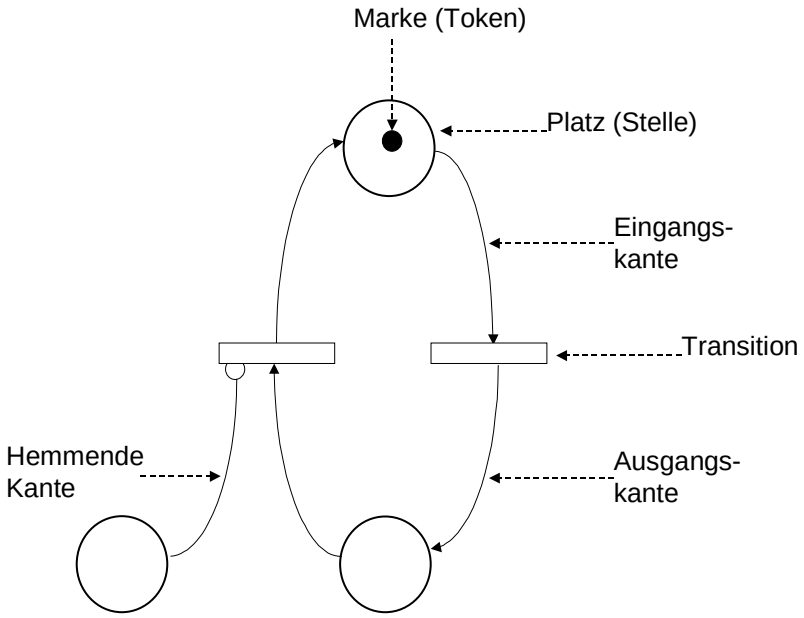


Bild 2.2.1 Graphiksymbole eines SPN

2.2.2.2 Berechnung von Leistungskenngrößen

Bei einem SPN wird jeder Transition eine exponentiell verteilte Schaltrate $l_i, i \in \{1,...,m\}$ zugeordnet, welche die Verzögerung von der Aktivierung bis zum Feuern einer Transition angibt. Sind in einem SPN mit einer Markierung M mehrere Transitionen $t_i, i \in H$ (Menge der aktivierten Transitionen) gleichzeitig aktiviert, dann feuert Transition t_i mit der Wahrscheinlichkeit $l_i \Big/ \sum_{k \in H} l_k$.

Daraufhin erreicht das SPN eine neue Markierung M', in der Transitionen noch aktiviert sein können, die schon in der vorhergehenden Markierung aktiviert waren, aber nicht feuern konnten.

Aufgrund der Eigenschaft der *Gedächtnislosigkeit* der Exponentialverteilung ist der nächste Zustand nur vom aktuellen „gegenwärtigen“ Zustand abhängig und nicht von der Vergangenheit. Diese Eigenschaft der Gedächtnislosigkeit ist charakteristisch für so genannte *Markovprozesse*. Betrachtet man zustandsdiskrete Prozesse, so spricht man auch von *Markovketten* /MaBC-86/.

Eine quantitative Analyse eines SPN kann durchgeführt werden, indem man den korrespondierenden Markovprozess analysiert. Zu den Leistungskenngrößen eines SPN, die berechnet werden können, zählen beispielsweise:

- *Auslastung einer Transition*: gibt den Anteil der Zeit an, in der die Transition aktiviert ist.
- *Mittlerer Durchsatz einer Transition*: gibt die durchschnittliche Anzahl der Schaltvorgänge pro Zeiteinheit an. Dies entspricht der mittleren Schaltfrequenz.
- *Mittlere Markenanzahl in einem Platz*: diese Zahl wird häufig zur Berechnung von Auslastungen von Ressourcen benötigt.
- *Mittlere Antwortzeit einer Transition*: Zeit, die von der Ankunft einer Marke bis zu ihrem Weiterschalten im Mittel verstreicht.

Aufgrund der einfachen Umformung in einen Markovprozess kann die Auswertung eines SPN analytisch auf diesem Prozess stattfinden. Problematisch bei der analytischen Lösung wird allerdings, dass es zu einer hohen Zustandsanzahl kommen kann, da zusätzliche Plätze und Marken den Erreichbarkeitsgraphen stark vergrößern /Zieg-98/. In diesem Fall wird häufig auf die Simulation zur Bestimmung der Leistungskenngrößen zurückgegriffen.

Oftmals ist es nicht wünschenswert, mit jeder Transition eine zufällige Zeit zu assoziieren. Stattdessen würde man es vorziehen, Zeiten nur mit solchen Ereignissen zu assoziieren, von denen man glaubt, dass sie den größten Einfluss auf die Systemleistung haben. Im nächsten Abschnitt werden dazu einige Varianten von SPN vorgestellt.

2.2.2.3 Varianten von SPN

Die im Folgenden vorgestellten Varianten zeichnen sich dadurch aus, dass Transitionen unterteilt werden können in zeitlose Transitionen (immediate transitions), die ohne Verzögerung feuern, und zeitbehaftete Transitionen (timed transitions), die nach einer bestimmten Verzögerung feuern. Mögliche Konflikte zwischen zeitlosen Transitionen (IM) werden durch Zuweisung von Prioritäten und Gewichten gelöst. Verzögerungen von zeitbehafteten Transitionen werden durch deterministische Verzögerungen oder durch Zufallsvariablen spezifiziert. Wichtige Fälle sind Transitionen mit folgender Schaltzeitverteilung:

- *exponentiell (EXP):* zufällig verteilt mit exponentieller Verteilungsfunktion
- *deterministisch (DET):* die Schaltverzögerung ist fest
- *allgemein (GEN):* zufällig verteilt mit benutzerdefinierter Verteilungsfunktion

Eine bestimmte Klasse der allgemeinen Verteilung wird Expolynomialverteilung genannt und ist dadurch charakterisiert, dass sie stückweise durch exponentielle Polynome definiert sein kann. In /Zimm-01/ werden folgende Varianten von SPNs unterschieden:

- *Generalized SPNs (GSPNs):* Sowohl Transitionen ohne zeitliche Verzögerung als auch Transitionen mit exponentiell verteilter Schaltrate können spezifiziert werden.
- *Deterministic and SPNs (DSPNs):* DSPNs erweitern GSPNs um Transitionen mit deterministischer Verzögerung mit der Bedingung, dass in jeder Markierung höchstens eine von ihnen aktiviert ist.
- *Concurrent DSPNs (CDSPNs):* Transitionen mit exponentiell und deterministisch verteilter Schaltrate dürfen ohne Restriktionen aktiviert sein.
- *Extended DSPNs (EDSPNs):* Höchstens eine Transition mit expolynomiell verteilter Schaltrate darf in jeder Markierung aktiviert sein.

Petrinetze lassen sich auch mit Hilfe qualitativer Analysemethoden untersuchen, worauf im nächsten Abschnitt eingegangen wird.

2.2.3 Qualitative Analyse

Zu den qualitativ überprüfbaren Eigenschaften eines Petrinetzes zählen die Folgenden:

- *Lebendigkeit:* Die Lebendigkeit ist ein Synonym für das Fehlen totaler und partieller Verklemmungen /Schn-92/. Bei einer totalen Verklemmung ist keine Transition mehr schaltfähig, während eine partielle Verklemmung einer Markierung in einem Petrinetz entspricht, aus der heraus nicht mehr alle Transitionen aktiviert werden können.
- *Erreichbarkeit:* Eine Markierung M ist in einem Petrinetz erreichbar, wenn es eine Folge von Anwendungen der Schaltregel gibt, die M aus der Anfangsmarkierung M_0 erzeugt /Baum-90/.
- *Reversibilität:* Die Reversibilität bewertet die Erreichbarkeit von Netzzuständen. Ein durch ein reversibles Petrinetz beschriebenes System kann, ausgehend von jedem möglichen Zustand wieder in den Anfangszustand überführt werden, gleichgültig, welche Schaltvorgänge dazu erforderlich sind /Schn-92/.
- *Beschränktheit:* Ein Petrinetz heißt k-beschränkt, mit Bezug auf eine initiale Markierung M_0, wenn jeder Platz im Netz höchstens k Marken enthält für alle Markierungen in der Erreichbarkeitsmenge. Ein Spezialfall von Beschränktheit ist *Sicherheit*. Ein Petrinetz ist sicher, wenn es 1-beschränkt ist /DeJa-95/.

Zum Nachweis der oben beschriebenen Eigenschaften können Verfahren aus der Graphentheorie und der linearen Algebra herangezogen werden, auf die in den folgenden beiden Abschnitten eingegangen wird.

2.2.3.1 Analyse mittels Graphentheorie

Auf der Grundlage der Graphentheorie kann eine Reihe von Netzeigenschaften bestimmt werden. Hierzu zählen unter anderem die Erreichbarkeit einer Markierung, die Lebendigkeit, die Reversibilität und die Beschränktheit. Folgende Graphentypen können unterschieden werden:

- *Erreichbarkeitsgraph:* Der Erreichbarkeitsgraph ist ein gerichteter Graph, dessen Knoten die Markierungen wiedergeben, die ausgehend von der Anfangsmarkierung erreicht werden können. Die Kanten repräsentieren die Transitionen, deren Schalten eine Markierung in eine andere überführt /Schn-92/.
- *Kondensation:* Die Kondensation ist ein reduzierter Graph, der aus dem Erreichbarkeitsgraph gewonnen wird. Die Knoten der Kondensation entsprechen den starken Komponenten des Erreichbarkeitsgraphen und dessen Kanten geben deren Zuordnungen wieder. Unter den starken Komponenten werden diejenigen Knoten verstanden, zwischen denen in jeweils beiden Richtungen Verbindungswege existieren.
- *Überdeckungsgraph:* Der Überdeckungsgraph ist endlich und enthält jede erreichbare Markierung, ausgehend von der Anfangsmarkierung M_0, die entweder explizit durch einen Knoten dargestellt wird oder von einem Knoten mit Hilfe einer ω Notation „überdeckt" wird. Das ω Symbol repräsentiert eine Anzahl von Marken, die beliebig groß sein kann /DeJa-95/. Die Idee besteht darin, unendliche Pfade durch einen Knoten zu ersetzen.

Die oben beschriebenen Eigenschaften von Petrinetzen können mit Hilfe von Verfahren aus der Graphentheorie wie folgt nachgewiesen werden:

- Ein Petrinetz ist genau dann lebendig, wenn jede Senke (Knoten ohne auslaufende Kanten) der Kondensation des Erreichbarkeitsgraphen alle Transitionen enthält.
- Die Erreichbarkeit einer Markierung lässt sich unmittelbar aus dem Erreichbarkeitsgraphen ablesen.
- Notwendiges und hinreichendes Kriterium für die Reversibilität eines Petrinetzes ist der starke Zusammenhang des Erreichbarkeitsgraphen - die Kondensation darf demnach aus nur einem Knoten bestehen.
- Ein Petrinetz ist k-beschränkt bezüglich einer Anfangsmarkierung M_0 genau dann, wenn das ω Symbol nirgendwo im Überdeckungsgraphen vorkommt. Falls k=1 gilt, ist das Petrinetz sicher.

2.2.3.2 Analyse mittels linearer Algebra

Alternativ zur Analyse mittels Graphentheorie ermöglichen Methoden der linearen Algebra, Aussagen über Petrinetzeigenschaften auch direkt aus der Netzstruktur, das heißt nicht über den Umweg der Konstruktion gerichteter Graphen, abzuleiten. Grundlage dieser Verfahren ist das so genannte *Erreichbarkeitskalkül*, welches den Zusammenhang zwischen der Erreichbarkeit einer Markierung und der Lösung eines linearen Gleichungssystems herstellt. Das Petrinetz wird dabei in Form einer Netzmatrix N beschrieben, welche die Verknüpfungen zwischen den Stellen und den Transitionen eines Petrinetzes enthält.

Eine notwendige Bedingung für die Erreichbarkeit einer Markierung M ist die Lösbarkeit des Gleichungssystems

$$N \cdot V = M - M_0 \quad \text{mit } V = (v_i)_{1 \le i \le |T|} \qquad \textit{(Gleichung 2-1: Erreichbarkeitskalkül)}$$

in natürlichen Zahlen $v_1, v_2, \ldots, v_{|T|}$. Die Umkehrung gilt nicht unbedingt: falls Gleichung 2-1 eine nicht negative Lösung gefunden hat, so kann die Markierung M erreichbar sein von der

Anfangsmarkierung M_0 aus oder auch nicht. Falls keine Lösung gefunden wird, dann ist die gewünschte Markierung M nicht erreichbar.

Stellen- und Transitions-Invarianten sind Lösungen spezieller aus Gleichung 2-1 abgeleiteter Gleichungssysteme. Die *Stellen-Invarianten* charakterisieren bestimmte Stellenmengen in einem Petrinetz. Diese sind festgelegt durch die ganzzahligen Lösungen I_s von

$$N^T \cdot I_S = 0 \qquad \textit{(Gleichung 2-2: Stellen-Invarianten)}$$

Die *Transitions-Invarianten* beschreiben Transitionenmengen eines Petrinetzes und sind nicht negative, ganzzahlige Lösungen von

$$N \cdot I_T = 0 \qquad \textit{(Gleichung 2-3: Transitions-Invarianten)}$$

Die Stellen- und Transitions-Invarianten können verwendet werden, um Eigenschaften von Petrinetzen nachzuweisen. Aus Gleichung 2-2 lässt sich eine Interpretation der Lösungen mit Hilfe von Gleichung 2-1 ableiten. Hieraus folgt in wenigen Schritten die Beziehung

$$M^T \cdot I_S = M_0^T \cdot I_S \qquad \textit{(Gleichung 2-4)}$$

welche als Erhaltungsgleichung aufgefasst werden kann. Ein Vergleich mit Gleichung 2-1 zeigt, dass durch Transitions-Invarianten Schaltsequenzen in einem Netz beschrieben werden, die Markierungen reproduzieren. Die daraus abgeleiteten Folgerungen können wie folgt zusammengefasst werden:

- In einem Petrinetz mit der Struktureigenschaft nach Gleichung 2-2 unterliegt jeder Schaltvorgang einer durch die Stellen-Invariante I_s gewichteten Konstanz der Markenanzahl.
- Ein Petrinetz kann aufgrund seiner Transitions-Invarianten in reversible Teilnetze zerlegt werden, die als kleinste, eigenständig zu untersuchende Funktionseinheiten angesehen werden können. Ein reversibles Petrinetz besitzt daher mindestens eine von Null verschiedene Transitions-Invariante, ein lebendiges sogar eine, die alle Transitionen enthält /Schn-92/.

2.2.4 Direkte Parameteroptimierung von Simulationsmodellen

Dieser Abschnitt führt zunächst in die Vorgehensweise bei der Parameteroptimierung von Simulationsmodellen ein. Im Anschluss daran wird näher auf direkte Optimierungsmethoden eingegangen.

2.2.4.1 Vorgehensweise

Ein Optimierungsproblem kann als Tupel (S,F) formalisiert werden. Der Lösungsraum S gibt die Menge aller möglichen Problemlösungen an, die durch eine *Zielfunktion*

$$F : S \rightarrow R$$

bewertet werden. Das Problem besteht darin, eine Lösung i* zu finden, für die gilt:

$$\forall i \in S : F(i) \circ F(i^*) = F^* \qquad \circ \in \{\leq, \geq\}$$

Der Zielfunktionswert F(i*)=F* wird als globales Optimum von F bezeichnet, wobei es sich sowohl um ein globales Maximum ($\circ = \leq$) als auch um ein globales Minimum ($\circ = \geq$) handeln kann. Die Lösung i* heißt „global-optimale“ Lösung. Neben „global-optimalen“ Lösungen können auch „lokal-optimale“ Lösungen iˆ vorhanden sein, welche die Eigenschaft aufweisen, dass alle benachbarten Lösungen den gleichen oder einen schlechteren Zielfunktionswert haben. Falls es sich beim Suchraum um den R^n handelt, werden „global-optimale“ Lösungen als globale Optimumpunkte bezeichnet und „lokal-optimale“ Lösungen entsprechend als lokale Optimumpunkte. Zielfunktionen mit mehreren „global-optimalen“ und/oder „lokal-optimalen“ Lösungen heißen multimodale Funktionen. Ein Optimierungsproblem ist entweder ein Minimierungs- oder ein Maximierungsproblem. Minimierungsprobleme können in Maximierungsprobleme umgewandelt werden und umgekehrt aufgrund folgender Beziehung:

$$\min\{F(S)\} = -\max\{-F(S)\}$$

Ein Simulationsmodell beschreibt eine Beziehung zwischen Modelleingabeparametern und Modellausgabegrößen, die in der Regel nicht durch eine mathematische Funktion darstellbar ist. Daher kann ein Simulationsmodell als „Black-Box“-Funktion

$$f_M : S \subset R^n \rightarrow R^m$$

aufgefasst werden, die einen Vektor von Modelleingabeparametern

$$\vec{x} = (x_1, x_2, ..., x_n) \quad x_i \in R,\ i \in \{1, ..., n\}$$

auf mehrere Modellausgabegrößen

$$f_{M_j}(\vec{x}) \quad j \in \{1, ..., m\}$$

abbildet (Bild 2.2.2). Eine Funktion f_M, die auf einem Simulationsmodell basiert, wird im Folgenden *Modellfunktion* genannt. Modelleingabeparameter repräsentieren Systemparameter wie beispielsweise Bearbeitungszeiten oder Ressourcen. Modellausgabegrößen stellen Informationen über das Systemverhalten dar. Dazu zählen zum Beispiel Durchsatz, Wartezeiten oder Auslastung.

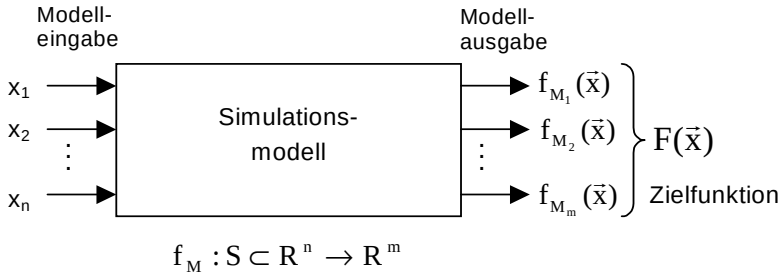


Bild 2.2.2 Simulationsmodell als „Black-Box“

Bei einer *Sensitivitätsanalyse* /Page-91/ wird untersucht, wie die Modellausgabe auf Veränderungen der Modelleingabe reagiert und von welchen Parametern die Modellergebnisse am stärksten abhängen. Dadurch ist es möglich, kritische Größen zu erkennen, die im Modell wie in der Realität besonders beachtet werden müssen. Des Weiteren lassen sich wesentliche Einflussgrößen von unwesentlichen trennen und Aussagen über die Prognosequalität des Modells gewinnen. Als formales Maß für die Sensitivität können die relativen Veränderungen eines Modellausgabeparameters bei Veränderung eines Modelleingabeparameters betrachtet werden. Diese Beziehung wird durch so genannte Sensitivitätskoeffizienten ausgedrückt.

Das Ziel bei der *Modelloptimierung* besteht darin, Modelleingabeparameterwerte zu finden, die zu einem optimalen Systemmodellverhalten oder zumindest zu einer Verbesserung führen. Die Optimierung eines Simulationsmodells kann demnach als ein Parameteroptimierungsproblem betrachtet werden. Die Problemlösungen sind Parametervektoren $\vec{x}$. Das Ziel besteht darin, einen Vektor $\vec{x}*$ zu finden, der zu einem optimalen Modellverhalten führt. Dies wird mit einer Zielfunktion F quantifiziert, die sich aus den Modellausgabegrößen $f_{M_j}(\vec{x})$, $j \in \{1,...,m\}$ zusammensetzt. Häufig wird die Zielfunktion als gewichtete Summe der Modellausgabegrößen wie folgt definiert:

$$F(\vec{x}) = \sum_{k=1}^{m} \omega_k \, f_{M_k}(\vec{x}) \qquad \omega_k \in R$$

Bild 2.2.3 illustriert den iterativen Prozess der Modelloptimierung /Syrj-97/. Dazu wird abwechselnd zwischen zwei eigenständigen, voneinander getrennt ablaufenden Prozessen hin und her geschaltet. Es handelt sich dabei zum einen um den Optimierungsprozess, der Parametervektoren generiert, und zum anderen um den Simulationsprozess, dessen Aufgabe darin besteht, die erzeugten Parametervektoren zu bewerten. Ziel dieses alternierenden Prozesses ist es, die durch das Simulationsmodell vorgegebene Zielfunktion schrittweise zu verbessern und innerhalb möglichst weniger Iterationsschritte eine möglichst „global-optimale“ Lösung zu bestimmen.

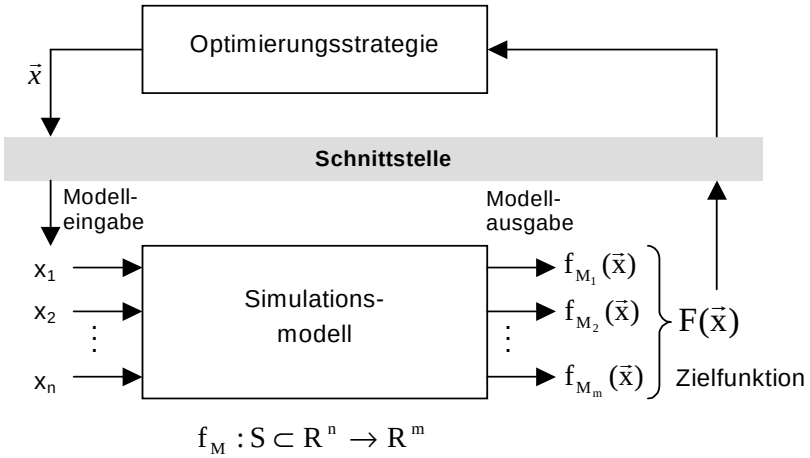


Bild 2.2.3 Der iterative Prozess der Modelloptimierung

Für die Optimierung von Simulationsmodellen werden in der Regel ableitungsfreie iterativ arbeitende Optimierungsstrategien eingesetzt. Diese so genannten direkten Optimierungsme-

thoden benötigen im Gegensatz zu analytischen Optimierungsmethoden keine mathematische Repräsentation der zu optimierenden Zielfunktion. Im folgenden Abschnitt werden einige bekannte Vertreter dieser Verfahrensklasse vorgestellt.

2.2.4.2 Direkte Optimierungsmethoden

Eine Unterscheidung unter den direkten Optimierungsmethoden kann danach erfolgen, ob nach einem globalen oder einem lokalen Optimum gesucht wird. Ausgehend von den globalen und lokalen Optimierungsmethoden lassen sich verschiedene Realisierungsalternativen für hybride Optimierungsstrategien zusammenstellen. Des Weiteren wird zwischen einstufigen und mehrstufigen Optimierungsmethoden unterschieden.

Globale Optimierung

Die globale Optimierung hat zum Ziel, zu einem Optimierungsproblem das globale Optimum der Zielfunktion aufzufinden. Zu den bekanntesten Strategien zur globalen Optimierung zählen heute:

- Evolutionäre Algorithmen (EA)
- Simulated Annealing (SA)
- Monte-Carlo Verfahren (MC)

Unter dem Begriff *Evolutionäre Algorithmen* /Hafn-98/ werden *Genetische Algorithmen (GA)* /Gold-89/ und Evolutionsstrategien zusammengefasst. Diese Optimierungsmethoden ahmen Prinzipien aus der biologischen Evolution nach, um schwierige Optimierungsaufgaben zu lösen. In der Begriffswelt der Evolutionären Algorithmen heißt eine zulässige Parametereinstellung Individuum. Dieses stellt somit einen Satz von Entscheidungsvariablen dar bzw. repräsentiert einen Punkt im Lösungsraum. Eine Menge von Individuen bildet die Population, die sich unter Verwendung folgender (genetischer) Operatoren verändert: Die Mutation nimmt leichte Veränderungen der genetischen Information eines Individuums vor, nachdem die Rekombination genetisches Material meist zweier Eltern neu zu einem Nachkommen zusammengesetzt hat. Die verschiedenen Klassen evolutionärer Algorithmen unterscheiden sich durch die Repräsentation der Individuen und durch die auf ihr arbeitenden Operatoren. Bild 2.2.4 stellt das allgemeine Iterationsschema eines Evolutionären Algorithmus dar /Hafn-98/. Nachdem eine Anfangspopulation generiert wurde, wird die Schleife solange durchlaufen, bis eine Terminierungsbedingung greift. Die Paarungsselektion wählt zufällig zwei oder mehr aus jenen Individuen aus, welche die Umweltselektion des vorherigen Schleifendurchlaufes überlebt haben. Die ausgewählten Individuen dürfen ihre Informationen zu einem Nachkommen rekombinieren. Anschließend sorgt die Mutation für eine (in der Regel geringe) Modifikation der Informationen, die dann anhand der Zielfunktion bewertet werden. Dieses Maß bildet die Basis für die Umweltselektion, die nur einem Teil der Population erlaubt, in der nächsten Iteration Nachkommen zu erzeugen. Die Grundidee besteht darin, durch Kombination von Eigenschaften ausgewählter Elternlösungen neue Lösungsalternativen zu generieren und dabei mit fortschreitender Generationszahl in immer bessere Regionen des Lösungsraumes vorzudringen.

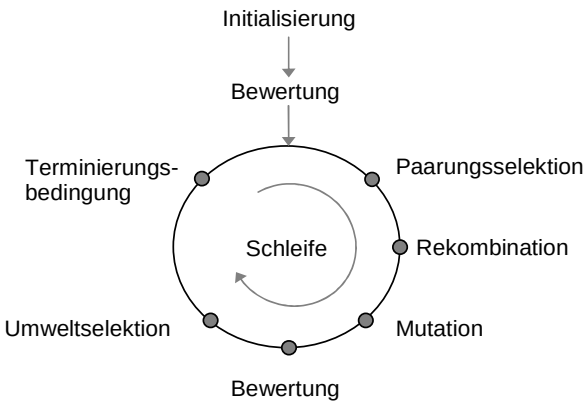


Bild 2.2.4 Allgemeines Iterationsschema eines Evolutionären Algorithmus

Das *Simulated Annealing* /AaKo-89/ ist ebenfalls ein probabilistisches Verfahren zur globalen Optimierung. Das Konzept basiert auf der Analogie zu einem physikalischen Abkühlungsprozess, bei dem Materie zuerst soweit erhitzt wird, dass der Schmelzpunkt erreicht wird, um sie dann wieder langsam abkühlen zu lassen, mit dem Ziel, einen Zustand minimaler Energie zu erreichen. Dieser ergibt sich, wenn die Materie so auskristallisiert, dass die Atome in einer perfekten Gitterstruktur angeordnet sind.

Bei der einfachsten Form des *Monte-Carlo Verfahrens* wird eine bestimmte Anzahl von Suchpunkten im Lösungsraum zufällig generiert. Anschließend wird zu jedem erzeugten Suchpunkt der Zielfunktionswert ermittelt und als Ergebnis der Suchpunkt ausgegeben, der den besten Zielfunktionswert aufweist.

Lokale Optimierung

Bei der lokalen Optimierung besteht das Ziel darin, ausgehend von einer bestimmten Startlösung, durch schrittweise Verbesserung der Zielfunktion eine in der Nachbarschaft der Startlösung gelegene „lokal-optimale“ Lösung zu bestimmen. Zu den bekanntesten Vertretern lokaler Optimierungsverfahren gehören die so genannten *Hill-Climbing (HC)* Verfahren. Unter dem Begriff Hill-Climbing werden direkte Parameteroptimierungsstrategien zusammengefasst, deren Vorgehensweise bei der Suche nach einem Maximum weitestgehend der intuitiven Art eines (blinden) Bergsteigers entspricht, der sich von einem Tal aus zum höchsten Gipfel eines Berges empor tastet. Eine häufig eingesetzte Hill-Climbing Strategie ist die Mustersuche von Hooke und Jeeves /HoJe-61/.

Hybride Optimierungsstrategien

Ausgehend von den globalen und lokalen Optimierungsverfahren lassen sich verschiedene Realisierungsalternativen für hybride Optimierungsstrategien zusammenstellen, bei denen die Optimierung in eine Vor- und eine Feinoptimierungsphase unterteilt ist. Die Voroptimierung wird beispielsweise von einem Genetischen Algorithmus ausgeführt und ist über ein Umschaltverfahren mit dem Feinoptimierer, einem Hill-Climbing-Algorithmus, verbunden. Basierend auf den Ergebnissen der Voroptimierung wird für den Hill-Climbing-Algorithmus ein Startpunkt ausgewählt. Eventuell wird der Hill-Climbing-Algorithmus auch an mehreren

Punkten gestartet. Dieser lokalisiert durch effiziente Punkt-zu-Punkt-Suche den beziehungsweise die entsprechenden Optimumpunkt(e) mit einer vom Anwender vorgegebenen Genauigkeit. Realisierungsalternativen für diese so genannten kombinierten 2-Phasen-Strategien sind beispielsweise:

- Genetische Algorithmen & Hill-Climbing (GA+HC)
- Simulated Annealing & Hill-Climbing (SA+HC)

Mehrstufige sequentielle Optimierung

Bei der mehrstufigen sequentiellen Optimierung wird durch mehrfaches Hintereinanderausführen einer einstufigen Optimierungsstrategie (GA, SA, MC, HC, kombinierte 2-Phasen-Strategie) eine Folge von Optimumpunkten ermittelt. Bei der monotonen Variante muss für jeden neu aufgefundenen Optimumpunkt garantiert werden, dass dessen Zielfunktionswert entweder gleich gut ist oder eine Verbesserung gegenüber dem Zielfunktionswert des vorangegangenen Optimumpunkts darstellt. Bei der nichtmonotonen Variante spielt dies keine Rolle. Weitere Details sind in /Syrj-97/ beschrieben.

2.3 Web- und Komponententechnologien in der Modellierung und Simulation

Dieser Abschnitt führt in die Grundlagen der Web- und Komponententechnologien ein. Dazu werden zunächst jeweils die grundlegenden Begriffe erläutert und Abgrenzungen vorgenommen. Im Anschluss daran werden Anwendungsmöglichkeiten in der Modellierung und Simulation (M&S) aufgezeigt.

2.3.1 Web-Technologien

Die folgenden Abschnitte behandeln das Thema Web-Technologien und deren Einfluss auf den Bereich der Modellierung und Simulation.

2.3.1.1 Das Internet

Die Entwicklung des Internet geht auf ein Projekt der US-Behörde im Jahre 1968 zurück, welches seinerzeit vier Rechner miteinander verband. Ziel war es, im Auftrag des US-amerikanischen Verteidigungsministeriums (Department of Defense, DoD) ein Netzwerk zur Verbindung unterschiedlicher Rechner zu entwickeln. 1973 startete die DARPA (Defense Advanced Research Projects Agency) ein weiteres Projekt zur Entwicklung von einheitlichen Techniken zur Verbindung von paketvermittelnden Netzen. In diesem Projekt wurde mit TCP/IP (Transmission Control Protocol/Internet Protocol) ein Standardprotokoll entwickelt, bei dem Wert darauf gelegt wurde, dass es von verschiedenen Rechnertypen verstanden wird. Aufgrund von Zugangsbeschränkungen seitens des DoD zum ARPAnet gründete die National Science Foundation (NSF) 1981 ein nicht-kommerzielles Forschungsnetz namens CSnet. Dieses Netz ist bis heute Bestandteil des Internets, während sich der militärische Teil, das MILnet, vom ARPAnet 1983 abspaltete. Anfang der 80er Jahre wurde TCP/IP vom DoD zum

nationalen Standard erklärt. Mit dem 1986 von der NSF gegründeten NSFnet bildet das MILnet das Hauptrückgrat des Internets, wobei das NSFnet selbst aus vielen Einzelnetzwerken besteht und das größte Teilnetzwerk im Internet darstellt. ARPAnet, MILnet sowie weitere Militärnetzwerke bilden unter Verwaltung des DoD das Defense Data Network (DDN), dessen Network Information Center (NIC) für globale Fragen zum Internet zuständig ist. Das Internet ist somit ein Überbegriff für alle Rechner und Teilnetze, wie z.B. das USENET in den USA oder das Wissenschaftsnetz (WIN) in Deutschland, die über TCP/IP miteinander kommunizieren.

Heute werden im Internet verschiedene Dienste angeboten, die auf dem Client/Server-Prinzip beruhen. Ein Client ist ein Computer, von dem aus ein Benutzer einen Dienst im Internet aufrufen möchte. Da die Dienste und Informationen des Internets nicht lokal auf dem Rechner des Benutzers liegen, muss er mittels eines Client-Programms eine Verbindung zu dem Rechner aufbauen, der diesen Dienst oder die gesuchten Informationen anbietet. Dieser Rechner wird Server genannt. Die Kommunikation zwischen den Rechnern erfolgt dabei über das TCP/IP Protokoll. Einen sehr populären Informationsdienst im Internet stellt heutzutage das World Wide Web, auch bekannt als „WWW“, „Web“ oder „W3“ dar, welches auf ein Projekt Anfang der 90er Jahre am CERN (Centre Européen de Recherche Nucléaires) in Genf unter der Leitung von Tim Berners-Lee zurückgeht. Dieser gründete 1994 das World Wide Web Consortium (W3C) mit dem Ziel, Web-Standards und -Technologien zu entwickeln /W3Co-03a/. Der nächste Abschnitt befasst sich näher mit einzelnen Web-Technologien und deren Anwendungsmöglichkeiten in der Modellierung und Simulation.

2.3.1.2 Web-Technologien und ihre Anwendungsmöglichkeiten in der Modellierung und Simulation

Durch den Einsatz von Hypertext- und Multimedia-Techniken wird dem Benutzer ein einfacher Zugang zu Dokumenten und Dateien im Internet ermöglicht. Aufbauend auf der Auszeichnungssprache HTML (Hypertext Markup Language) kann ein Text mit Auszeichnungsbefehlen (Tags) wie beispielsweise fett, kursiv, Überschriften verschiedener Größen, Listen usw. versehen werden, die vom Browser in der Darstellung umgesetzt werden. Des Weiteren können Textstellen als so genannte Verweise (Links) definiert werden, die beim Anklicken eine vorher festgelegte Aktion auslösen. Dabei kann es sich um Verweise auf Stellen im gleichen Dokument handeln oder um Verweise auf andere Dokumente bzw. Dateien im Internet. Diese müssen nicht unbedingt im Hypertext-Format vorliegen, sondern können beispielsweise auch Postscript-Dokumente oder Bild-, Video- und Musik-Dateien sein. Zum Anzeigen und/oder Abspielen dieser Dokumente bzw. Dateien müssen sich die Browser allerdings externer Programme bedienen. Auf diese Weise lassen sich bereits vorhandene Informationen schnell und ohne Konvertierung zugänglich machen. Die Form der Verweise ist standardisiert und wird als so genannte URL (Uniform Ressource Locator) bezeichnet. Um Texte in das HTML-Format zu übersetzen, existieren mittlerweile eine Reihe von Konvertierungswerkzeugen. Neben den oben erwähnten Auszeichnungsbefehlen sind in HTML auch Formularelemente definiert, welche Eingaben des Benutzers an den Server senden. Dort werden die Eingaben über so genannte CGI-Scripts (Common Gateway Interface) ausgewertet und an ein Programm zur Bearbeitung weitergegeben. Dieses liefert seine Ausgabe dann entweder direkt

als HTML-Dokument oder schickt sie unformatiert an das Script zurück, welches die Formatierung vornimmt und das Ergebnis an den Benutzer zurückschickt. Durch diese Schnittstelle lassen sich beispielsweise leicht Oberflächen zur Bedienung textorientierter Programme realisieren. CGI-Programme werden heute jedoch zunehmend durch modernere auf Java basierende Technologien wie beispielsweise Java-Servlets verdrängt. Java ist eine objektorientierte Programmiersprache, mit der sich Anwendungen erstellen lassen, die sowohl auf einem einzelnen Computer ablaufen können als auch auf mehreren untereinander vernetzten Rechnern verteilt sein können. Java eignet sich auch zur Programmierung kleinerer Anwendungsmodule (Applets) als Elemente einzelner Web-Seiten. Die Haupteigenschaften von Java sind /Java-03/:

- Java-Programme sind flexibel in einem Netzwerk übertragbar, da sie in Bytecode kompiliert werden, der plattformunabhängig ist. Der Bytecode kann auf jedem Rechner eines Netzwerks laufen, der über eine virtuelle Java-Maschine (JVM) verfügt. Diese virtuelle Maschine übersetzt den Bytecode in einen Code, der von der jeweiligen Computerhardware verarbeitet werden kann. Dadurch spielen Unterschiede zwischen den einzelnen Computerplattformen keine Rolle mehr und es sind keine verschiedenen Programmversionen nötig.
- Java-Code enthält keine Zeiger, die in verbotene Speicherbereiche verweisen und somit das Betriebssystem beschädigen könnten. Der Java-Interpreter jedes Betriebssystems führt an den einzelnen Objekten eine Reihe von Kontrollen durch, um die Integrität zu gewährleisten.
- Java ist objektorientiert, so dass leicht wiederverwendbare Objekte (JavaBeans) erstellt werden können.
- Mit dem Applet-Konzept bietet Java die Möglichkeit, Anwendungen innerhalb von Web-Browsern auszuführen.

Java wurde 1995 von Sun Microsystems eingeführt und eröffnet Anwendern neue interaktive Möglichkeiten im Web. Die meisten führenden Entwickler von Betriebssystemen bieten seither Java-Compiler als Teil ihrer Produktpalette an. Ebenso enthalten Browser von Microsoft und Netscape integrierte JVM, so dass sie in HTML-Seiten eingebettete Applets ausführen können. Neben der virtuellen Maschine verfügt Java über einen Just-In-Time-Compiler, der den Java-Bytecode auf der jeweiligen Plattform, auf der das Programm laufen soll, zur schnelleren Ausführung vorbereitet.

Im Bereich der Modellierung und Simulation gewinnen Web-Technologien immer mehr an Bedeutung. Dieser Trend spiegelt sich in zahlreichen zu diesem Thema stattfindenden Konferenzen wider /FiHS-98/, /BrUP-99/, /SiBl-00/, /SiWM-01/, /SiWi-02/. Bild 2.3.1 gibt einen grundlegenden Überblick über die heute existierenden Web-Technologien und deren Anwendungsmöglichkeiten in der Modellierung und Simulation. Dazu zählen neben Online-Simulationen und –Animationen auch Java-basierte Simulationspakete wie beispielsweise Simjava /SimJ-03/, Silk /Silk-03/ und JavaSim /JaSi-03/. Zur dreidimensionalen Visualisierung von simulierten Abläufen können Web3D-Technologien eingesetzt werden, zu denen VRML (Virtual Reality Modeling Language), Java3D und X3D zählen. Für den Datenaustausch zwischen Anwendungen aus dem Bereich der Modellierung und Simulation ist die Entwicklung

standardisierter Modellaustauschformate von großer Bedeutung, worauf im folgenden Abschnitt näher eingegangen wird.

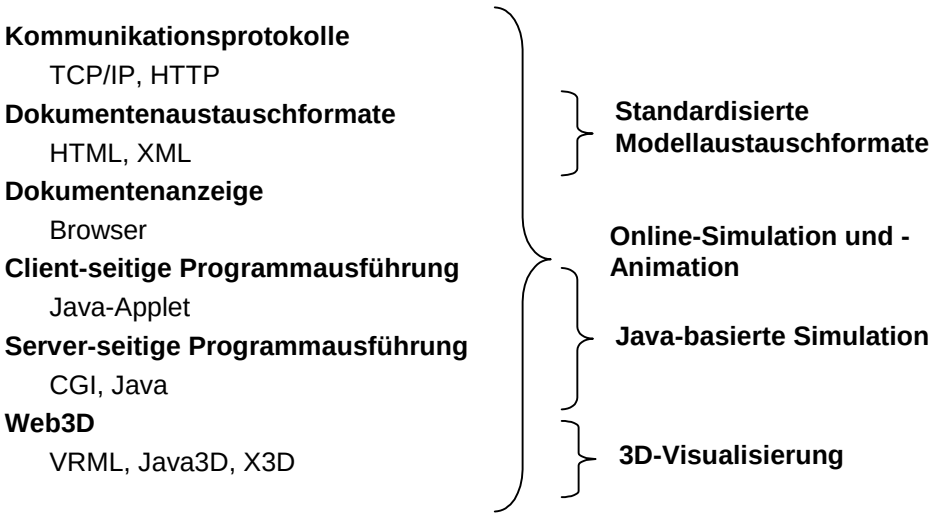


Bild 2.3.1 Web-Technologien und ihre Anwendungsmöglichkeiten in der M&S

2.3.1.3 Standardisierte Modellaustauschformate

Innerhalb nur weniger Jahre wurde die eXtensible Markup Language (XML), welche vom World Wide Web Consortium (W3C) im Februar 1998 verabschiedet wurde, zu einem weitverbreiteten Standardformat für den Datenaustausch /Brad-98/. Aufgrund der Eigenschaft von XML, Daten in jeder erdenklichen Struktur darstellen zu können, fand XML eine weite Verbreitung zur Speicherung von Dokumenten und zur Integration von Anwendungen. Beispiele für Auszeichnungssprachen aus unterschiedlichen Anwendungsbereichen sind MathML (Mathematical Markup Language), CML (Chemical Markup Language) oder AIML (Astronomical Instrument Markup Language). Die Verwendung von XML hat vor allem die folgenden Vorteile:

- XML ist ein offener Technologie-Standard, der firmen- und plattformunabhängig ist.
- XML unterstützt den internationalen Zeichensatz-Standard des erweiterten ISO Unicodes.
- Der XML-Standard ist programmiersprachen- und API (Application Programming Interfaces)-neutral. Für die Erzeugung und Verarbeitung von XML existieren eine Reihe von APIs. Beispiele dafür sind DOM (Document Object Model), SAX (Simple API for XML) und Web-DAV (Web-based Distributed Authoring and Versioning).
- XML verfügt über die Möglichkeit, Regeln für die Struktur eines Dokuments zu definieren. Damit ist es möglich, mit Hilfe geeigneter Werkzeuge, ein XML-Dokument hinsichtlich seiner spezifizierten Grammatik zu validieren.
- Ein XML-Dokument enthält keine Informationen über die Art und Weise der visuellen Darstellung. Stattdessen werden Technologien wie XSL (eXtensible Stylesheet Language) angewandt, mit deren Hilfe sich Stilvorlagen (Stylesheets) erstellen lassen.

- Zu XML gibt es eine wachsende Anzahl von Unterstützungswerkzeugen wie beispielsweise verschiedene Arten von Parsern.

Die großen internationalen Standardisierungsorganisationen haben die Vorzüge von XML bereits erkannt und sind heute dabei, zu den verschiedensten Modellierungstechniken einheitliche XML-basierte Austauschformate zu erarbeiten. Die ISO (International Organisation for Standardisation) beispielsweise treibt unter anderem die Standardisierung eines Modellaustauschformats für erweiterte Petrinetze voran. Der aktuelle Stand dieser noch laufenden Arbeiten kann im World Wide Web unter /PNTS-03/ eingesehen werden. Ein XML-basiertes Austauschformat ermöglicht nicht nur Interoperabilität zwischen Modellierungswerkzeugen, sondern vereinfacht auch die Integration existierender Alt-Werkzeuge, die auf derselben Modellierungstechnik basieren aber jeweils unterschiedliche Modellbeschreibungsformate verwenden.

Wie in Bild 2.3.2 angedeutet, steigt bei Verwendung eines standardisierten Modellaustauschformats die Anzahl der zur Werkzeugintegration benötigten Konverter K_i, $i \in N$ mit zunehmender Werkzeuganzahl lediglich linear an. Ohne standardisiertes Modellaustauschformat würde die Zahl der benötigten Konverter dagegen quadratisch ansteigen (um n unterschiedliche Modellbeschreibungsformate ineinander zu überführen, werden $(n^2-n)/2$ bidirektionale Konverter benötigt).

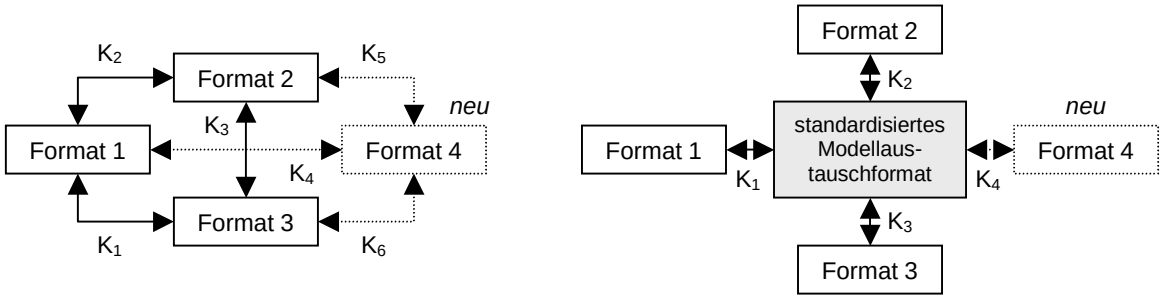


Bild 2.3.2 Gegenüberstellung des Konvertierungsaufwands ohne und mit standardisiertem Modellaustauschformat

2.3.2 Komponenten-Technologien

Die folgenden Abschnitte führen in den Bereich der Komponenten-Technologien ein und zeigen deren Anwendungsmöglichkeiten im Bereich der Modellierung und Simulation auf.

2.3.2.1 Komponentenorientierte Softwareentwicklung

Hinter dem Begriff komponentenorientierte Softwareentwicklung verbirgt sich ein Softwareentwicklungsansatz, bei dem Softwaresysteme aus genormten Einzelkomponenten baukastenartig zusammengesetzt werden. Idealer Weise müssen die einzelnen Komponenten nicht selbst entwickelt werden, sondern liegen bereits vor oder können von Drittherstellern erworben werden. Bei der Entwicklung von Softwaresystemen können komponentenorientierte An-

sätze auf verschiedenen Abstraktionsebenen eingesetzt werden. Shaw und Garlan unterscheiden bei der Softwareentwicklung die folgenden drei Ebenen /ShGa-96/:

- Architektur
- Programmcode
- Ausführungsmodul

Die Art der Komponenten ist auf den verschiedenen Ebenen sehr unterschiedlich. Die unterste Ausführungsmodul-Ebene behandelt Fragen, die das Speicherabbild eines Programmmoduls, Registerbelegungen, Aufbau des Call-Stack beim Aufruf von Programmroutinen und anderen „Low-Level“-Aspekten betreffen. Komponenten dieser Ebene sind entsprechend Binärcodekomponenten. Die nächsthöhere Programmcode-Ebene betrachtet dagegen die Struktur eines einzelnen Programms. Komponenten dieser Ebene sind die primitiven Sprachelemente der verwendeten Programmiersprache. In der Vergangenheit setzten sich Softwareentwickler hauptsächlich mit diesen beiden unteren Ebenen auseinander. Komponenten wurden im Wesentlichen als ausführbare Codemodule angesehen. Auch in aktuelleren Arbeiten wird Komponentenorientierung teilweise lediglich als eine besondere Implementierungstechnik verstanden /Grif-98/. Komponentenorientierte Techniken können aber auch sinnvoll auf der abstrakten Softwarearchitektur-Ebene eingesetzt werden /Turo-99/. Gerade komplexe und anspruchsvolle Softwaresysteme lassen sich durch eine komponentenorientierte Modellierung einfacher und übersichtlicher handhaben, da komponentenorientierte Softwarearchitekturen vom konkreten Programmcode abstrahieren und eine klare Trennung von architekturspezifischen und implementierungsspezifischen Aspekten ermöglichen.

2.3.2.2 Der Komponentenbegriff

Der Begriff Komponente wird in der Literatur auf vielfältige Art und Weise verwendet. In einigen Definitionen ist der Begriff sehr eng an Aspekte der Implementierung und Code-Wiederverwendung gebunden. Die Unified Modeling Language (UML) beispielsweise definiert eine Komponente wie folgt:

> *A physical, replaceable part of a system that packages implementation and conforms to and provides the realization of a set of interfaces /RuJB-99/.*

In UML ist eine Komponente eine physische Einheit eines Computersystems, in der Softwarecode gebündelt wird. Der Begriff Code umfasst dabei Quellcode, Binärcode, ausführbare Programmmodule, Scripte und Befehlsdateien. Typische Komponenten in UML sind Bibliotheken wie DLL’s (Dynamic Link Library) und JAR’s (Java Archive) oder auch einzelne Dateien, wie beispielsweise Java Class-Files. In UML wird darüber hinaus zwischen zustandslosen Komponenten und Komponenten mit eigenem Zustand und eigener Identität (Identity Component) unterschieden. Shaw und Garlan fassen den Komponentenbegriff weiter. Sie begrenzen ihn nicht auf Code-Komponenten, sondern verwenden Komponenten als konzeptionelle Modellelemente unabhängig von der Art ihrer Implementierung.

Allgemeiner Konsens herrscht darüber, dass eine Komponente eine (wieder)verwendbare Einheit eines Softwaresystems darstellt, die ihre Implementierungsdetails verbirgt und mit ihrer Umgebung ausschließlich über explizite Schnittstellen kommuniziert. In /Bäum-98/ wird

eine weitere Eigenschaft von Komponenten betont, die besagt, dass eine Komponente neben ihrer technischen Interpretation auch eine fachliche Interpretation besitzt. In einem anwendungsfachlichen Kontext erbringt eine Komponente eine definierte fachliche Dienstleistung. Komponenten werden als Grundbausteine zur Modellierung von Systemarchitekturen eingesetzt, die eine von Implementierungsdetails abstrahierende Sicht auf komplexe Softwaresysteme fördern. Im Rahmen dieser Arbeit soll eine Komponente wie folgt verstanden werden:

> *Eine Komponente ist eine benannte, (wieder)verwendbare Einheit des Architekturmodells, die innerhalb eines Softwaresystems definierte fachliche Dienstleistungen erbringt. Eine Komponente besitzt eine Menge explizit definierter und benannter Schnittstellen, über die sie mit ihrer Umwelt interagiert und hinter denen Details der Komponentenimplementierung verborgen bleiben.*

2.3.2.3 Komponentenorientiert vs. objektorientiert

In diesem Abschnitt wird der Frage nachgegangen, worin sich komponentenorientierte von objektorientierten Ansätzen unterscheiden. Hauptelemente der objektorientierten Modellierung sind Klasse und Objekt. Nach /Züll-98/ sollen Klassen hierbei die fachlichen Konzepte und Begriffe, die hinter den Gegenständen der täglichen Arbeit stehen, modellieren. In einer objektorientierten Programmiersprache stellen Klassen aber auch immer das grundsätzliche technische Konstrukt der Programmierung dar. Dadurch haben Klassen eine doppelte Rolle. Mit Klassen werden nicht nur abstrakte fachliche Aspekte, sondern auch alle konkreten softwaretechnischen Aspekte beschrieben. Die enge Beziehung zwischen der objektorientierten Modellierung und der zur Konstruktion verwendeten objektorientierten Programmiersprache erleichtert zwar den Übergang vom Modell zur Implementierung, aber in vielen Bereichen erweist sich eine Klasse als Modellierungselement für komplexe Konzepte als zu feingranular. Sobald zur Modellierung eines fachlichen Konzepts nicht nur eine einzelne Klasse, sondern eine ganze Kollektion von Klassen erforderlich ist, verwischt eine rein klassenbasierte Modellierung wichtige Architekturaspekte des Systems, da das fachliche Konzept als Ganzes nicht als separates Modellelement existiert.

Bei der komponentenorientierten Modellierung stellen Komponenten und Komponentenklassen die wesentlichen Modellelemente dar. Mit ihnen ist es möglich, eine weitere Abstraktion oberhalb von Objekten und Klassen zu beschreiben. Eine Komponentenimplementierung umfasst in der Regel mehrere Klassen und erlaubt damit auch die explizite Modellierung komplexer fachlicher Konzepte. Wichtig für die Modellierung mit Komponenten ist, dass sie weitgehend implementierungsunabhängig spezifiziert werden. Die Betonung liegt stärker auf den Schnittstellen, über die Komponenten miteinander interagieren, als auf deren Implementierung. In diesem Sinne können Komponentenorientierung und Objektorientierung als zwei sich ergänzende Techniken aufgefasst werden.

Eine nähere Betrachtung zeigt, dass auf der Implementierungsebene auch bei objektorientierten Techniken größere Einheiten als einzelne Klassen existieren. So genannte Rahmenwerke implementieren allgemeine generische Lösungen zur Verwendung in einem bestimmten Kontext. In einem Softwaresystem interagieren hierbei meist mehrere Rahmenwerke in geordneter Weise miteinander, um die Gesamtfunktionalität zu erbringen. Die Interaktion zwischen den Rahmenwerken lässt sich beispielsweise mit Entwurfsmustern beschreiben

/Bäum-98/. Hier werden Parallelen zu Komponenten deutlich. Auch Komponenten kapseln allgemeine generische Lösungen, die in einem größeren Kontext ihre Wiederverwendung finden. Der wesentliche Unterschied besteht darin, dass eine Komponentenklasse ein Konzept auf der Modellierungsebene beschreibt, während ein Rahmenwerk ein Konzept implementiert. Rahmenwerke sind daher bei einem objektorientierten Implementierungsmodell ein geeignetes Mittel zur Realisierung von Komponentenklassen.

Objektorientierte Techniken sind auch bei der komponentenorientierten Softwareentwicklung nicht überflüssig geworden. Vielmehr lassen sich Komponenten und deren Interaktionen mit den heute zur Verfügung stehenden Softwaretechniken am effektivsten mit objektorientierten Sprachen realisieren.

2.3.2.4 High Level Architecture

Zur Entwicklung und Ausführung von Komponenten wird durch einen Komponentenstandard ein verbindlicher Rahmen festgelegt. Dieser stellt strukturelle Anforderungen hinsichtlich Verknüpfungs- bzw. Kompositionsmöglichkeiten sowie verhaltensorientierte Anforderungen hinsichtlich Kollaborationsmöglichkeiten an die Komponenten /GrTh-00/. Im Bereich der Modellierung und Simulation hat das US-amerikanische Verteidigungsministerium im September 1996 mit der *High Level Architecture (HLA)* /KuWD-00/ einen Standard herausgegeben, der spezielle zum Aufbau verteilter Simulationsanwendungen erforderliche Infrastrukturdienste anbietet. Ein in diesem Zusammenhang sehr wichtiger Dienst ist beispielsweise das in der HLA integrierte Zeitmanagement, das die Synchronisation der an einer verteilten Simulation beteiligten Komponenten zur Laufzeit erlaubt. Im Gegensatz zu anderen Technologien und Konzepten aus der verteilten Simulation, die vor allem eine bestmögliche Parallelisierung auf Modellebene als Zielsetzung haben, besteht das Hauptanliegen der HLA nicht in der Steigerung der Ausführungsgeschwindigkeit, sondern vielmehr darin, den auch im Simulationsbereich immer lauter werdenden Forderungen nach Wiederverwendbarkeit und Interoperabilität gerecht zu werden. Dazu werden von der HLA eine Reihe neuer Ansätze und Mechanismen bereitgestellt, die es erlauben, Computermodelle aus modularen Teilmodellen aufzubauen. Diese Teilmodelle können auf beliebigen heterogenen Simulationssystemen implementiert sein, sofern diese eine HLA-Anbindung besitzen.

Bild 2.3.3 gibt einen Überblick über die wesentlichen Bestandteile der HLA, deren Grundidee in der Trennung von Simulationsfunktionalität und Infrastruktur für die Interoperabilität besteht. Die Teilnehmer eines gemeinsamen Simulationslaufs werden in der HLA-Terminologie als *Föderierte (Federates)* bezeichnet. Der Zusammenschluss mehrerer Föderierter zu einer Gesamtsimulation stellt eine so genannte *Föderation (Federation)* dar. Ein großer Vorteil der HLA besteht in der Offenheit für verschiedene Arten von Föderierten. Dabei muss es sich nicht unbedingt um Simulationen handeln, sondern Föderierte können auch andere Softwarebausteine (Datenbanken, etc.) oder Systeme der realen Außenwelt (Hardware aller Art) darstellen. Eine Föderation kann als ein Vertrag zur Durchführung eines Simulationslaufs (*Federation Execution*) zwischen den an der Simulation beteiligten Föderierten angesehen werden. In diesem Vertrag sind die Objektmodelle der Föderierten *(Simulation Object Model (SOM))* und der Gesamtföderation *(Federation Object Model (FOM))* in einer definierten dem *Object Model Template (OMT)* entsprechenden Form festgelegt. Darüber hinaus

legt die HLA zwingende Verhaltensregeln für Föderierte und Föderationen fest. Dieser HLA-Regelsatz, der aus insgesamt 10 Regeln besteht, beschreibt die Art und Weise, wie die HLA zu verwenden ist. Beispielsweise muss, um Interoperabilität und Plattformunabhängigkeit zu erreichen, die Kommunikation zwischen Föderierten ausschließlich über die so genannte *Runtime Infrastructure (RTI)* erfolgen. Um mit dieser in Kontakt zu treten, müssen die Föderierten eine einheitliche Schnittstelle (*HLA Interface Specification*) aufweisen. Durch die HLA-Schnittstellenspezifikation werden die von der RTI zur Verfügung gestellten Kommunikations- und Koordinationsdienste sowie die von den Föderierten zu erbringenden „Callback“-Funktionen beschrieben.

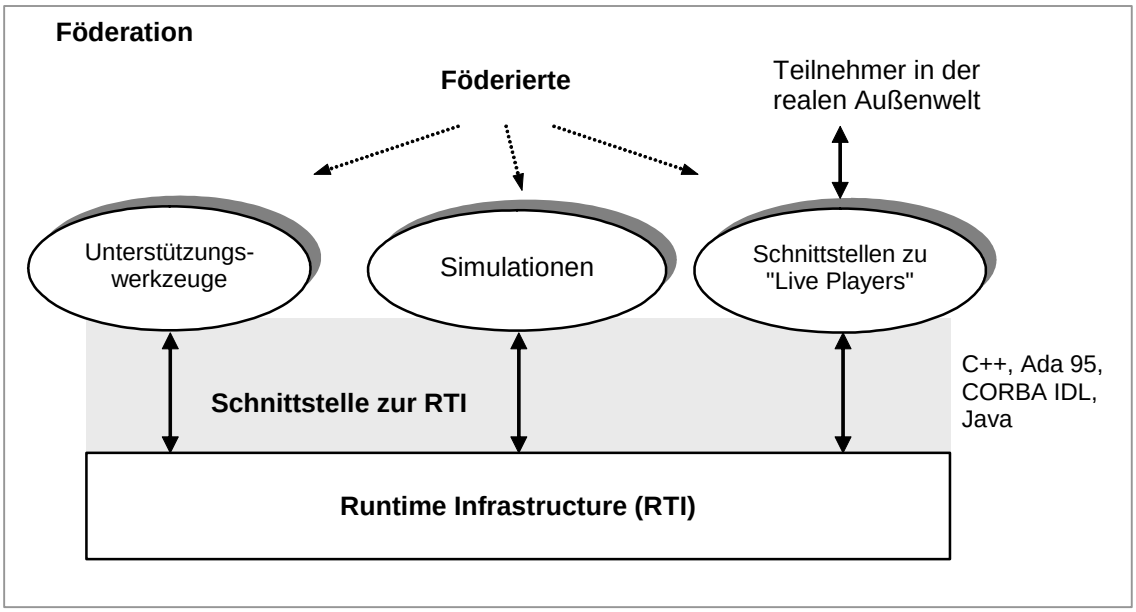


Bild 2.3.3 Funktionale Komponenten der High Level Architecture (HLA)

Die Dienste der RTI

Die RTI stellt die Implementierung der HLA-Schnittstellenspezifikation dar und stellt den an einer Föderation beteiligten Simulationen verschiedene Dienste zur Verfügung. Diese sind in insgesamt sechs funktionale Gruppen unterteilt:

1. Federation Management

   Aufgabe dieser Dienstgruppe ist die Koordination von Aktivitäten, die das Föderationsmanagement betreffen. Dazu werden Dienste zum Erzeugen und Beenden von Föderationen zur Verfügung gestellt. Weitere Dienste gestatten den Föderierten den dynamischen Beitritt bzw. Austritt aus einer Föderation. Darüber hinaus werden Steuermöglichkeiten für das Anhalten, Speichern und Wiederherstellen von Föderationszuständen angeboten.

2. Declaration Management

   Die HLA zeichnet sich durch einen impliziten Datenaustausch aus. Das bedeutet, dass Föderierte Daten des gemeinsamen Objektmodells nicht explizit durch Angabe eines Föderiertennamens an andere Simulationsteilnehmer versenden, sondern implizit über

die RTI der Föderation zur Verfügung stellen. Die Dienste des „Declaration Managements“ ermöglichen es, den Föderierten der RTI mitzuteilen, dass sie beabsichtigen Daten zu publizieren (publish) bzw. zu abonnieren (subscribe).

3. Object Management

   „Object Management“ Dienste bewerkstelligen den tatsächlichen Daten- und Nachrichtenaustausch. Ein Föderierter verwendet Dienste dieser Art, um im FOM definierte Interaktionen zu senden bzw. zu empfangen. Des Weiteren ermöglichen diese Dienste die Erzeugung von Objektinstanzen der im FOM spezifizierten Objektklassen sowie die Modifikation der dazugehörenden Objektattribute.

4. Ownership Management

   Föderierte müssen eine entsprechende Berechtigung besitzen, um Attribute von Objektinstanzen der im FOM spezifizierten Objektklassen verändern zu können. Die Dienste des „Ownership Managements“ ermöglichen es, solche Berechtigungen an Föderierte zu vergeben bzw. bereits vergebene Berechtigungen an andere Föderierte weiterzugeben.

5. Time Management

   Das Zeitmanagement stellt ein zentrales Problem verteilter Simulationen dar. Die vom Zeitmanagement zur Verfügung gestellten Dienste ermöglichen
   - den Föderierten die Fortschaltung ihrer logischen Zeit in Koordination mit anderen Föderierten,
   - die kontrollierte Vergabe von Ereignissen mit Zeitstempel, wobei bei konservativer Synchronisation ausgeschlossen wird, dass Föderierte ein Ereignis erhalten, dessen Zeitstempel in der logischen Vergangenheit des Föderierten liegt.

   Die RTI erlaubt den Föderierten, sich in verschiedenen Abstufungen am Zeitmanagement zu beteiligen. Ein Föderierter kann zeitreguliert und/oder zeitregulierend sein. Zeitreguliert bedeutet, dass das Fortschalten der logischen Zeit des Föderierten von den zeitregulierenden Föderierten der Föderation kontrolliert wird.

6. Data Distribution Management

   Die Dienste des „Data Distribution Management“ kontrollieren die Erzeuger-Konsumenten-Beziehungen zwischen Föderierten und ermöglichen es, solche Beziehungen weiter zu verfeinern. Der „Routing Spaces“-Ansatz gewährleistet eine effiziente Datenübermittlung, wobei im Gegensatz zum „Broadcast“-Ansatz von DIS (Distributed Interactive Simulation) lediglich dort, wo sich Angebot und Nachfrage von Objektinformationen überschneiden, Kommunikation stattfindet.

Der HLA-Regelsatz

Die HLA-Regeln legen fest, wie Föderationen und die daran beteiligten Föderierten über die RTI kommunizieren. Das Regelwerk ist in zwei Blöcke unterteilt: die ersten 5 Regeln betreffen die Föderationen, die übrigen 5 Regeln betreffen die Föderierten.

1. Föderationen müssen ein FOM besitzen, das gemäß dem OMT aufgebaut ist.
2. Die Werte sämtlicher simulationsbezogener Objektinstanzen müssen in den Föderierten gehalten werden und nicht in der RTI.
3. Der Austausch von Daten aus dem FOM zwischen den Föderierten darf nur über die RTI erfolgen.
4. Föderierte müssen sich bei Interaktion mit der RTI strikt an die HLA-Schnittstellenspezifikation halten.
5. Ein Attribut einer Objektinstanz darf von höchstens einem Föderierten verwaltet werden.
6. Föderierte müssen ein SOM besitzen, das gemäß dem OMT aufgebaut ist.
7. Föderierte müssen in der Lage sein, gemäß ihrer SOM-Spezifikation Objektattribute zu aktualisieren und/oder zu reflektieren sowie Interaktionen zu versenden und/oder zu empfangen.
8. Föderierte müssen in der Lage sein, gemäß ihrer SOM-Spezifikation die Verwaltung von Objektattributen dynamisch während der Föderationsausführung abzugeben und/oder zu übernehmen.
9. Föderierte müssen in der Lage sein, gemäß ihrer SOM-Spezifikation die Bedingungen zu verändern, unter denen sie Aktualisierungen von Objektattributen vornehmen.
10. Föderierte müssen in der Lage sein, ihre lokale Simulationszeit so zu verwalten, dass ein koordinierter Datenaustausch mit anderen Föderierten gemäß den Diensten des „Time Management“ erfolgen kann.

Die HLA-Objektmodellschablone

Die HLA nutzt eine objektorientierte Sichtweise, um die in einer Föderation abgebildeten Systemstrukturen und Abläufe zu beschreiben. Abweichend von der klassischen objektorientierten Systematik besitzen Objekte zwar Attribute als statische Elemente, dynamische Elemente (üblicherweise Methoden genannt) werden allerdings als eigenständige so genannte „Interactions“ dargestellt, welche ebenfalls über eine Hierarchie und Eigenschaften (Parameter genannt) verfügen und nicht-persistente, zu einem Zeitpunkt auftretende Ereignisse darstellen. Jeder Föderierte hat ein SOM aufzuweisen, das gemäß dem OMT zu dokumentieren ist. Das OMT enthält Vorgaben sowohl für das FOM als auch für das SOM. Im FOM wird festgelegt, welche Föderierten an dem Simulationsverbund teilnehmen und welche Informationen innerhalb des Verbundes ausgetauscht werden. Im SOM wird festgelegt, welche Methoden und Attribute der jeweilige Föderierte nach außen hin veröffentlicht.

Eine ausführliche Beschreibung der HLA ist in /KuWD-00/ zu finden. Informationen über aktuelle Entwicklungen werden vom US-Verteidigungsministerium über das WWW unter /HiLA-03/ zur Verfügung gestellt.

2.3.2.5 Resumé zur verteilten Komponenten-orientierten Modellierung und Simulation

Eine verteilte Komponenten-orientierte Modellierung und Simulation ermöglicht insgesamt eine Vereinfachung der Entwicklung, Wartung, Änderung, Erweiterung und Verteilung von Software. Des Weiteren ist in diesem Zusammenhang auch eine Integration von monolithischen (Alt-)Werkzeugen /StKl-98/ möglich. Dies soll dazu führen, dass in Zukunft zeitaufwändige Re-Implementierungen vermieden werden können. Stattdessen soll eine offene, verteilte Modellierungsumgebung zur Verfügung stehen, in der Entwickler ihren Fokus leichter auf neuartige Anwendungen richten können.

Bild 2.3.4 zeigt zwei unterschiedliche Aspekte der Verteilung im Bereich der Komponenten-orientierten Modellierung und Simulation auf. Dabei handelt es sich zum einen um die Verteilung von Modellen und zum anderen um die Verteilung der Funktionalität von Modellierungswerkzeugen. Zum Aufbau und Ablauf verteilter Simulationsmodelle bzw. Simulationsanwendungen eignet sich die im letzten Abschnitt beschriebene High Level Architecture, die eine entsprechende Infrastruktur zur Verfügung stellt. Im Gegensatz dazu sind Modellierungswerkzeuge dadurch gekennzeichnet, dass sie eine Vielzahl an unterschiedlichen Funktionalitäten zur Erstellung und Pflege von Modellen, aber auch zu deren Analyse und Optimierung anbieten. Eine Verteilung dieser Funktionalitäten würde ebenfalls eine entsprechende Infrastruktur benötigen, auf der der Fokus im Rahmen dieser Arbeit liegt.

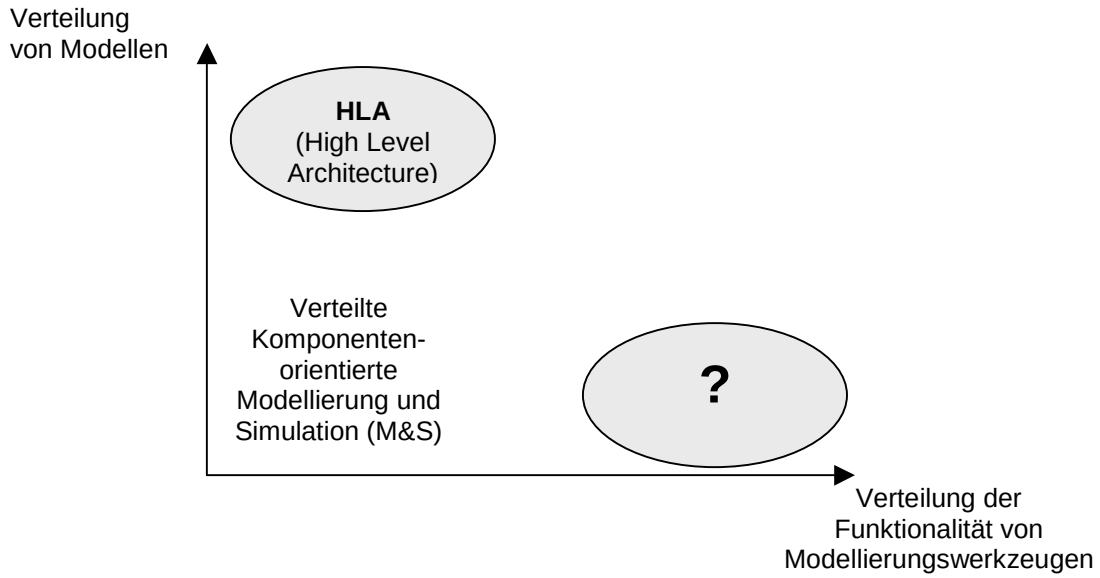


Bild 2.3.4 Unterschiedliche Aspekte der Verteilung

2.4 Analyse klassischer Modellierungswerkzeuge

Heutzutage stehen zahlreiche computergestützte Werkzeuge bereit, um die Entwicklung und Analyse verschiedener Arten von Modellen zu unterstützen. Dieser Abschnitt geht zunächst der Frage nach, wie klassische Modellierungswerkzeuge aufgebaut sind. Anschließend wird auf deren Nachteile beim gegenwärtigen Stand der Technik eingegangen. Der Abschnitt endet mit einer Darstellung unterschiedlicher Integrationsansätze.

2.4.1 Aufbau

Bild 2.4.1 zeigt die Grundstruktur eines klassischen Modellierungswerkzeugs, das im Wesentlichen aus den folgenden Teilen besteht /Saue-99/:

- *Editor/Übersetzer*: Die Dateneingabe erfolgt mit Hilfe eines Text- oder Graphikeditors. Eine besondere Form stellen Übersetzer dar, die Quelltexte einer Simulationssprache in eine interne, ausführbare Darstellung überführen.
- *Simulationsmodul*: Das Simulationsmodul enthält Algorithmen zum Durchrechnen bzw. Ausführen der Modelle, wie Simulationssteuerung, numerische Verfahren zum Lösen von Gleichungen, Vektor-Transformationen usw. Zur Simulation müssen das Modell und die Daten aus der Datenbasis ausgelesen und die Ergebnisse wieder dorthin zurückgeschrieben werden. Die Schnittstelle zur Benutzeroberfläche umfasst die interaktive Eingabe von Parametern und exogenen Größen sowie Echtzeitanimationen.
- *Analysemodul*: Das Analysemodul bereitet die Simulationsergebnisse auf und bietet eventuell neben verschiedenen statistischen Auswertungen auch grafische und tabellarische Darstellungen an. An dieser Stelle können auch Schnittstellen zu anderen Systemen, insbesondere Statistik-Paketen, vorhanden sein.
- *Datenbasis*: Gemeinsame Basis der drei Hauptfunktionen stellt eine zentrale Datenbasis dar, die zur Laufzeit im Hauptspeicher vorhanden sein kann, aber zur Ablage von Daten, Modellen und Ergebnissen in der Regel eine dauerhafte Form der Speicherung benötigt. Dabei kann es sich um einfache Dateien oder vollständige, meist externe Datenbanksysteme handeln.

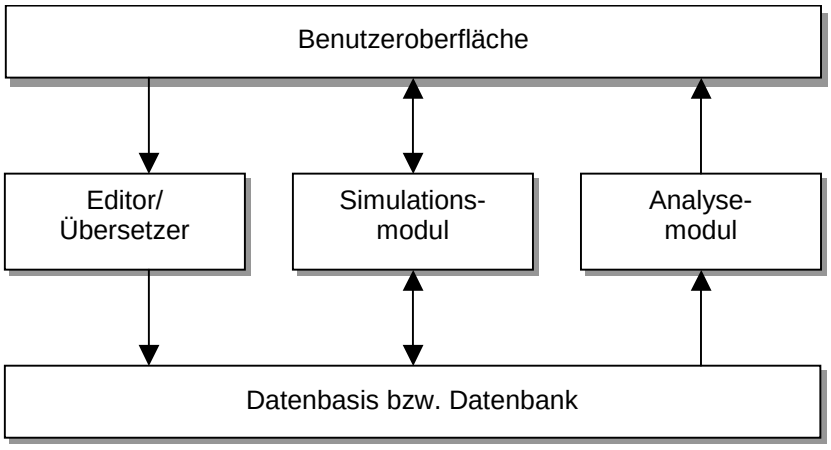


Bild 2.4.1 Grundstruktur eines Modellierungswerkzeugs

2.4.2 Nachteile beim gegenwärtigen Stand der Technik

Heutzutage existiert eine Vielfalt an Modellierungskonzepten, Beschreibungsmitteln und Analysemöglichkeiten für die unterschiedlichsten Problemdomänen. Darüber hinaus findet man eine heterogene Werkzeuglandschaft vor, die sich am Beispiel von Petrinetz-basierten Modellierungswerkzeugen besonders gut verdeutlichen lässt. Zu dieser Modellierungstechnik gibt es eine Vielzahl an Werkzeugen, die sich hinsichtlich ihres Funktionsumfanges stark

voneinander unterscheiden können. Eine Übersicht ist im WWW unter /PNTS-03/ zu finden. Vergleicht man diese Werkzeuge miteinander, so stellt sich eine große Implementierungsvielfalt hinsichtlich graphischer Modelleditoren, Markenspiel-Animationen sowie Struktur- und Verhaltensanalysen heraus. Dieses Spektrum an Werkzeugen bietet dem Modellierer zwar vielfältige Möglichkeiten zum Experimentieren mit verschiedenen Petrinetzarten, jedoch fallen bei näherer Betrachtung dieser Modellierungswerkzeuge auch einige Nachteile auf, die sich wie folgt zusammenfassen lassen:

- *Monolithisches Softwaredesign:* Viele Modellierungswerkzeuge weisen ein monolithisches Softwaredesign auf. Für diese Systeme ist typisch, dass sie sehr komplex sind, einen langen evolutionären Entwicklungsprozess hinter sich haben, plattformabhängig implementiert und teilweise nur unzureichend dokumentiert wurden. Dementsprechend sind solche Systeme heute schwierig zu warten, zu modifizieren und zu erweitern.
- *Mangel an Interoperabilität*: Uneinheitliche Schnittstellen und unzureichende Dokumentation erschweren den Austausch bzw. die Kombination unterschiedlicher Werkzeugteile. In solchen Fällen ist es beispielsweise nicht möglich, ein Modell mit einem bestimmten Modelleditor zu editieren und für die Auswertung ein Analysemodul eines anderen Werkzeugs zu verwenden.
- *Vielfalt unterschiedlicher Modellbeschreibungsformate*: Ein weiteres Merkmal klassischer Modellierungswerkzeuge besteht darin, dass jedes Werkzeug sein eigenes Modellbeschreibungsformat verwendet. Das heißt, die Art und Weise, wie Modelle und Informationen in Dateien abgespeichert werden, ist herstellerabhängig. Dies verhindert die gleichzeitige Nutzung des Funktionsumfangs mehrerer Werkzeuge innerhalb einer Modellstudie, weil ein Modellaustausch zwischen den beteiligten Werkzeugen nicht möglich ist.
- *Mangelnde Präsentationsmöglichkeiten:* Die Möglichkeiten der Präsentation von Modellen und Analyseergebnissen variieren von Werkzeug zu Werkzeug. Manche Werkzeuge verzichten sogar ganz auf eine graphische, anschauliche Präsentation. Dies erschwert jedoch das Modellverständnis sowie die Interpretation von Analyseergebnissen. Ein weiterer Nachteil besteht darin, dass die Art der Präsentation nicht flexibel ist, das heißt sie lässt sich nicht an die unterschiedlichen Sichten verschiedener Benutzergruppen anpassen.
- *Mangel an Web-Fähigkeit:* Die meisten klassischen Modellierungswerkzeuge sind nicht web-fähig. Dies verhindert einen orts- und plattformunabhängigen Zugriff auf Modelle, Daten sowie Simulations- und Analysefunktionen eines Modellierungswerkzeugs. Des Weiteren ist eine internetbasierte Interaktion mit anderen Benutzern in einem Werkzeugverbund nicht möglich.

Zusammenfassend lässt sich festhalten, dass heute kein universelles Modellierungswerkzeug existiert, das alle Arten von Modellierungskonzepten und Analysemethoden unterstützt. Auch zu einer bestimmten Modellierungstechnik gibt es in der Regel kein Werkzeug, das alle anderen in allen Belangen übertrifft. Die vollständige Neuimplementierung eines solchen Werkzeugs erscheint nicht sinnvoll, da hierfür ein immenser Programmieraufwand notwendig wäre. Vielmehr sollte aus ökonomischen Gründen, aber auch zur Beherrschung und Pflege

der entstandenen Modelle sowie zu Wartungs- und Vernetzungszwecken eine Konvergenz von Methoden, Beschreibungsmitteln und bereits existierenden Werkzeugen angestrebt werden, die neue Ansätze im Bereich der Integration erfordern.

2.4.3 Integrationsansätze

Die Ursachen für einen Integrationsbedarf sind vor allem darin zu sehen, dass bei der Planung, Entwicklung und Einführung komplexer Systeme häufig unterschiedliche Modellierungssprachen und Werkzeuge verwendet werden. Heute existieren bereits zahlreiche Ansätze, die versuchen, unterschiedliche Modellkonzepte, Beschreibungsmittel und Werkzeuge zu integrieren. Um eine Einordnung und Abgrenzung dieser Arbeit vornehmen zu können, wird zunächst eine Klassifikation von Integrationskonzepten nach verschiedenen Kriterien vorgenommen. Im Anschluss daran werden verschiedene Integrationsansätze für Modelle und Werkzeuge vorgestellt.

2.4.3.1 Klassifikation von Integrationskonzepten

In /NaWe-03/ findet eine Unterscheidung von Integrationskonzepten nach folgenden Kriterien statt:

- Integrationsebenen
- Art des konzeptionellen Ansatzes

Aus der Sichtweise einer konzeptionellen und einer technischen Integration gehen verschiedene *Integrationsebenen* hervor. Diese Ebenen lassen sich wie folgt unterscheiden:

- Begriffe
- Modellkonzept
- Beschreibungsmittel, Dokumentation
- Methoden
- Werkzeug
- Realisierung, Produkt
- Betrieb, Migration

Eine konzeptionelle Integration resultiert in einem einheitlichen Modellkonzept, während eine technische Integration bis zur Migration der Einrichtungen im Betrieb führt. Des Weiteren kann eine Integration innerhalb einer Ebene (*Intra-Ebenen-Integration*) sowie ebenenübergreifend (*Inter-Ebenen-Integration*) praktiziert werden. Bei der ersten Art kann es sich beispielsweise um die Integration verschiedener Beschreibungsmittel handeln, bei der zweiten Art um die Integration verschiedener Beschreibungsmittel in einem Werkzeug. Innerhalb einer Integrationsebene ist häufig noch das *Integrationsniveau* zu berücksichtigen, z.B. bei Beschreibungsmitteln, wo eine syntaktische Integration nicht immer mit einer semantischen Integration korrespondiert.

Unabhängig von der Intra- bzw. Inter-Ebenen-Integration werden Integrationskonzepte weiterhin hinsichtlich der *Art ihres konzeptionellen Ansatzes* (siehe Bild 2.4.2) unterschieden. Bei einer einfachen Zusammenstellung mehrerer, aber gleichartiger isolierter Dinge, z.B. Werkzeuge, spricht man von *isolierter* Integration. Eine nächste Integrationsart ist die *additive* Verknüpfung von Dingen, die notwendigerweise über semantische Fähigkeiten verfügen muss, z.B. bei einer Verschaltung elektrischer Bauelemente oder bei einer konsekutiven Verbindung von Modellen einzelner Entwicklungsphasen. Von *kooperativer* Integration wird gesprochen, wenn bestimmte Eigenschaften erst durch Verknüpfung von mehreren Dingen erreicht werden, die singulär nicht erbracht wurden. Beispiele hierfür auf modell-konzeptioneller Ebene sind so genannte hybride Systeme, z.B. diskret-kontinuierliche oder deterministisch-stochastische Systeme. Eine weitere Integrationsart heißt *integrativ*, wenn komplementäre Dinge zu einem neuen Ding verschmolzen werden, ohne ihren originären Charakter aufzugeben. Spezielle Formen der integrativen Integration sind die *Transformation* und die *Abstraktion*. Bei der Transformation dominiert eine Form der ursprünglich verschiedenen Formen, in welche die komplementäre Form überführt wird, während bei der Abstraktion verschiedene Dinge in einem übergreifenden, umfassenden Neuen aufgehen.

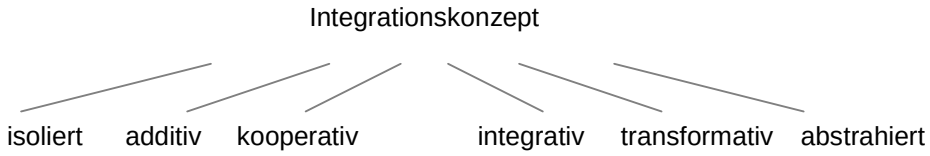


Bild 2.4.2 Klassifikation von Integrationskonzepten nach der Art des konzeptionellen Ansatzes

Eine umfassendere Klassifikation erhält man, wenn man die oben beschriebenen Merkmale orthogonal zueinander anordnet. Damit lassen sich Integrationskonzepte in insgesamt 42 verschiedene Klassen einteilen.

2.4.3.2 Integrationsansätze für Modelle und Werkzeuge

In /Schn-03/ wird zwischen drei verschiedenen Integrationsansätzen für Modelle und Werkzeuge unterschieden (siehe Bild 2.4.3):

- *Modell-zentrierter Ansatz*: Beim Modell-zentrierten Ansatz werden verschiedene Beschreibungsmittel in ein gemeinsames Modellkonzept transformiert und auf dieser Ebene integriert. Dadurch können aufgrund der gleichen Modellbasis unterschiedliche Werkzeuge verschiedener Funktionalität genutzt werden.
- *Werkzeug-zentrierter Ansatz*: Der Werkzeug-zentrierte Ansatz integriert verschiedene Modellkonzepte in Form von Werkzeugen.
- *Kooperativer Werkzeugverbund*: Der dritte Ansatz koppelt Werkzeuge ohne größere Eingriffe miteinander zu einem kooperativen Werkzeugverbund.

Im Rahmen dieser Arbeit liegt der Focus auf Konzepten zur Integration von heterogenen Modellierungswerkzeugen. Darunter wird in dieser Arbeit eine Kombination aus den oben beschriebenen Integrationsansätzen verstanden, d.h.:

> *Heterogene Modellierungswerkzeuge zeichnen sich dadurch aus, dass ihnen unterschiedliche Modellierungskonzepte zu Grunde liegen können bzw. dass sie bei gleichem Modellierungskonzept auf unterschiedliche Werkzeugfunktionalitäten zurückgreifen können oder dass sie Bestandteil innerhalb eines kooperativen Werkzeugverbundes sind.*

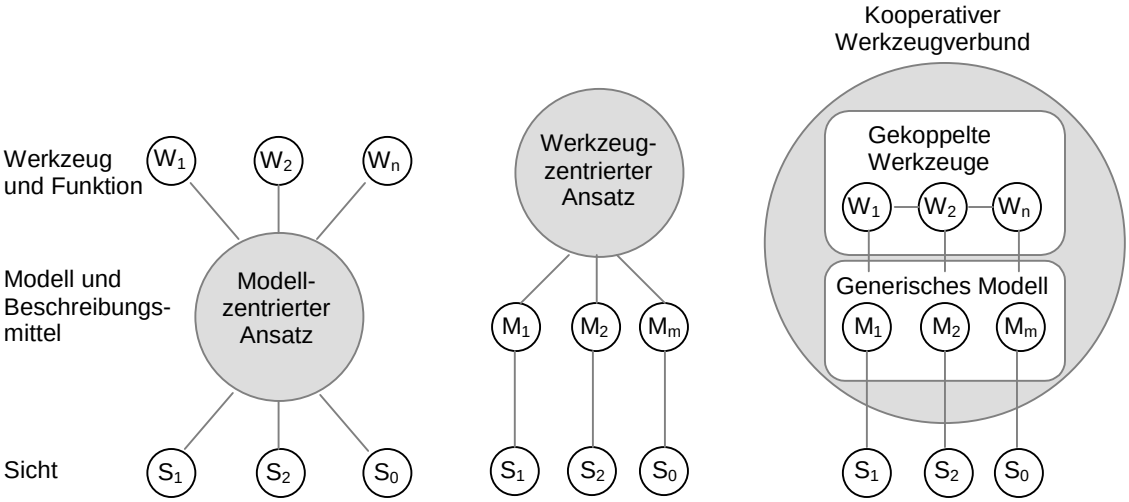


Bild 2.4.3 Integrationsansätze für Modelle und Werkzeuge

Auf die einzelnen Ziele der vorliegenden Arbeit wird im nächsten Abschnitt näher eingegangen.

2.5 Ziele der vorliegenden Arbeit

Um die im letzten Abschnitt angeführten Probleme zu beheben, bestand das Ziel dieser Arbeit darin, ein neues Konzept zur Integration von Modellierungswerkzeugen zu entwerfen, dem ein komponentenbasiertes Architekturkonzept zu Grunde liegt. Aus den bisherigen Betrachtungen ergeben sich folgende Anforderungen an das zu entwickelnde System:

- *Offenheit:* Ein wesentlicher Aspekt ist die Offenheit des Systems, das heißt, eine Erweiterung um neue Werkzeuge soll ohne Kenntnisse des internen Aufbaus und ohne Änderung des Quellcodes möglich sein. Bei offenen Systemen ist es möglich, Softwarewerkzeuge unter Nutzung definierter Schnittstellen je nach Erfordernis hinzuzufügen bzw. zu entfernen und Werkzeuge innerhalb der Umgebung für jedes spezifische Anwendungsgebiet zu modifizieren.
- *Verwendung von Standards*: Um eine möglichst breite Akzeptanz des zu entwickelnden Systems zu erzielen, soll bei dessen Realisierung möglichst auf Standards zurückgegriffen werden. Um beispielsweise den Datenaustausch zwischen verschiedenen Werkzeugen

zu erleichtern, bietet es sich an, die Auszeichnungssprache XML zur Datenmodellierung einzusetzen.

- *Berücksichtigung verschiedener Benutzersichten*: Um auf die individuellen Bedürfnisse verschiedener Benutzergruppen eingehen zu können, soll eine unterschiedliche Aufbereitung und Präsentation von Modellierungsdaten möglich sein. Dies umfasst neben textbasierten auch graphische, anschauliche Visualisierungsformen. In diesem Zusammenhang sind auch Ausgabemöglichkeiten auf unterschiedlichen Endgeräten zu untersuchen.
- *Integration von Altsystemen*: Da in der Regel eine Reimplementierung von Altsystemen aus ökonomischen Gründen nicht in Frage kommt, soll eine Integration dieser Systeme durch entsprechende Integrationsschnittstellen möglich sein. Diese sollen einerseits möglichst universell gehalten werden, um unterschiedliche Modellierungswerkzeuge einbinden zu können, andererseits sollen diese Schnittstellen einfach aufgebaut sein, damit der Integrationsaufwand möglichst gering gehalten werden kann.
- *Verwendung von Web-Technologien*: Eine wichtige technologische Randbedingung im Rahmen der Integration von Modellierungswerkzeugen stellt der Einsatz von Web-Technologien dar, da Modellierungswerkzeuge in der Regel über das Intranet und/oder das Internet verfügbar sein sollen. Des Weiteren wird bei der Entwicklung verschiedener Funktionalitäten von Modellierungswerkzeugen immer häufiger auf Web-Technologien zurückgegriffen. Daraus resultiert, dass die Daten bei der Integration so zur Verfügung gestellt werden müssen, dass sie direkt oder nach Weiterverarbeitung im Web-Browser darstellbar sind.

Weitere Ziele dieser Arbeit bestanden zum einen darin, die wichtigsten Konzepte durch entsprechende Prototypimplementierungen zu realisieren und zum anderen den Einsatz und die Bewertung der Konzepte anhand eines Beispielszenarios durchzuführen.

3 Eine Komponentenarchitektur zur Integration heterogener Modellierungswerkzeuge

In diesem Kapitel wird ausgehend von den im vorangegangenen Kapitel diskutierten Problemen klassischer Modellierungswerkzeuge eine Architektur vorgeschlagen, der ein Komponenten-basierter Ansatz zu Grunde liegt. Dazu werden zunächst in Abschnitt 3.1 die Entwurfsentscheidungen für die interne Organisation der Architektur vorgestellt. Anschließend erfolgt in Abschnitt 3.2 ein Überblick über die einzelnen Bestandteile dieser Architektur. In Abschnitt 3.3 wird auf die Interaktionen der heterogenen Komponenten eingegangen und in Abschnitt 3.4 werden verschiedene Arten von Schnittstellen beschrieben. Der Abschnitt 3.5 behandelt ein Integrationskonzept auf Basis standardisierter Dokumente. Abschließend wird in Abschnitt 3.6 eine Möglichkeit der technischen Realisierung aufgezeigt.

3.1 Entwurfsentscheidungen

Herkömmliche Modellierungswerkzeuge haben den Nachteil, dass ihr Aufbau monolithisch ist und sie nur bestimmte Funktionalitäten anbieten. Dieses Angebot an Funktionalitäten kann meistens nicht ergänzt werden, da entsprechende Schnittstellen dafür nicht vorgesehen wurden. Die grundlegende Idee des in dieser Arbeit entwickelten Ansatzes besteht darin, die Funktionalität von Modellierungswerkzeugen in unabhängig voneinander arbeitende Softwarekomponenten zu zerlegen. Aus diesen Komponenten können dann, je nach Anforderungen, entsprechende Modellierungsumgebungen individuell zusammengestellt werden. Um ein Zusammenspiel der Komponenten zu erreichen, müssen ihre Funktionalitäten und Schnittstellen aufeinander abgestimmt sein. Zur Integration dieser heterogenen Komponenten soll die in diesem Kapitel vorgestellte Architektur dienen, welche die Entwickler der Komponenten in die Lage versetzt, neue oder bessere Komponenten so zu entwickeln, dass sie leicht integriert werden können. Im Gegensatz zu existierenden Ansätzen orientiert sich damit die architektonische Struktur nicht am klassischen monolithischen Aufbau, sondern an der Konzeption und Entwicklung von Komponenten, die Einzelfunktionalitäten mit weitgehender Kontextunabhängigkeit darstellen. Insgesamt hat der Komponenten-basierte Software-Entwurf die folgenden Vorteile:

- Er ermöglicht eine flexible Verteilung der involvierten Komponenten innerhalb eines Rechnernetzes.
- Er erlaubt Benutzerzugang durch herkömmliche Applikations-Klienten oder durch Java-basierte Web-Klienten.
- Er ermöglicht eine Integration von bereits vorhandenen monolithischen Werkzeugen durch Transformation eines proprietären Modellbeschreibungsformats in ein standardi-

siertes XML-basiertes Beschreibungsformat und/oder durch geeignete Komponenten-„Adapter“.

- Er vereinfacht die Modifikation und Erweiterung von Werkzeugen.
- Er stellt eine gute Ausgangsbasis für agentenbasierte Systeme dar.

Der Vorteil dieser Vorgehensweise besteht darin, dass durch den einfachen Austausch einzelner Komponenten veränderte Anforderungen schneller umgesetzt und neue Merkmale oder verbesserte Funktionen schneller eingesetzt werden können. Dies führt insgesamt zu einer Beschleunigung der Entwicklung und zu einer einfacheren Wartung der Einzelfunktionalitäten. Neben technischen Vorteilen eröffnet eine Komponentenorientierung auch zahlreiche ökonomische und organisatorische Vorteile wie beispielsweise Wiederverwendung von Software sowie eine klare Trennung von Verantwortlichkeiten.

3.2 Gesamtübersicht über die entwickelte Komponentenarchitektur

Die gemäß den Entwurfskonzepten aus Abschnitt 3.1 entwickelte Komponentenarchitektur für Modellierungswerkzeuge /SySS-02/ ist in Bild 3.2.1 dargestellt. Sie beschreibt die heterogenen Komponenten eines Modellierungswerkzeugs, ihre Aufgaben und ihr Zusammenwirken über eine verteilte Infrastruktur. Komponenten können auch externe Systeme darstellen. Dazu zählen beispielsweise Altsysteme, Dateiverwaltungs- bzw. Datenbanksysteme und verschiedene Arten von Endgeräten für unterschiedliche Benutzertypen. Im Folgenden wird näher auf die Funktionalität der einzelnen Komponenten eingegangen:

- *Modell-Editor(en):* Ein Modell-Editor ermöglicht dem Modellierer, neue Modelle zu editieren und existierende Modelle zu modifizieren. Die Art der Notation hängt dabei von der verwendeten Technik ab. Prinzipiell wird zwischen Text- und Graphik-Editoren unterschieden. Im Gegensatz zu einem Text-Editor bietet ein graphischer Modell-Editor dem Benutzer die Eingabe von graphischen Grundsymbolen und deren Platzierung auf dem Bildschirm an. Als Symbole werden ausschließlich diejenigen angeboten, die in der Notation der unterstützten Modellierungstechnik vorgesehen sind.
- *Komponenten zur Modellauswertung:* Modellauswertungskomponenten werden dazu verwendet, um Modelle zu analysieren und zu evaluieren. Dabei wird zwischen einer mathematischen Analyse von strukturellen Eigenschaften (Stellen-Invarianten, Transitions-Invarianten, usw.) und Leistungsevaluationen (stationäre Analyse, transiente Analyse) unterschieden. Leistungsevaluationen können entweder analytisch oder durch Simulation durchgeführt werden (vgl. Abschnitt 2.2).
- *Spezialkomponenten:* Spezialkomponenten ermöglichen gezieltes Experimentieren mit einem Modell wie zum Beispiel das Auffinden optimaler Parametereinstellungen, die Bestimmung empfindlicher Modellparameter oder das Durchführen einer Modellvalidation (vgl. Abschnitt 2.2).

Die Funktionen von Altwerkzeugen, Dateiverwaltungs- und Datenbanksystemen sowie von unterschiedlichen Endgeräten werden als bekannt vorausgesetzt. Der folgende Abschnitt beschreibt das Zusammenwirken der einzelnen Komponenten.

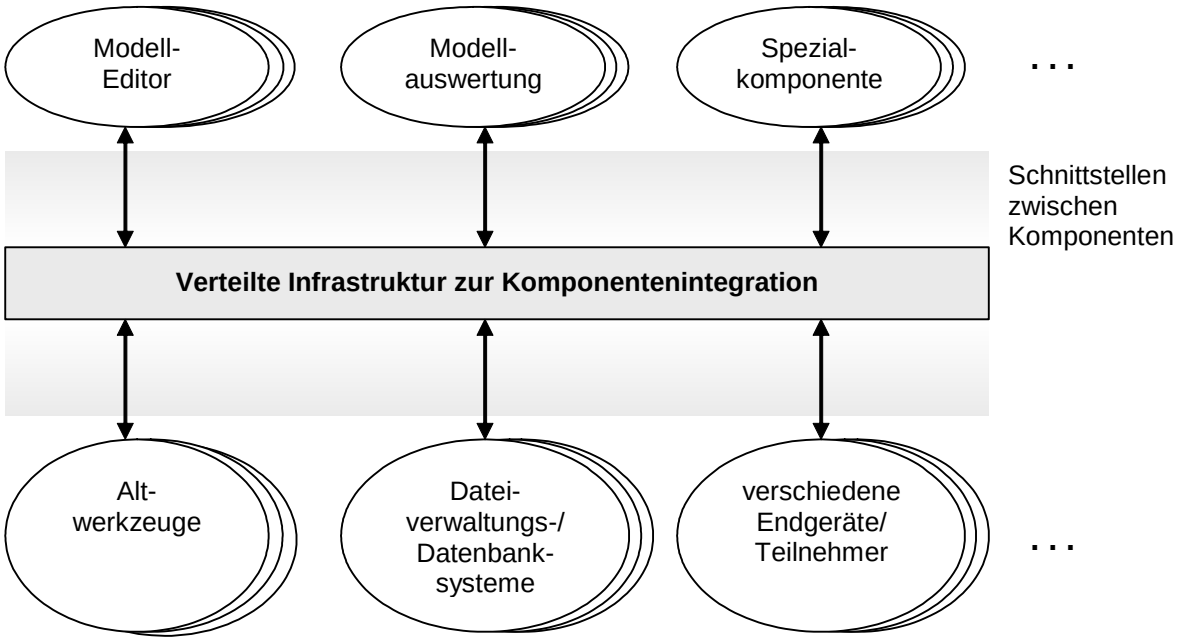


Bild 3.2.1 Gesamtübersicht über die Architektur zur Integration heterogener Komponenten

3.3 Interaktion der Komponenten

Jede der Komponenten zum Editieren, Auswerten und Experimentieren von Modellen besitzt, im Gegensatz zur monolithischen Architekturform, ihre eigene graphische Benutzungsschnittstelle (graphical user interface, GUI), wodurch die angebotene Funktionalität insgesamt überschaubarer wird. Die Ausgabe eines Modell-Editors ist eine Modellbeschreibung in einem spezifischen Modellbeschreibungsformat, welches durch die unterstützte Modellierungstechnik charakterisiert ist. Die graphische Benutzungsschnittstelle der Modellauswertungskomponente soll dem Modellierer ermöglichen, Modelle zu parametrisieren sowie Modellauswertungen vorzunehmen. Im Anschluss daran sollen die Auswertungsergebnisse in anschaulicher Form präsentiert werden. Modellauswertungskomponenten benötigen als Eingabe eine Modellbeschreibung und erzeugen als Ergebnis ein Dokument mit Auswertungsergebnissen.

Um empirische Modelluntersuchungen zu automatisieren, bei denen wiederholt Modellauswertungsexperimente unter systematischer Variation der Modellparameter durchgeführt werden, ist eine *enge Kopplung* der dazu eingesetzten Spezialkomponenten mit Modellauswertungskomponenten erforderlich. Die graphische Benutzungsschnittstelle der Spezialkomponente soll dem Modellierer dazu die Möglichkeit bieten, die zu variierenden Modellparameter und deren Bereichsgrenzen festzulegen sowie eine geeignete Modellauswertungskomponente auszuwählen. Bei Spezialkomponenten, die auf rein analytischen Methoden basieren, genügt eine *lose Kopplung* auf Basis von Dokumentenaustausch mit Modelleditoren, da zur Modellanalyse ausschließlich auf die Modellbeschreibung zurückgegriffen wird. Daten, die das zu analysierende Modell betreffen, werden dokumentenbasiert über die Modellbeschreibung bzw. über die von der Modellauswertungskomponente erzeugte Ergebnisdatei zur Verfügung gestellt. Die Experimentierergebnisse werden, wie in Bild 3.3.1 dargestellt, ebenfalls in einem XML-basierten Beschreibungsformat abgelegt.

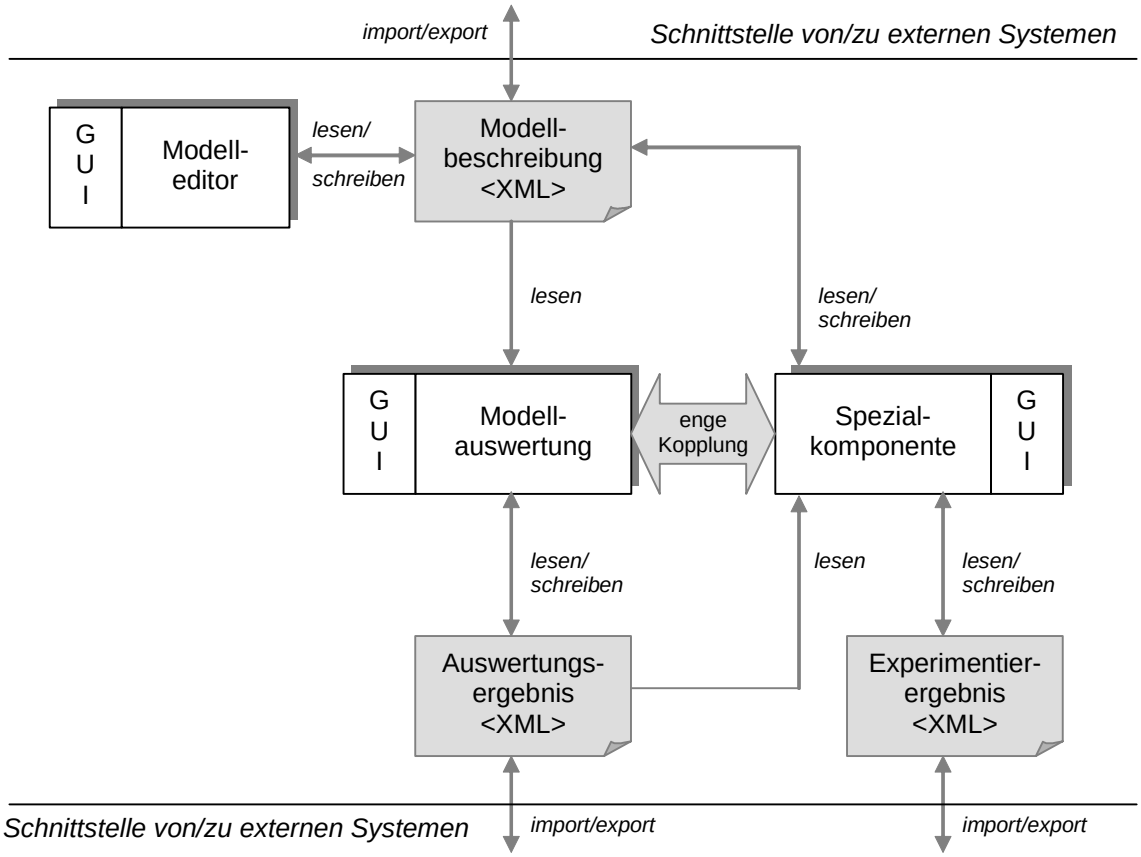


Bild 3.3.1 Interaktion der Komponenten

Insgesamt basiert die Kopplung der einzelnen Komponenten auf zwei Arten von Interaktionen: Austausch von XML-Dokumenten sowie direkter Aufruf von Methoden bzw. Prozeduren zur Modellauswertung. Der folgende Abschnitt geht näher auf die unterschiedlichen Schnittstellen zwischen den einzelnen Komponenten ein.

3.4 Schnittstellen zwischen Komponenten

Ausgehend von allgemeinen Konzepten zur Kopplung heterogener Komponenten werden im Anschluss daran spezifische Konzepte zur Integration externer Systeme aufgezeigt.

3.4.1 Kopplungsarten

Die Relationen zwischen den einzelnen Komponenten lassen sich durch zwei Arten von Abhängigkeiten unterscheiden: lose und enge Kopplung. Im Folgenden werden diese beiden Kopplungsarten näher erläutert.

3.4.1.1 Lose Kopplung

Die lose Kopplung von Komponenten basiert auf der Basis von Dokumenten, die über entsprechende Import- und Export-Schnittstellen zwischen den Komponenten ausgetauscht werden. Bild 3.4.1 veranschaulicht dazu folgende Konzepte:

- Punkt-zu-Punkt-Kopplungen proprietärer Austauschformate
- Zentrale Kopplung mit Hilfe eines generischen Austauschformats

Wie bereits in Abschnitt 2.2.1.3 beschrieben, etabliert sich heute zur Speicherung von Dokumenten und für den Datenaustausch zwischen heterogenen Anwendungen zunehmend die eXtensible Markup Language (XML). Damit lassen sich auch Modellierungsdaten, die zwischen den einzelnen Komponenten ausgetauscht werden, beschreiben. Im Einzelnen handelt es sich dabei um Modellbeschreibungen, Auswertungsergebnisse sowie Ergebnisse von Spezialkomponenten. Zur Kopplung von mehr als zwei verschiedenen Komponenten ist die Entwicklung eines generischen Austauschformats zweckmäßiger, vgl. Bild 3.4.1.

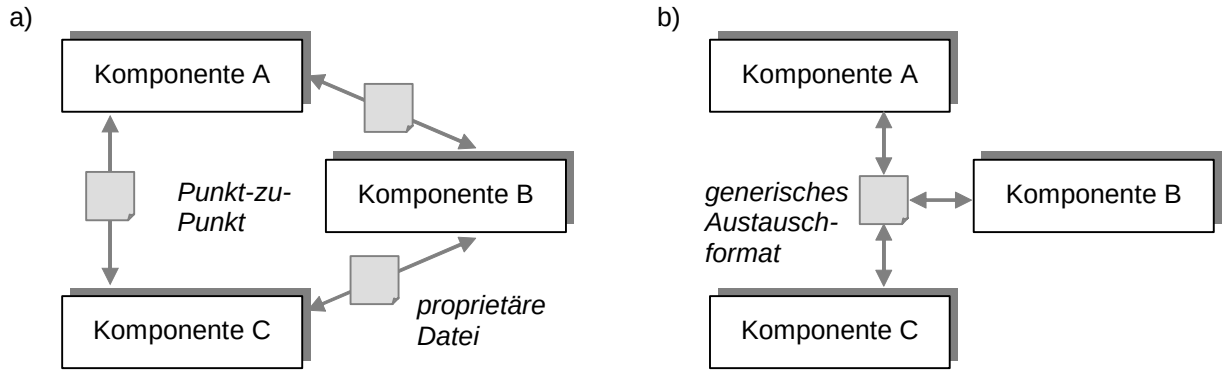


Bild 3.4.1 Konzepte zur losen Kopplung zwischen Komponenten

3.4.1.2 Enge Kopplung

Eine enge Kopplung von Komponenten basiert auf (entfernten) Prozedur- bzw. Methodenaufrufen. Prinzipiell existieren dazu zwei unterschiedliche Kopplungskonzepte:

- Punkt-zu-Punkt-Kopplungen über proprietäre Schnittstellen
- Zentrale Kopplung über geeignete Middleware-Standards

Bei mehr als zwei verschiedenen Komponenten ist eine zentrale Kopplung zweckmäßiger, vgl. Bild 3.4.2. Dazu können Kommunikationsinfrastrukturen wie beispielsweise CORBA (Common Object Request Broker Architecture) /OrHa-98/ angewandt werden. CORBA repräsentiert einen mächtigen „Middleware“-Standard, der folgende Vorteile bietet:

- Programmiersprachenunabhängige Schnittstelle, d.h. Schnittstellen zwischen Dienstnehmern (Klienten) und Dienstgebern (Server) werden in einer standardisierten Interface Definition Language (IDL) definiert.

- Mit Hilfe der IDL können Programmierer existierende Applikationen mittels so genannter „Adapter“ kapseln und diese als Objekte auf dem ORB (Object Request Broker) zur Verfügung stellen.
- CORBA bietet eine Vielzahl an verteilten Objektdiensten an, da verteilte Applikationen mehr Funktionalitäten als nur einfache Methodenaufrufe erfordern.

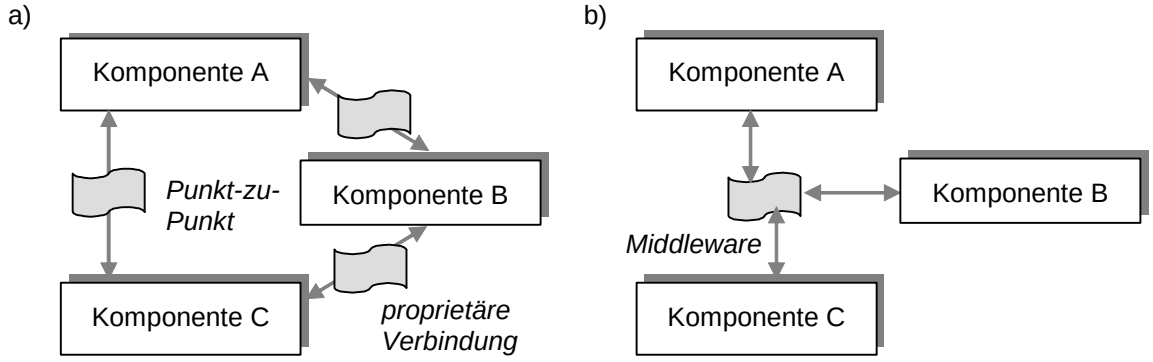


Bild 3.4.2 Konzepte zur engen Kopplung zwischen Komponenten

Eine weitere Möglichkeit, um proprietäre Schnittstellen zu vermeiden, bieten die Web-Service-Technologien /Deit-03/. Die Schnittstelle einer Applikation kann mit WSDL (Web Service Description Language) beschrieben werden und mit UDDI (Universal Description, Discovery and Integration of Web Services) publiziert und gefunden werden. Zum Austausch von XML-basierten Daten in einer dezentralen verteilten Umgebung eignet sich das leichtgewichtige SOAP-Protokoll (Simple Object Access Protocol). SOAP basiert auf XML und besteht aus den folgenden drei Teilen: (1) Einem Umschlag, der eine Umgebung zum Beschreiben des Nachrichteninhalts und der weiteren Bearbeitung definiert, (2) einem Satz an Kodierungsregeln, die anwendungsdefinierte Datentypen beschreiben und (3) Konventionen, um entfernte Methodenaufrufe und deren Rückgabe zu repräsentieren.

3.4.2 Schnittstellen zu externen Systemen

In diesem Abschnitt werden Möglichkeiten zur Integration von Altsystemen und von Endgeräten aufgezeigt. Um nicht nur eine Wiederverwendung von Altsystemen, sondern auch von Modellen bzw. Modellierungsdaten zu ermöglichen, wird im Anschluss daran auf die Anbindung an gängige Dateiverwaltungs- und Datenbanksysteme eingegangen.

3.4.2.1 Altsysteme

Die Integration von Altsystemen in die vorgeschlagene Komponentenarchitektur kann auf folgende zwei Arten erfolgen:

- (Lose) Integration auf Basis von Dokumentenaustausch
- (Enge) Integration auf Basis von (entfernten) Prozedur- bzw. Methodenaufrufen

Für die lose Integration wird ein bidirektionaler Konverter benötigt, der das vom Altsystem unterstützte proprietäre Modellbeschreibungsformat in das vom Komponenten-orientierten Werkzeug verwendete XML-basierte Modellbeschreibungsformat umwandelt und umgekehrt (siehe Bild 3.4.3 Fall a). Durch einen solchen Konverter ist es möglich, flexibel Modellbeschreibungen zwischen Modellierungswerkzeugen auszutauschen. Ein weiterer Vorteil dieser Integrationsart besteht darin, dass keinerlei Änderungen am Altsystem notwendig sind.

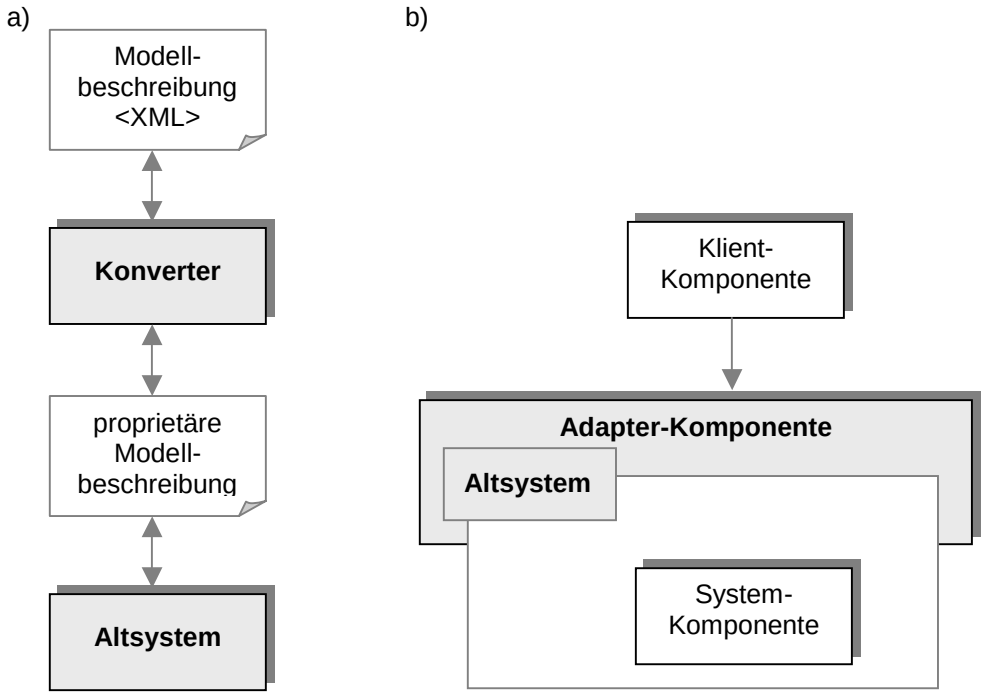


Bild 3.4.3 Konzepte zur Integration eines Altsystems

Zur engen Integration von Altsystemen auf Basis von (entfernten) Prozedur- bzw. Methodenaufrufen werden so genannte Adapter-Komponenten benötigt. Diese stellen die Schnittstelle zu einem Altsystem dar. Adapter-Komponenten hüllen ein Altsystem so ein, dass es von außen ansprechbar ist. Sie stellen spezielle Gateway-Komponenten dar, die auf der spezifischen Plattform des zu integrierenden Altsystems entwickelt werden /Andr-03/. Der Nutzer eines Altsystems greift dabei direkt auf die Adapter-Komponente zu, wobei das Altsystem aus Sicht des Klienten als Black-Box angesehen werden kann (siehe Bild 3.4.3 Fall b). Wie das Altsystem im Innern aufgebaut ist, spielt bei der Black-Box-Integration keine Rolle, solange es den Schnittstellen-Spezifikationen genügt.

3.4.2.2 Endgeräte

Eine Integration unterschiedlicher Endgeräte in die Komponentenarchitektur soll eine individuelle, auf den Benutzer abgestimmte, Repräsentation von Modellen und Modellierungsdaten unterstützen. Dazu wird eine entsprechende unidirektionale Konvertierung aus der allgemeinen XML-basierten Modellbeschreibung in eine proprietäre vom entsprechenden Endgerät unterstützte Formatbeschreibung benötigt (siehe Bild 3.4.4).

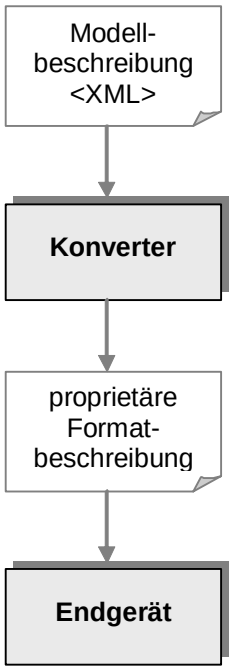


Bild 3.4.4 Integration eines Endgeräts

3.4.2.3 Dateiverwaltungs- und Datenbanksysteme

Zur Speicherung von Modellen und Modellierungsdaten werden folgende Möglichkeiten betrachtet:

- Dateiverwaltungssysteme
- Datenbanksysteme

Bei der ersten Variante werden alle Modelle und Modellierungsdaten in einem Archiv, dem Repository, gehalten und können von dort geholt und zurückgeschrieben werden (check out, check in). Ein Repository dient den Komponenten als gemeinsamer Datenspeicher, auf den die einzelnen Komponenten zugreifen können. Bild 3.4.5 veranschaulicht den „Shared Repository"-Interaktionsmodus. Mit Hilfe von Werkzeugen wie CVS (Concurrent Version System) können alle Versionen im Repository gehalten werden, da die Änderungshistorie von Dateien im Repository gemerkt und die jeweils aktuellste Version angeboten wird. CVS unterstützt dabei die Team-Kooperation, das heißt nur ein Team-Partner darf zu einem bestimmten Zeitpunkt eine Datei ändern oder Änderungen mehrerer Team-Partner werden zusammengeführt /Wiki-04/.

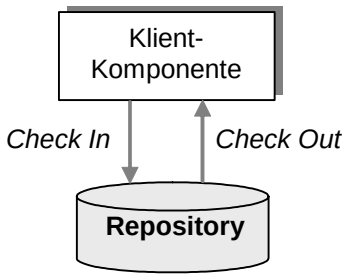
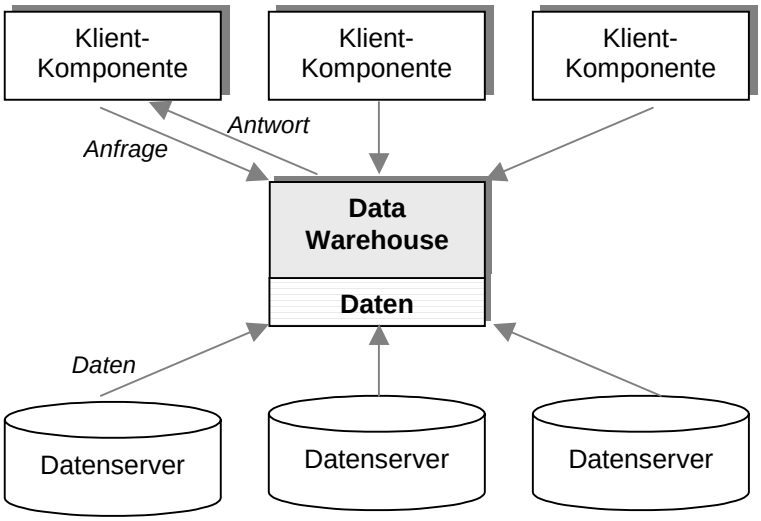


Bild 3.4.5 Shared Repository

Bei der zweiten Variante wird zwischen traditionellen Datenbanken und Web-Datenbanken unterschieden /BiAl-97/. Bei traditionellen Datenbanken kann der Klient nur einen Datenserver ansprechen. Web-Datenbanken beziehen dagegen ihre Daten von mehreren Datenservern und müssen diese in einem standardisierten Format (XML) liefern können. Im Gegensatz zu traditionellen Datenbanken befindet sich bei Web-Datenbanken eine neue Schicht zwischen Klienten und den Datenservern. Wie in Bild 3.4.6 dargestellt, lassen sich hierbei wiederum zwei verschiedene Varianten unterscheiden: Data Warehouse und Mediator Systems.

a) Data Warehouse



b) Mediator

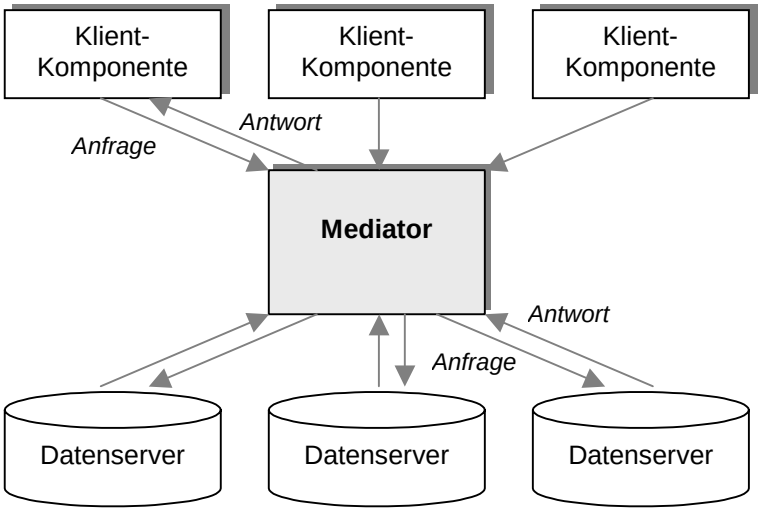


Bild 3.4.6 Data Warehouse vs. Mediator Systems

Data Warehouse speichert Daten von mehreren Datenservern und führt in regelmäßigen Abständen Updates durch. Anfragen werden direkt vom Data Warehouse beantwortet. Der Klient spricht eine Datenbank an, erhält aber Daten, die aus mehreren Datenservern stammen können. Im Gegensatz zum Data Warehouse speichert der Mediator keine Daten. Seine Auf-

gabe besteht darin, Anfragen an die jeweiligen Datenserver weiterzuleiten. Des Weiteren muss er einerseits die erhaltenen Daten eines Servers in einem gewünschten Format liefern (data conversion) und andererseits muss er ein Dokument erstellen können, das sich aus Daten mehrerer Server zusammensetzt (data integration).

3.5 Integration mittels standardisierter Dokumente

Zur Integration heterogener Komponenten spielt der Dokumenten-basierte Integrationsansatz eine wesentliche Rolle. Dieser Abschnitt stellt für die Spezifikation einheitlicher XML-basierter Modellaustauschformate einen im Rahmen dieser Arbeit entwickelten Metamodell-basierten Ansatz vor. Im Anschluss daran wird näher auf die in diesem Zusammenhang eingesetzten Standards eingegangen.

3.5.1 Metamodell-basierter Ansatz

Die folgenden Abschnitte beschreiben den Ausgangspunkt und die Vorgehensweise des Metamodell-basierten Ansatzes. Anschließend wird das Gesamtkonzept vorgestellt und am Beispiel stochastischer Petrinetzmodelle erläutert.

3.5.1.1 Ausgangspunkt und Vorgehensweise

Das Integrationskonzept auf Basis standardisierter Dokumente erfordert eine Spezifikation einheitlicher XML-basierter Modellaustauschformate. Grundvoraussetzung für einen Dokumenten-basierten Austausch zwischen Komponenten ist die Abstimmung zwischen den Dateninhalten und –strukturen sowie die Abstimmung der Terminologie und der Semantik für die benötigten Datenelemente. Eine syntaktische Heterogenität kann dabei durch strukturelle Differenzen hervorgerufen werden. Diese treten bei gleicher Modellierungssprache des Datenmodells durch unterschiedliche Modellierungen auf (z.B. Attribut statt Element in einem XML-Datenmodell). Semantische Unterschiede werden durch Homonyme und Synonyme bei der Bezeichnung von Daten verursacht (z.B. haben die Bezeichnungen Stelle und Platz dieselbe Bedeutung in einem Petrinetzmodell).

Um einen verlässlichen und effizienten Datenaustausch zwischen den einzelnen Komponenten ermöglichen zu können, ist ein einheitliches *Metamodell* von Vorteil. Dieses dient dazu, eine Modellierungssprache zu modellieren, um ihre Konzepte darzulegen, sie besser zu verstehen, einen Konsens über ihre Verwendung und Bedeutung herbeizuführen sowie ihre Elemente und Konstrukte präzise zu definieren und damit

- syntaktische Überprüfungen zu ermöglichen,
- Erweiterungen, Anpassungen zu vereinfachen und
- den Modellaustausch zu erleichtern.

Metamodelle stehen damit auf einer höheren Ebene als das zu beschreibende Modell und erlauben es, die Syntax und zum Teil auch die Semantik eines Modells formal zu beschreiben. Im Gegensatz zum klassischen Modellbildungsprozess, der von jeweils einem Problembereich

(einer Domäne) zu einem Modell des Modellbereichs führt, beschreibt die Metamodellierung die im Modellbereich verwendete Modellierungssprache, das heißt die verwendbaren Modellelemente und deren Zusammenhänge (siehe Bild 3.5.1).

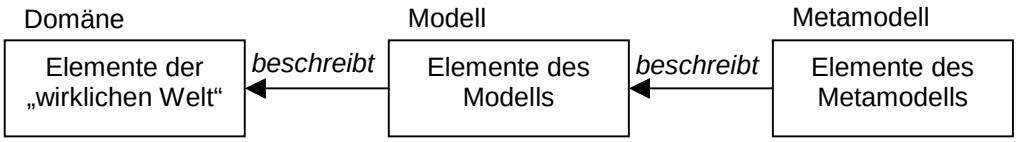


Bild 3.5.1 Vorgehensweise bei der Metamodellierung

Ein wichtiger Vorteil beim Einsatz von Metamodellen ist ihr Richtliniencharakter. Wird von mehreren Seiten an einem Modell gearbeitet z.B. durch den Einsatz verschiedener Komponenten für dasselbe Modell, so ist bei der Entwicklung die Vereinheitlichung durch ein Metamodell von großem Nutzen. Des Weiteren besteht die Möglichkeit, ein Modell gegen dieses Metamodell zu validieren. Liegen beide in einer formalen Beschreibung und in maschinenlesbarer Form vor, so kann dieser Vorgang sogar vollständig automatisiert werden. Soll ein Metamodell die Elemente und Konstrukte einer Modellierungssprache präzise definieren, so geht das nur, wenn auch die Elemente und Konstrukte des Metamodells präzise definiert sind. Dies erfordert ein Metamodell des Metamodells, ein so genanntes *Meta-Metamodell*. Dieses wird benötigt, wenn mehrere unterschiedliche Modellbereiche, die in einem Metamodell beschrieben sind, durch ein gemeinsames Modell zu beschreiben sind.

3.5.1.2 Gesamtübersicht

Im Folgenden werden eine Reihe von Grundannahmen vorgestellt, auf denen der im weiteren Verlauf dargestellte Ansatz basiert:

- Der Entwurf und die Nutzung einzelner Komponenten erfordert die Berücksichtigung unterschiedlicher Sichten und mit ihnen einhergehender Fachsprachen.
- Angesichts der Komplexität heutiger Systeme empfehlen sich Modelle auf verschiedenen Abstraktionsebenen. Um die Integration der mit verschiedenen Sichten korrespondierenden Modelle zu fördern, müssen die jeweils verwendeten Modellierungssprachen gemeinsame Konzepte aufweisen.
- Da Modelle als Medium der zwischenmenschlichen Kommunikation dienen, wird davon ausgegangen, dass graphische Darstellungen dazu besonders gut geeignet sind.
- Modelle sind in der Regel ausgesprochen umfangreich und weisen vielfältige Abhängigkeiten auf. Je präziser Modelle beschrieben sind, desto eher lassen sich gehaltvolle Aussagen über ihre Integrität machen. Gleichzeitig bieten sich Teile von Modellen für die Generierung von Kode bzw. die Durchführung von Simulationen an, was den Einsatz von Werkzeugen voraussetzt. Beide Aspekte erfordern eine formalsprachliche Beschreibung der Modelle.

Eine Berücksichtigung unterschiedlicher Sichten und mit ihnen unterschiedlicher Präsentationsmöglichkeiten ist notwendig, um den Wahrnehmungs- und Konzeptualisierungsmustern

verschiedener Anwendergruppen entgegenzukommen. Beispielsweise könnte ein Manager an einer graphischen Darstellung von Prozessaktivitäten interessiert sein, während ein Entwickler eine Darstellung der dazu benötigten Prozessressourcen in Form eines Balkendiagramms benötigt. Zur Definition anwendungsspezifischer Sichten und darauf basierender Anwendungsmodelle dient ein gemeinsamer Modellkern. Dieser erlaubt anwendungsspezifischen Werkzeugen, auf Sichten zu arbeiten, die unidirektional mit dem Gesamtmodell gekoppelt sind. Bild 3.5.2 zeigt symbolisch verschiedene Facetten individueller Sichten auf einen technischen Prozess, wie er beispielsweise in einer interdisziplinären Kooperation vorkommen könnte /Syrj-02/. Dabei lassen sich folgende Sichten unterscheiden:

- *Prozess-Sicht*: zur Grobplanung, Entwicklung und Konstruktion von Prozessmodellen. Die Darstellung erfolgt in Form eines Netzplans.
- *Dynamische Prozess-Sicht*: zur Validation, Simulation und Optimierung von Prozessmodellen. Diese Sicht zeigt den Prozess als graphisches Petrinetz.
- *Zeit-Sicht:* zur Steuerung und zum Controlling. Soll- und Ist-Werte werden bezüglich der Dauer der einzelnen Prozesse in einer Tabelle angezeigt.
- *Ressourcen-Sicht:* zur Kalkulation. Die mittlere Auslastung der verwendeten Ressourcen einzelner Prozesse wird mit Hilfe eines Balkendiagramms dargestellt.

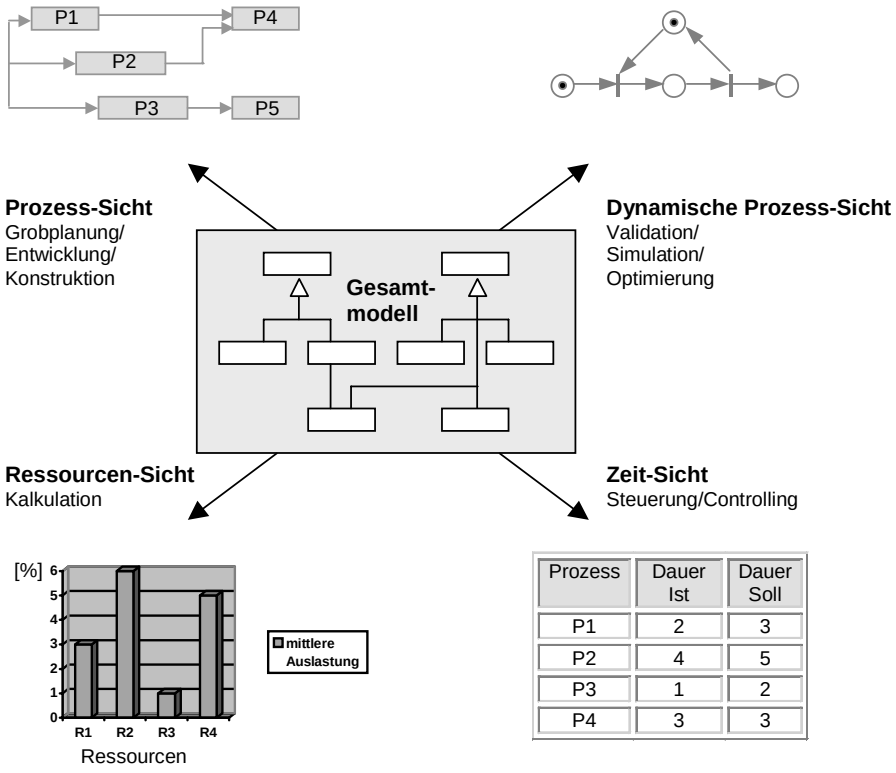


Bild 3.5.2 Interdisziplinäre Sichten auf einen technischen Prozess

Die Integration einzelner Modellierungssprachen erfolgt durch Verwendung eines einheitlichen Meta-Metamodells (siehe Bild 3.5.3). Bei den Metamodellen handelt es sich um Ob-

jektmodelle, die mit der UML (Unified Modeling Language) erstellt werden. Die einzelnen Objektmodelle der dedizierten Modellierungssprachen werden in einem gemeinsamen Objektmodell zusammengeführt, das als Grundlage für die integrierte Modellierungsumgebung dient. Die semantische Integration der verschiedenen Modelle wird durch gemeinsame Konzepte der jeweils verwendeten Sprachen erreicht. Die syntaktische Korrektheit eines Modells wird dadurch sichergestellt, dass die Beziehung zwischen Modell und Metamodell konsistent und vollständig sein muss. Ein Modell ist konsistent gegenüber dem zugrunde liegenden Metamodell, wenn alle im Modell verwendeten Notationselemente durch das Metamodell beschrieben werden. Vollständigkeit gegenüber dem zugrunde liegenden Metamodell liegt vor, wenn das Modell alle syntaktisch notwendigen Konstrukte der Notation enthält, die vom Metamodell vorgeschrieben werden.

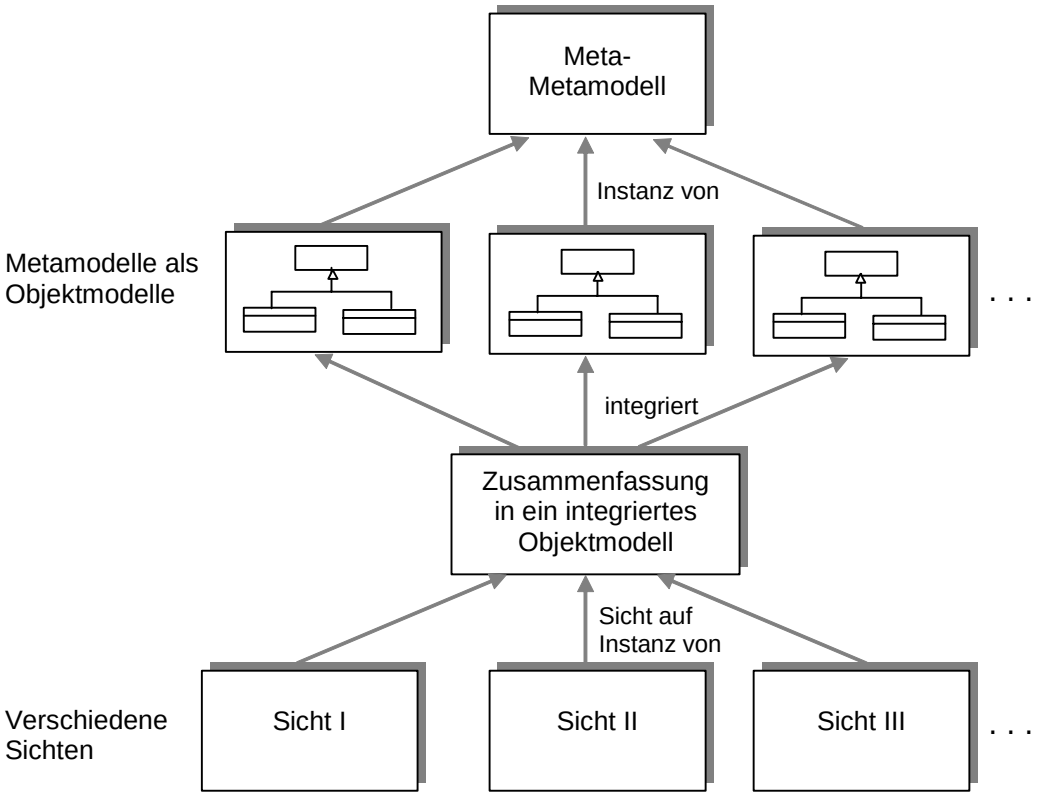


Bild 3.5.3 Gesamtübersicht

3.5.1.3 Ein Metamodell für stochastische Petrinetze

Zur Verdeutlichung des Metamodell-basierten Ansatzes wird in dieser Arbeit auf Petrinetze zurückgegriffen, die zu den gängigen formalen Sprachen zur Prozessmodellierung zählen. Sie zeichnen sich dadurch aus, dass sie eine präzise Semantik aufweisen und eine Integration mit Objektmodellen möglich ist. Das folgende Beispiel stellt ein stochastisches Petrinetzmodell dar, das einen elementaren Prozess beschreibt. Dieser ist ein Vorgang, der ein zeiterforderndes Geschehen mit definiertem Anfang und Ende beinhaltet und im allgemeinen dadurch gekennzeichnet ist, dass seine Durchführung eine Bereitstellung von Einsatzmitteln erfordert und damit Kosten verursacht. Die Marken in den Plätzen „Ankunft“ und „Ausgang“ modellieren Objekte, die auf die Bearbeitung warten bzw. bearbeitet sind. Die Marken im Platz „Res-

sourcen“ spezifizieren die für die Durchführung benötigten Ressourcen. Durch eine Unterscheidung des Transitionstyps lassen sich zeitliche Verzögerungen modellieren. Die Bearbeitungszeit kann beispielsweise mit Hilfe einer exponentiell verteilten Transition „Ende“ spezifiziert werden. Das entsprechende Petrinetzmodell ist in Bild 3.5.4 dargestellt.

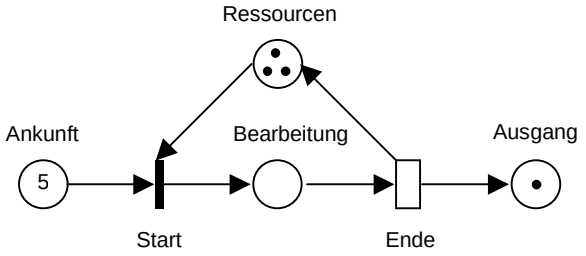


Bild 3.5.4 Stochastisches Petrinetz eines elementaren Prozesses

Wie in Abschnitt 2.2 beschrieben, besteht ein stochastisches Petrinetz aus einer Menge von Plätzen, Transitionen, Kanten sowie einer Anfangsmarkierung. Darüber hinaus können auch weitere Aspekte von Interesse sein wie beispielsweise eine graphische Darstellung, Beschriftungen und allgemeine Werkzeug-spezifische Informationen zu einem stochastischen Petrinetz. Dies kann wie in Bild 3.5.5 als UML-Klassendiagramm modelliert werden. Dieses spezifiziert eine Klasse *SPN*, die über *besteht_aus*-Aggregationen mit den Klassen *Place*, *Transition, Label* und *GeneralInfo* verbunden ist. Dabei wird durch die *Multiplizität* * ausgedrückt, dass ein stochastisches Petrinetz aus mehreren Plätzen, Transitionen und Kennzeichnungen bestehen kann. Entsprechend drückt die *Multiplizität 1* aus, dass ein stochastisches Petrinetz genau eine Beschreibung allgemeiner Informationen über das Netz hat. Die Klasse *Transition* hat eine weitere *besteht_aus*-Aggregation mit der *Multiplizität 1..** zur Klasse *Arc*, die besagt, dass eine Transition mit mindestens einem Platz über eine Kante verbunden sein muss. Die Repräsentation graphischer Information wird durch die Klasse *Graphics*, die über eine *besteht_aus*-Aggregation mit der Klasse *Coordinate* verbunden ist, definiert. Jeder Platz und jede Transition, Kante und Kennzeichnung ist mit graphischer Information ausgestattet, so dass diese Klassen über entsprechende *besteht_aus*-Aggregationen mit der Klasse *Graphics* verbunden sind.

Tabelle A2.1 im Anhang gibt eine Übersicht über die Semantik der einzelnen Klassenattribute basierend auf /Zimm-01/. Solch ein Modell einer Modellierungssprache stellt ein Metamodell für stochastische Petrinetze dar, das die möglichen mathematischen Strukturen beschreibt, die in dieser Modellierungssprache ausgedrückt werden können. Des Weiteren kann dieses Metamodell für verschiedene Varianten von stochastischen Petrinetzen verwendet werden (GSPN, DSPN, CDSPN, EDSPN), da unterschiedliche Schaltzeitverteilungen für Transitionen spezifiziert werden können. Eine Familie von Metamodellen kann mit Hilfe eines Meta-Metamodells beschrieben werden. Solch eine Spezifikation beinhaltet die grundlegenden Elemente, die für den Entwurf eines Metamodell-Formalisierungsmechanismus verwendet werden können. Falls neue Konzepte und Strukturen benötigt werden, so kann dies gewöhnlich auf einer Meta-Metaebene modelliert werden. Ein Meta-Metamodell vereinfacht den Austausch von Objektmodellen, die von verschiedenen Metamodellen instanziiert werden

dürfen. Ein weiterer Vorteil besteht in der Flexibilität mit der neue Formalismen entworfen werden können.

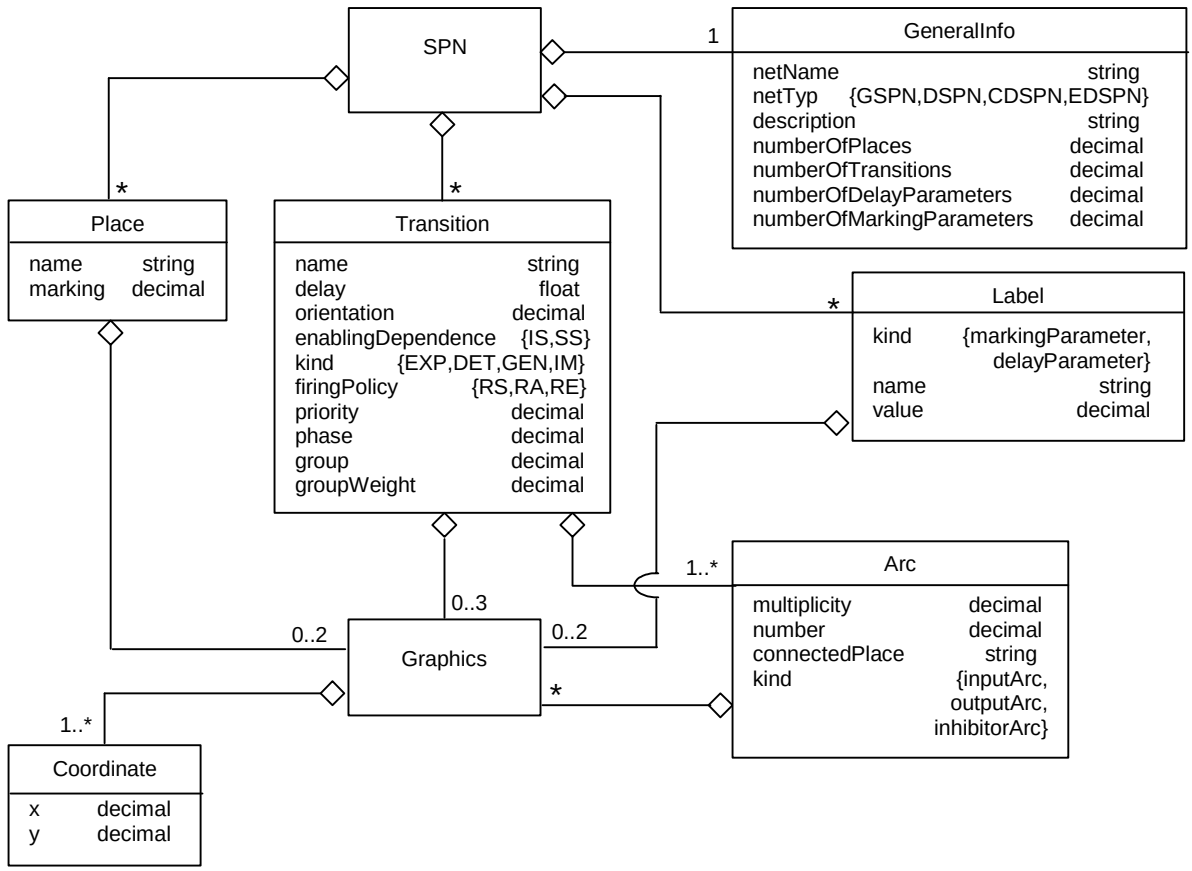


Bild 3.5.5 Metamodell zur Beschreibung von SPN-Modellen als UML-Klassendiagramm

3.5.2 Überblick über geeignete Standards für die Metamodellierung

Der in dieser Arbeit entwickelte Ansatz basiert auf einer Kombination folgender Standards der OMG (Object Management Group) /OMGr-04a/:

- *UML (Unified Modeling Language)*: Sie dient der (graphischen) Beschreibung von Modellen, die je nach Modellart z.B. die Struktur oder das Verhalten des modellierten Systems festlegen.
- *MOF (Meta Object Facility)*: Dieser Standard ist auf der Meta-Metamodellebene angesiedelt und definiert den Aufbau der Metamodellebene, auf der z.B. die UML liegt.
- *XMI (XML Metadata Interchange)*: Hierbei handelt es sich um ein XML-basiertes Austauschformat für UML-Modelle, das die Interoperabilität von Modellen zwischen heterogenen Modellierungswerkzeugen ermöglicht.

Die folgenden Abschnitte geben einen kurzen Überblick über die einzelnen Standards und zeigen deren vorteilhafte Eigenschaften auf.

3.5.2.1 Unified Modeling Language

Die UML /OMGr-04c/ ist heute die wohl am weitesten verbreitete Technik zur objektorientierten Beschreibung von Softwaresystemen. Sie entstand in den neunziger Jahren durch die Zusammenführung wichtiger objektorientierter Beschreibungstechniken. Anschließend wurde sie innerhalb der OMG standardisiert. Große Akzeptanz besitzt die UML im Bereich der Kommunikation von Entwürfen zwischen Entwicklern, insbesondere in der Analyse- und Design-Phase, bei denen es vor allem auf eine auf das Wesentliche reduzierte Darstellung ankommt. Im Hinblick auf den Einsatz der UML in technischen Applikationen gewinnt dagegen die Generierung von Kode und Testfällen aus UML-Modellen zunehmend an Bedeutung, wozu detaillierte und vollständige Modelle benötigt werden. Da Softwaresysteme oft zu komplex sind, um in einer einzigen Darstellung beschrieben werden zu können, sind mehrere einander ergänzende Modelle zur Darstellung spezifischer Sichten notwendig. Des Weiteren bildet die Modellierung der Struktur eine wichtige Basis in der Softwareentwicklung. Jedoch führt die inhärente Komplexität dazu, dass dem Verhalten und den Interaktionen der Softwarekomponenten eine bedeutendere Rolle zufällt. Aus diesen Gründen bietet die UML spezialisierte Diagramme zur Darstellung unterschiedlicher Aspekte der nachgebildeten Systeme an. Dabei erlaubt die UML die Konzentration auf eine für die jeweilige Aufgabenstellung hilfreiche Teilmenge der Gesamtsprache. Ein solcher Sprachkern kann beispielsweise aus den folgenden vier Modellierungsnotationen oder sogar nur einer Teilmenge davon bestehen:

- *Klassendiagramme:* zur Strukturdarstellung
- *Use-Case-Diagramme:* zur Darstellung von Anwendungsfällen
- *Statecharts:* zur Verhaltensbeschreibung
- *Sequenzdiagramme:* zur Darstellung von Interaktion

Eine wichtige Eigenschaft von UML ist ihre syntaktische und semantische Fundierung durch ein Metamodell /OMGr-04e/. Dieses definiert alle in UML verwendbaren Modellierungskonzepte und ihre Beziehungen. Auf der Basis dieses Metamodells lässt sich die UML an spezielle Anwendungsbereiche anpassen. Der dazu zur Verfügung gestellte Mechanismus „UML-Profil“ erlaubt die Definition neuer Modellierungselemente durch Spezialisierung bereits existierender. Zusammenfassend handelt es sich bei der UML um eine objektorientierte, ausreichend formalisierte und standardisierte Sprache, die neben einer umfangreichen Dokumentation auch Eigenschaften wie breite Akzeptanz, Verfügbarkeit und Erweiterbarkeit aufweist.

3.5.2.2 Meta Object Facility

Die MOF-Spezifikation /OMGr-04b/ beschreibt eine abstrakte Sprache und ein Rahmenwerk zur Verwaltung von plattformunabhängigen Metamodellen wie z.B. die UML. Der entscheidende Vorzug dieser Technologie ist die Bereitstellung objektorientierter Paradigmen zur Definition von Metamodellen sowie die Präsenz von Regeln, welche die automatische Ablei-

tung von Schnittstellen für so genannte „Repositories“ (siehe Abschnitt 3.4.2.3) ermöglichen. Die Modellierung nach MOF erfolgt dabei auf vier Ebenen (siehe Tabelle 3.5.1).

Schicht	**Beschreibung**	**Beispiel**
M3: Meta-Metamodell	Infrastruktur für eine Architektur zur Metamodellierung. Definiert die Sprache zur Spezifikation des Metamodells.	Meta-Klasse, Meta-Attribut, Meta-Operation, Meta-Beziehung
M2: Metamodell	Instanz eines Meta-Metamodells. Definiert die Sprache zur Spezifikation eines Modells.	Klasse, Attribut, Operation, Assoziation
M1: Modell	Instanz eines Metamodells. Spezifiziert Konzepte zur Beschreibung eines Anwendungsbereichs.	Klasse „Platz“, Klasse „Transition“, Attribut „Marke“
M0: Benutzer-Objekte	Instanz eines Modells. Definiert einen speziellen Fall des Anwendungsbereichs.	Platz „Ankunft“, Transition „Start_Prozess“, Marke „1“

Tabelle 3.5.1 4-Schichten-Metamodellierungsarchitektur

Der MOF-Standard definiert:

- eine Menge von Meta-Metamodellkonzepten, die zur Definition konkreter, MOF-konformer Metamodelle einsetzbar sind,
- Regeln für die Erzeugung von CORBA-IDL-Schnittstellen zum Zugriff auf Repositories für Modelle aus MOF-konformen Metamodellen sowie
- Regeln zur Erzeugung von XML-Dokumentenformaten (XML-DTD[2], XML-Schema) aus MOF-konformen Metamodellen zum Modellaustausch (enthalten in OMG-XMI).

Sind Metamodelle ausschließlich auf der Basis der in MOF enthaltenen Konstrukte beschrieben, so lassen sich mit den im Standard enthaltenen Regeln IDL-Schnittstellen für den Zugriff auf bzw. die Manipulation von Modellen automatisch erzeugen. Dabei wird ausgenutzt, dass das Metamodell alle Informationen über die Modellkonstrukte enthält. Die Erzeugung einer Implementierung für die automatisch erzeugten Schnittstellen ist ebenfalls automatisch möglich. Allerdings ist ein Verfahren für die Generierung solcher Implementierungen nicht standardisiert, sondern den Herstellern von entsprechenden Produkten überlassen. Sollen Modelle zwischen verschiedenen Repositories oder anderen Implementierungen wie z.B. Modellierungswerkzeugen ausgetauscht werden, so bieten sich zwei Möglichkeiten an: zum einen durch Benutzung der aus dem Metamodell erzeugten CORBA-IDL-Schnittstellen und zum anderen auf Basis von XML-Dokumenten. Das Format für diese XML-Dokumente ist durch eine XML-DTD oder XML-Schema vorgegeben und wird automatisch aus dem Metamodell erzeugt. Die zweite Methode wird dabei durch den separaten XMI-Standard ermöglicht. Zu-

[2] Eine **DTD** (Document Type Definition) stellt eine formale Spezifikation aller in einem Dokumenttyp erlaubten Strukturen dar. Man spricht auch von einer formalen Grammatik, in der zulässige Tags, Attribute und deren Verschachtelung definiert sind.

sammenfassend kann festgehalten werden, dass MOF eine geeignete Technologie für den Aufbau einer Modellierungsinfrastruktur darstellt.

3.5.2.3 XML Metadata Interchange

Der XMI-Standard /OMGr-04d/ spezifiziert, wie aus Objektmodellen XML-DTDs bzw. XML-Schematas zu erstellen sind. Da XML nicht objektorientiert ist, gäbe es unter Umständen mehrere Möglichkeiten, Objektmodelle nach XML abzubilden. XMI ermöglicht eine Standard-Abbildung von Objekten, die in UML definiert wurden, nach XML und bildet somit eine Brücke zwischen Objekten und XML. Dabei stellt XMI keine Erweiterung von XML dar, sondern basiert auf XML. Bild 3.5.6 veranschaulicht diese Beziehung.

XML-Dateien stellen Textdokumente dar, zu deren Austausch verschiedene Technologien eingesetzt werden können. Zusätzlich zum eigentlichen Modell kann auch dessen Metamodell ausgetauscht werden. Dieses wird benötigt, um die in einem Modell enthaltene Information interpretieren zu können, wenn auf der Empfängerseite das Metamodell nicht bekannt ist. Ermöglicht wird der Austausch von Metamodellen durch die Tatsache, dass die Konzepte zur Definition von Metamodellen durch das Meta-Metamodell festgelegt sind und die XMI-Generierungsregeln für XML-DTDs bzw. XML-Schematas aus Metamodellen auch für das Meta-Metamodell selbst angewandt werden können. XML-Dokumente, die der DTD bzw. dem Schema genügen und die aus dem Meta-Metamodell von MOF erzeugt wurden, stellen dabei MOF-konforme Metamodelle dar.

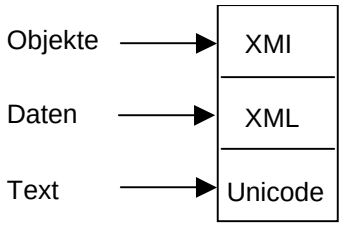


Bild 3.5.6 XMI basiert auf XML

Der Vorteil von XMI ist darin zu sehen, dass Entwickler von verteilten Systemen ihre Objektmodelle und andere Metadaten auf eine standardisierte Art und Weise über das Internet austauschen können. Das bedeutet, dass Anwendungen, die in einer kollaborativen Umgebung entwickelt werden, kompatibel zueinander sind und somit unnötige (Modell-)Redundanzen sowie Portierungsfehler bzw. –ungenauigkeiten beim Austausch vermieden werden.

3.6 Möglichkeit der technischen Realisierung

Eine mögliche technische Realisierung der in Abschnitt 3.2 beschriebenen Architektur zur Integration heterogener Komponenten für den Bereich der Modellierung und Simulation ist in Bild 3.6.1 dargestellt. Das System basiert auf einer Client/Server-Struktur, die in drei Schichten unterteilt ist. Auf der Klienten-Seite befindet sich die graphische Benutzungsschnittstelle (GUI), über die der Anwender entweder über Applikations-Klienten oder über das World

Wide Web auf die vom Applikations-Server zur Verfügung gestellten Werkzeugkomponenten zugreifen kann. Die Kommunikation mit dem Applikations-Server erfolgt dabei über eine geeignete Middleware. Zur Realisierung der dritten Architekturschicht, die dazu dient, Modellierungsdaten zu speichern, können gängige Dateiverwaltungs- bzw. Datenbanksysteme eingesetzt werden.

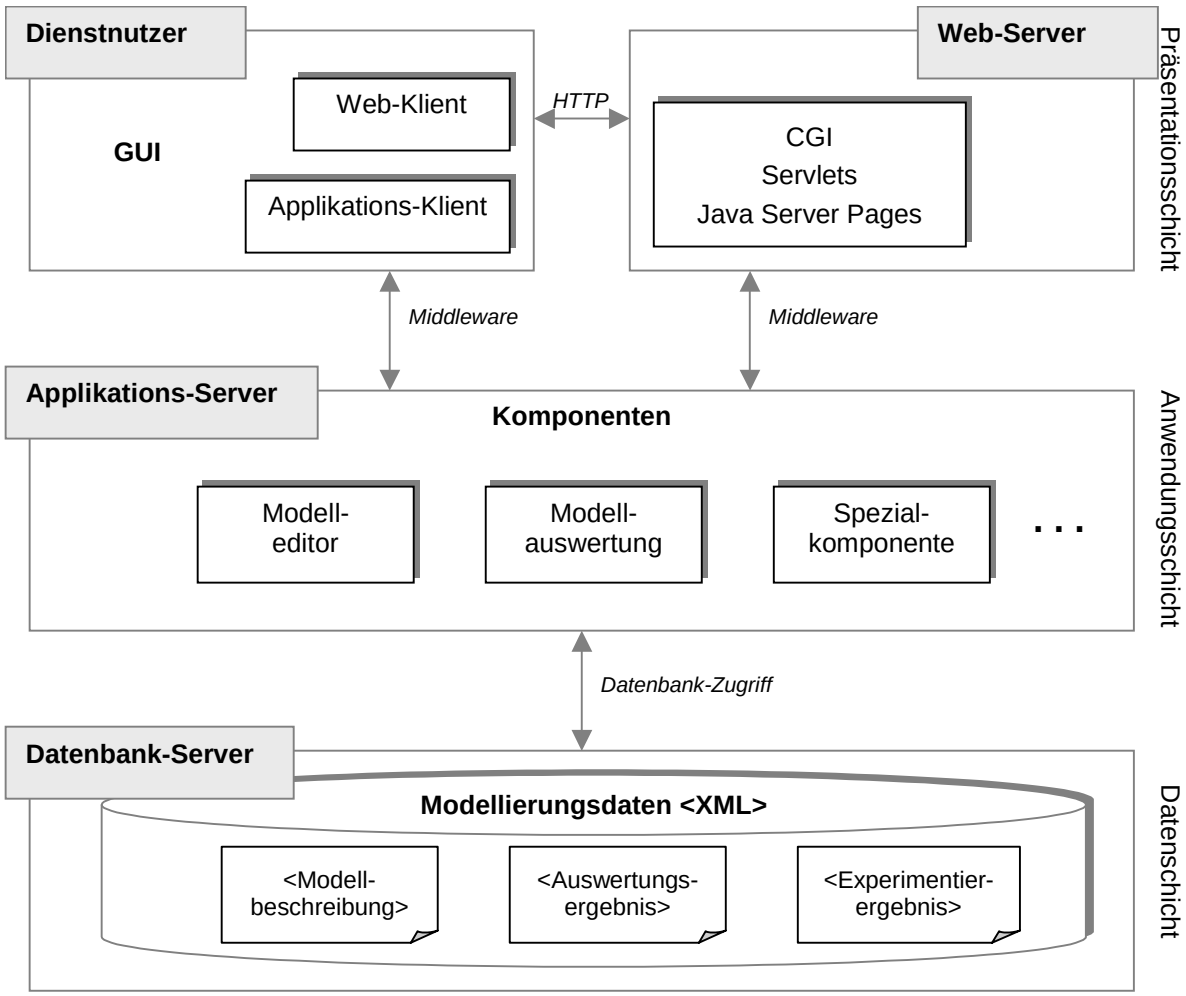


Bild 3.6.1 Mögliche technische Realisierung der Komponentenarchitektur

Durch die offene Systemarchitektur steht dem Benutzer eine Art Baukasten an Komponenten zur Verfügung. Da sämtliche Komponenten über das World Wide Web zugänglich sind, wird ein verteiltes Arbeiten an komplexen Modellen unterstützt. Dies ist insbesondere für interdisziplinäre Aufgabenstellungen von Vorteil, bei denen mehrere Experten aus unterschiedlichen Bereichen örtlich verteilt unter kooperativen Gesichtspunkten zusammenarbeiten müssen. In diesem Zusammenhang sind auch unterschiedliche Visualisierungsformen möglich, die auf die individuellen Sichten unterschiedlicher Benutzer abgestimmt sind.

4 Automatische Modelltransformationen

Dieses Kapitel stellt unterschiedliche Konzepte zur automatischen Modelltransformation vor. Dabei handelt es sich zunächst um eine automatisierte Gewinnung von XML-Strukturen aus UML-Modellen (vgl. Abschnitt 4.1). Ausgehend von einer XML-basierten Darstellung eines Modells befasst sich eine weitere Transformationsart mit der automatischen Erzeugung unterschiedlicher Dateiformate zur Integration von Altsystemen bzw. zur Benutzer-spezifischen Visualisierung (vgl. Abschnitt 4.2). Eine dritte Transformationsart bezieht sich auf eine Integration heterogener Modellierungsformalismen. Dazu wird ein Modell in einem anderen Formalismus erzeugt als demjenigen, mit dem es ursprünglich beschrieben wurde (vgl. Abschnitt 4.3).

4.1 Gewinnung von XML-Strukturen aus UML-Modellen

Ausgehend von der Verwendung der UML zur Metamodell-Spezifikation wie in Kapitel 3 beschrieben, besteht das Ziel bei dieser Art der Transformation darin, automatisch ein XML-basiertes Modellaustauschformat zu erzeugen. Die automatisierte Gewinnung von XML-Strukturen (DTDs oder XML-Schema) aus UML-Modellen erfolgt mittels der durch das XMI-Format definierten Generierungsprinzipien. Dieser Transformationsansatz vollzieht sich entsprechend Bild 4.1.1.

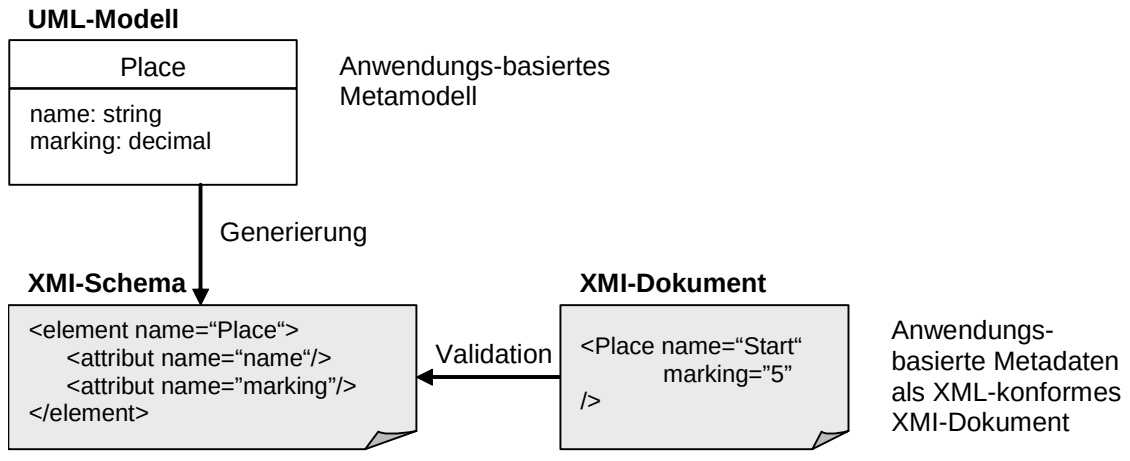


Bild 4.1.1 Zusammenhang zwischen einem UML-Modell, einem XMI-Dokument und einem XMI-Schema

Bild 4.1.1 zeigt den Zusammenhang zwischen einem UML-Modell, einem XMI-Dokument und einem XMI-Schema auf. Das UML-Modell stellt einen Platz eines SPN-Modells sowie

dessen Namen und Markierung dar. Prinzipiell gibt es mehrere Möglichkeiten, Informationen über einen Platz eines SPN-Modells in XML darzustellen, z.B. als Element oder als Attribut. XMI spezifiziert, wie aus diesem UML-Modell eine DTD bzw. ein Schema erzeugt wird. Jedes XMI-Dokument stellt dabei ein XML-Dokument dar, d.h. die Anwendungs-basierten Objekte werden nach XML konvertiert und können umgekehrt auch aus XML wiederhergestellt werden. Diese XML-Dokumente entsprechen jedoch dann dem XMI-Standard. Somit kann das mit UML spezifizierte Metamodell leicht angepasst werden, da die DTD bzw. das Schema leicht automatisch generiert werden können. Ein weiterer Vorteil entsteht dadurch, dass das XMI-Dokument anhand des XMI-Schemas validiert werden kann.

4.2 Transformationen zwischen heterogenen Dateiformaten

Eine XML-basierte Darstellung ermöglicht eine einfache Weiterverarbeitung. Beispielsweise können mit Hilfe eines XML-basierten Modellaustauschformats automatisch unterschiedliche Modell- und Präsentationsformate erzeugt werden. Dies erlaubt eine Integration verschiedener Alt- und Endsysteme, die jeweils über unterschiedliche Modell- bzw. Präsentationsformate verfügen. Damit wird zum einen ein Modellaustausch zwischen heterogenen Altsystemen unterstützt und zum anderen eine Applikations-spezifische Visualisierung ermöglicht (siehe Bild 4.2.1).

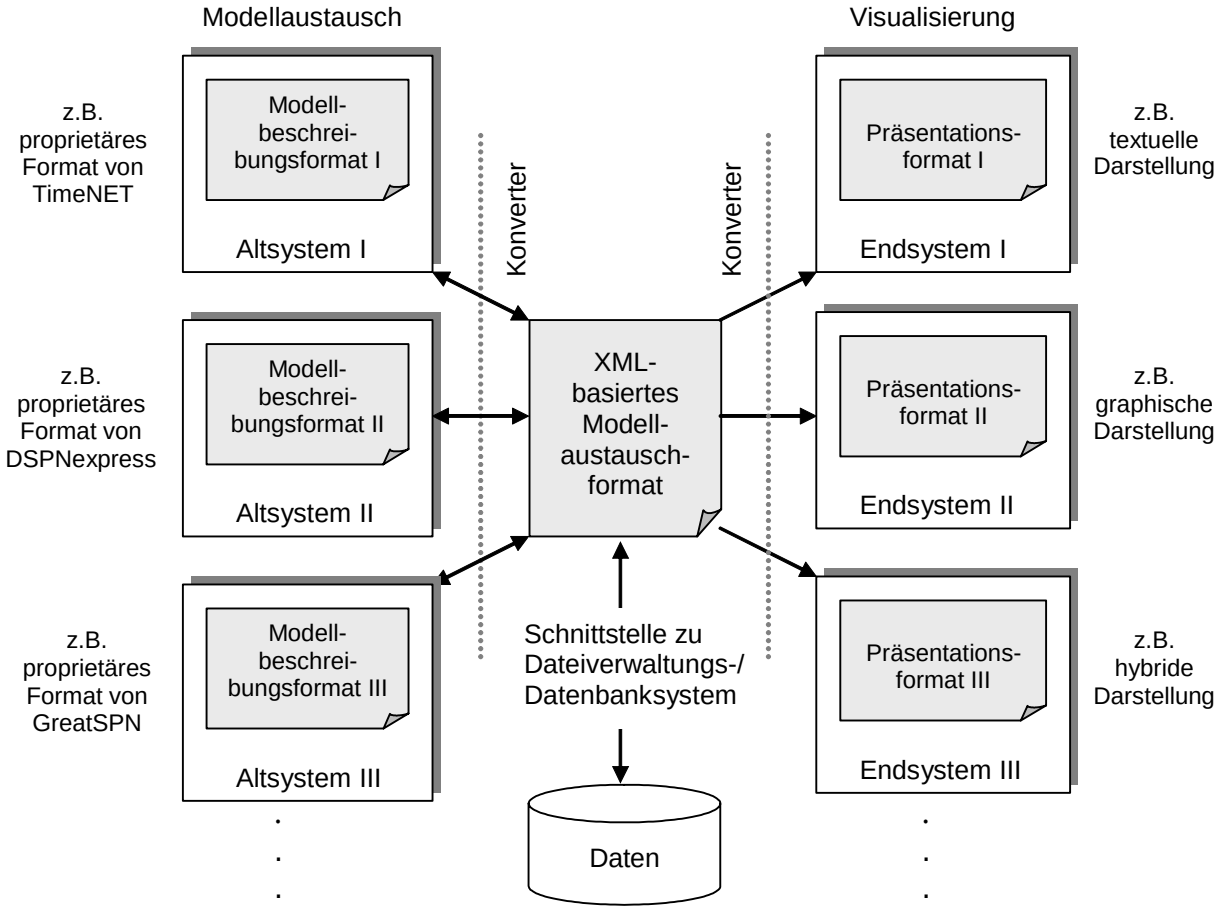


Bild 4.2.1 XML zur Datenintegration in einer heterogenen Modellierungsumgebung

Die automatischen Transformationen zwischen den unterschiedlichen Dateiformaten erfolgen mit Hilfe eines Konverters, wie in Bild 4.2.2 dargestellt. Auf Basis der Transformationssprache XSLT (XSL Transformations), welche als Teil von XSL (eXtensible Stylesheet Language) /W3Co-03b/ entwickelt wurde, wird das XML-basierte Modellaustauschformat in proprietäre Modellbeschreibungsformate bzw. proprietäre Präsentationsformate überführt. Dabei wird durch ein XSLT-Dokument definiert, wie die XML-Beschreibung zu transformieren ist. Für die Durchführung einer XSL-Transformation benötigt ein Programm, das auch als XSLT-Prozessor bezeichnet wird, sowohl ein XML- als auch ein XSLT-Dokument. XSLT-Transformationen erfolgen durch eine Reihe von Tests, die Teile des XML-Dokuments hinsichtlich Formatvorlagen vergleichen, die in einem XSLT-Dokument definiert sind. Zur Implementierung von XSL-Transformationen eignet sich Xalan, das ein bekanntes Werkzeug des Apache XML-Projekts darstellt.

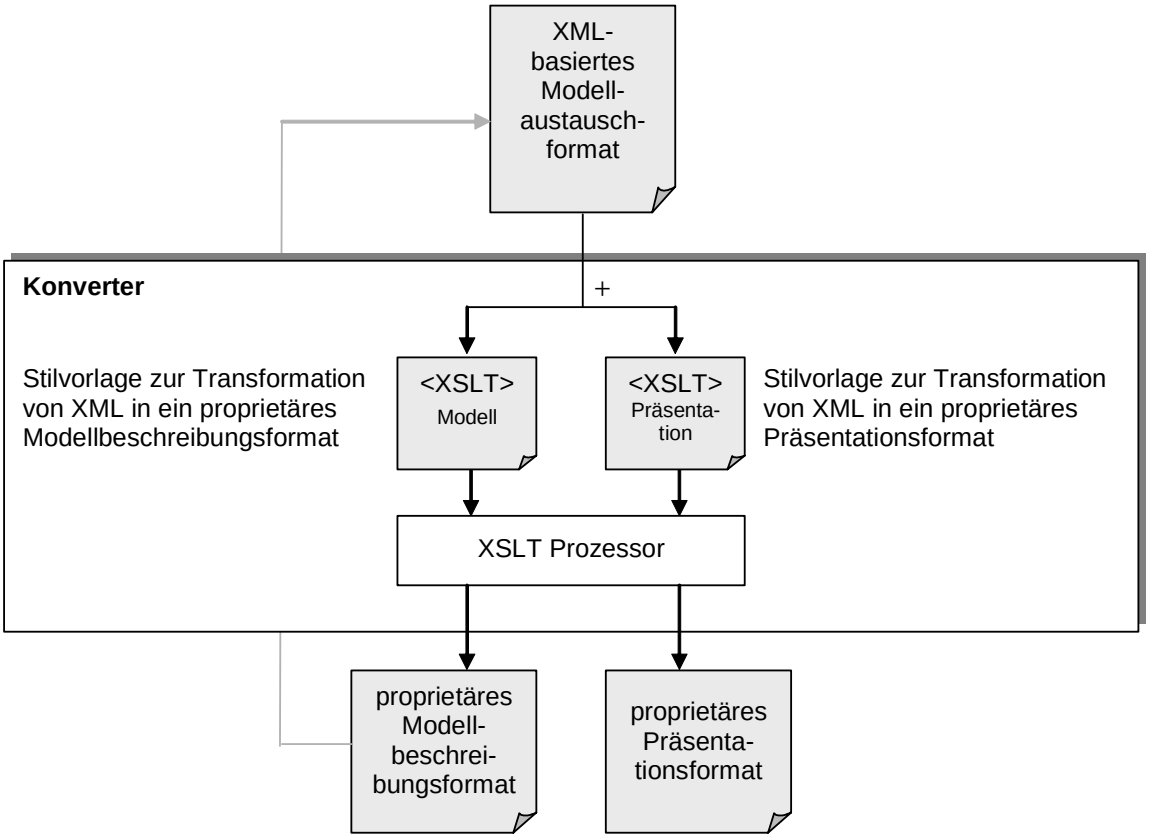


Bild 4.2.2 Architektur des Konverters

Zur Transformation eines proprietären Modellbeschreibungsformats eines Altsystems in das XML-basierte Modellaustauschformat können in der Regel keine Stilvorlagen (Stylesheets) verwendet werden, da es sich bei proprietären Modellbeschreibungsformaten oft um keine XML-basierten Dateien handelt. Diese muss der Konverter Zeile für Zeile importieren und die Informationen interpretieren. Im Anschluss daran, erfolgt eine Speicherung der Informationen gemäß dem XML-basierten Modellaustauschformat.

4.3 Transformationen zwischen heterogenen Modellierungsformalismen

Das Ziel bei dieser Transformationsart besteht darin, heterogene Modellierungsformalismen ineinander zu überführen. Die Notwendigkeit dafür entsteht dadurch, dass für eine Beschreibung und Analyse verschiedener Anwendungsbereiche in der Regel unterschiedliche Modellierungssprachen zum Einsatz kommen. Im Folgenden wird am Beispiel von Geschäftsprozessen aufgezeigt, wie ein Prozessmodell automatisch in ein Petrinetzmodell überführt werden kann.

Geschäftsprozesse können nach verschiedenen Kategorien (z.B. in Leistungs-, Unterstützungs- und Steuerungsprozesse) strukturiert und klassifiziert werden /Vers-97/. Nach dieser Einordnung richtet sich dann der jeweilige Detaillierungsgrad bei der Modellierung. Ein Prozess setzt sich aus einer Folge von Aktivitäten zusammen, die in einer logischen und zeitlichen Ablauffolge durchzuführen sind /Rump-99/. Prozesse können dabei durch so genannte Anordnungsbeziehungen (AOB) miteinander verknüpft werden. Eine Anordnungsbeziehung ist nach /DIN-69900/ definiert als „Quantifizierbare Abhängigkeit zwischen Ereignissen oder Vorgängen“. Im Einzelnen wird dabei zwischen folgenden Anordnungsbeziehungen unterschieden (siehe Bild 4.3.1):

- *Normalfolge (NF):* Anordnungsbeziehung vom Ende eines Vorgangs zum Anfang seines Nachfolgers.
- *Anfangsfolge (AF):* Anordnungsbeziehung vom Anfang eines Vorgangs zum Anfang seines Nachfolgers.
- *Endfolge (EF):* Anordnungsbeziehung vom Ende eines Vorgangs zum Ende seines Nachfolgers.
- *Sprungfolge (SF):* Anordnungsbeziehung vom Anfang eines Vorgangs zum Ende seines Nachfolgers.

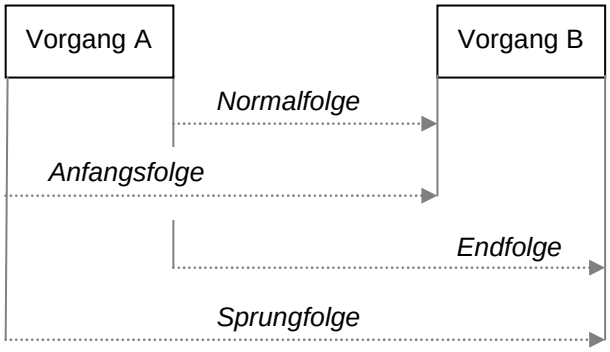


Bild 4.3.1 Anordnungsbeziehungen für die Prozessmodellierung

Die zeitliche Lage zweier Vorgänge zueinander lässt sich sowohl bei Überlappungen als auch bei seriellen Vorgängen durch unterschiedliche Anordnungsbeziehungen ausdrücken. Dies ist in Bild 4.3.2 dargestellt. Beispielsweise kann Vorgang B vier Zeiteinheiten vor Ende des Vorgangs A anfangen (siehe Bild 4.3.2 „Teilparallele Vorgänge“). Dieser Sachverhalt kann durch eine *Normalfolge* mit einem Zeitabstand von –4 modelliert werden. Allerdings kann dies auch wie folgt interpretiert werden: „Zwei Tage nach dem Start von A kann auch B be-

ginnen“. Daher lässt sich die Beziehung der beiden Vorgänge auch durch eine *Anfangsfolge* mit einem minimalen Zeitabstand +2 darstellen. Analog verhält es sich mit der *Sprungfolge*, die jedoch etwas komplizierter zu verstehen ist, da zeitlicher und logischer Ablauf nicht identisch sind. Beispiel: „Nachdem Wächter A mit der Wachablösung begonnen hat, darf Wächter B seine Wache beenden“. Logisch bedeutet dies: erst A, dann B. Zeitlich gesehen muss B allerdings schon vor A begonnen haben: erst B, dann A. Während in einem Netzplan A vor B abgebildet werden würde, veranschaulicht der Balkenplan die zeitliche Lage der Vorgänge. Im Beispiel sorgt allerdings der Zeitabstand von +7 dafür, dass das Ende von B zeitlich 7 Tage hinter dem Beginn von A liegt: B soll frühestens 7 Tage nach Beginn von A enden, der selbst nur sechs Tage dauert.

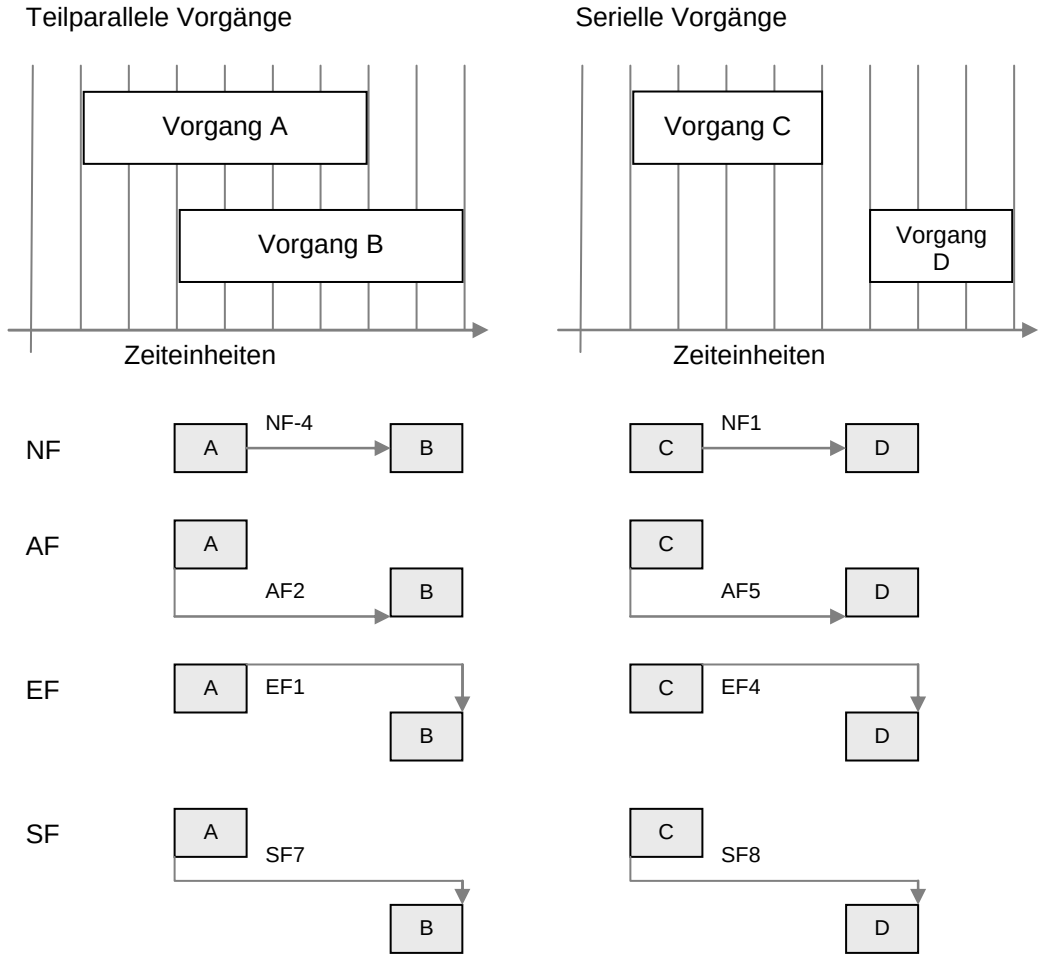


Bild 4.3.2 Anordnungsbeziehungen und Zeitabstände

Prozessmodelle dienen dazu, die logische Verkettung der Aktivitäten eines Geschäftsprozesses darzustellen. Der Einsatz der Prozesssicht ermöglicht eine schnelle und flexible Reaktion auf Veränderungen während einer Projektphase beispielsweise in Bezug auf die Arbeitsplanung. Bei den zu analysierenden Prozessen handelt es sich in der Regel um verteilte Prozesse, die durch gemeinsame Ressourcen miteinander gekoppelt sind. Um mit formalen Analysemethoden die Ablauffähigkeit dieser Prozesse sicherstellen zu können, eignen sich Petrinetze aus den in Abschnitt 2.2 dargestellten Gründen. Die automatische Transformation von einem Prozessmodell in ein Petrinetzmodell erfolgt in dieser Arbeit auf Basis eines Modul-basierten

Konzepts. Dazu werden für Prozesse und deren Kopplungsmöglichkeiten geeignete Petrinetzmodule spezifiziert /Müll-97/. Ein Petrinetzmodul für einen elementaren Prozess ist in Bild 4.3.3 dargestellt /Syrj-99/. Um ein Gesamtprozessmodell aus einzelnen Teilprozessmodellen erstellen zu können, werden die einzelnen Petrinetzmodule miteinander gekoppelt. Dazu existieren folgende Kopplungsmöglichkeiten:

- *Sequenz:* Bei der Sequenz werden mehrere Prozesse hintereinander ausgeführt. Hierbei ist die Ausführungsreihenfolge zu beachten.
- *Aufteilung:* Bei einer parallelen Bearbeitung wird ein Objekt von mehreren Prozessen gleichzeitig bearbeitet. Dadurch wird eine nebenläufige Abarbeitung unterstützt.
- *Auswahl:* Bei der Auswahl wird ein Objekt von einem aus insgesamt mehreren Prozessen bearbeitet. Diese Vorgehensweise erlaubt die Spezifikation von Alternativen.
- *Synchronisation:* Bei der Synchronisation benötigt ein Prozess die Resultate von mehreren Prozessen. Die Objekte der vorhergehenden Prozesse werden zu einem Objekt zusammengesetzt.
- *Vereinigung:* Bei der Vereinigung benötigt der nachfolgende Prozess das Ergebnis von mindestens einem aus mehreren vorhergehenden Prozessen.

Durch Spezifikation von gemeinsamen Plätzen lassen sich gemeinsame Ressourcen zwischen Teilprozessmodellen darstellen.

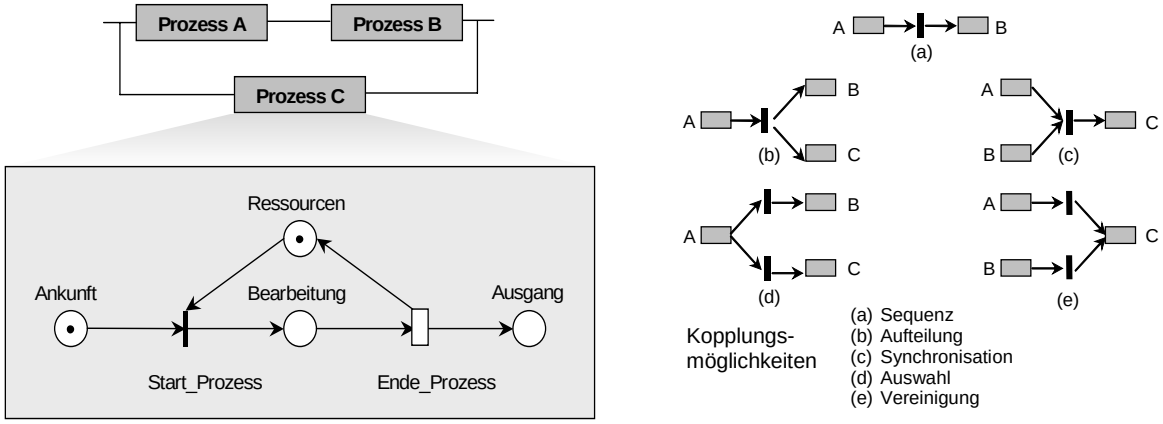


Bild 4.3.3 links: Petrinetzmodul für einen elementaren Prozess;
rechts: Kopplungsmöglichkeiten

Die Vorteile, die sich aus einer automatischen Modelltransformation ergeben, lassen sich wie folgt zusammenfassen:

- Es kann flexibel und innerhalb kurzer Zeit auf sich ändernde Randbedingungen oder Änderungen des Prozessmodells reagiert werden.
- Ein hoher Einarbeitungsaufwand in die Theorie der Petrinetze entfällt.

- Inkonsistenzen, die durch eine fehlerhafte Modelleingabe per Hand entstehen könnten, werden vermieden.

Des Weiteren wird durch diese Vorgehensweise eine Integration heterogener Modellierungsmethoden und -werkzeuge unterstützt, mit denen unterschiedliche Aufgabenbereiche aus der Modellierung, Simulation und Optimierung bearbeitet werden können.

5 Einsatz und Bewertung

In diesem Kapitel werden die in dieser Arbeit entwickelten Konzepte anhand eines komplexen Beispiels aus dem Bereich der Ingenieurwissenschaften validiert. Dazu wird zunächst in Abschnitt 5.1 der Einsatz der Methoden am Beispiel der Informationslogistik aufgezeigt. Im Anschluss daran wird in Abschnitt 5.2 ein Beispielszenario erläutert, das eine Kooperation von Unternehmen unterschiedlicher Branchen (Architektur, Bauingenieurwesen, Produktions- bzw. Fertigungstechnik, Informatik etc.) im Rahmen einer Projektabwicklung beschreibt. In Abschnitt 5.3 wird auf die allgemeine Vorgehensweise bei der „dynamischen Prozessoptimierung“ eingegangen und der Einsatz von Methoden aus der Modellierung, Simulation und Optimierung aufgezeigt. Abschnitt 5.4 stellt den Aufbau der Komponenten-orientierten Werkzeugarchitektur zur „dynamischen Prozessoptimierung“ dar, die auf den in dieser Arbeit entwickelten Entwurfskonzepten basiert. Anschließend erfolgt in Abschnitt 5.5 eine Beschreibung des zur Integration heterogener Komponenten aus dem Bereich der Modellierung und Simulation benötigten XML-basierten Modellaustauschformats. Eine Demonstration der Funktionalität anhand des entwickelten Prototyps erfolgt in Abschnitt 5.6. In diesem Zusammenhang wird die Handhabung bzw. die Benutzungsvielfalt der entwickelten Konzepte aus Anwendersicht anhand des beschriebenen Beispielszenarios aufgezeigt. Das Kapitel endet mit einer Bewertung in Abschnitt 5.7.

5.1 Einsatz der Methoden am Beispiel der Informationslogistik

Die Validation der entwickelten Konzepte erfolgte im Rahmen des vom Land Baden-Württemberg an der Universität Karlsruhe geförderten Forschungsschwerpunkts *„Informationslogistik für die internetbasierte Prozessinteraktion bei der branchenübergreifenden Kooperation“*. Die Gesamtzielsetzung dieses Projekts bestand darin, schlecht strukturierbare, verteilte Planungs- und Entwicklungsprozesse, wie sie bei einer Kooperation von Unternehmen unterschiedlicher Branchen auftreten können, zu beherrschen. Diese Kooperation ist in der Regel durch eine große Anzahl von Mitwirkenden, einem hohen Vernetzungsgrad der einzelnen Planungsaktivitäten, unscharfe und/oder unvollständige Information sowie sich häufig ändernde Randbedingungen gekennzeichnet. Durch die Einführung einer *Prozesssicht* zur Planung, Steuerung und Durchführung interdisziplinärer, branchenübergreifender Kooperationen können Vorgänge innerhalb von Unternehmen als Prozess mit aufeinander folgenden Schritten definiert werden. Hierzu zählen beispielsweise Abstimmungsvorgänge, definierte Arbeitsschritte oder Entscheidungsprozeduren. Durch die explizite Darstellung dieser Prozesse sind mehrere Vorteile gegeben. So können beispielsweise der Zeit- und Ressourcenbedarf für die unterschiedlichen Tätigkeiten im Unternehmen besser abgeschätzt, die gegenseitige Beeinflussung der Teilbereiche im Unternehmen sichtbar gemacht und schließlich Verbesserungs-

vorschläge für die Umgestaltung der Planungs- und Entwicklungsprozesse leichter gefunden werden.

Insgesamt kooperierten fünf verschiedene Institute aus drei verschiedenen Fakultäten (Architektur, Maschinenbau und Informatik) der Universität Karlsruhe miteinander, um unterschiedliche, miteinander vernetzte Aspekte beim Prozessmanagement interdisziplinärer Kooperationen zu erarbeiten. Diese waren:

- Institut für Industrielle Bauproduktion (ifib), Fakultät für Architektur
  Prof. Dr.ès.sc.techn. N. Kohler
- Institut für Prozessrechentechnik und Robotik (IPR), Fakultät für Informatik
  Prof. Dr.-Ing. R. Dillmann
- Institut für Rechnerentwurf und Fehlertoleranz (IRF), Fakultät für Informatik
  Prof. Dr.-Ing. D. Schmid
- Institut für Rechneranwendung in Planung und Konstruktion (RPK), Fakultät für Maschinenbau
  Prof. Dr.-Ing. Prof.E.h. Dr.h.c. H. Grabowski
- Institut für Werkzeugmaschinen und Betriebstechnik (wbk), Fakultät für Maschinenbau
  Prof. Dr.-Ing. D. Spath

Im Einzelnen handelte es sich dabei um die Aspekte *Prozessmodellierung (ifib)* und *Prozesssimulation* bzw. *–optimierung (IRF)*, die als Voraussetzung zur Strukturierung der zugrunde liegenden Kooperationen unter Berücksichtigung der Dynamik im Projektverlauf (veränderliche Randbedingungen und Zielvorstellungen) dienen sowie um ein projektbegleitendes *Prozesscontrolling (wbk)* für die Projektleitung. Eine *Informationsbereitstellung (IPR)* und eine *kontextorientierte Bereitstellung von Prozesswissen (RPK)* fungierten dabei als essentielle Basisdienste (siehe Bild 5.1.1).

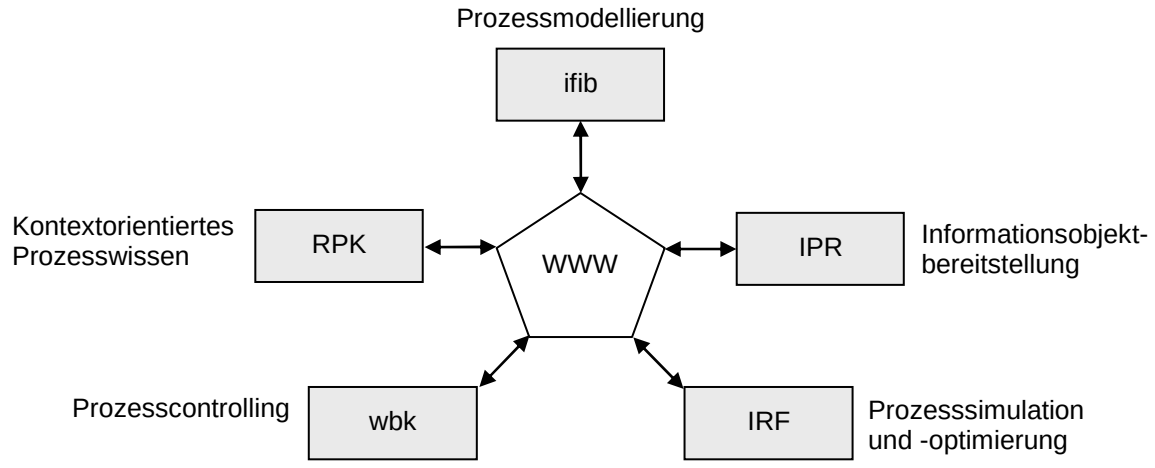


Bild 5.1.1 Komponenten für das Prozessmanagement interdisziplinärer Kooperationen

Das Zusammenspiel der einzelnen Komponenten ist in Bild 5.1.2 als Ablaufmodell für die Planungs- und Durchführungsphase beschrieben /GrKl-03/. Beginnend mit einer Problem-

stellung des Anwenders findet zunächst eine Aufgabenklärung in der Kooperationsumgebung statt (*Komponente Prozessmodellierung*). Anhand der Aufgabenstellung wird eine Suche nach entsprechenden Prozessen in der Wissensbasis gestartet, und die gefundenen Prozesse werden an die Kooperationsumgebung als Vorschlag übergeben (*Komponente Kontextorientiertes Prozesswissen*). Dort werden diese entsprechend der Aufgabenstellung angepasst, mit Zeit- und Ressourcenangaben versehen und anschließend die Prozesssimulation und –optimierung gestartet (*Komponente Simulation & Optimierung*). Die optimierten Prozesse dienen als Soll-Vorgabe für die nun folgende Durchführungsphase, in der zunächst abgefragt wird, ob Informationsobjekte benötigt werden (*Komponente Informationsobjektbereitstellung*). Ist dies nicht der Fall, dann wird sofort mit dem Prozesscontrolling fortgefahren (*Komponente Prozesscontrolling*). Ansonsten wird zuerst nach Informationsobjekten gesucht. Stellt sich dabei heraus, dass eine Änderung des Prozessablaufs notwendig ist, wird eine erneute Prozesssimulation und –optimierung angestoßen. Beim Prozesscontrolling werden Prozess-Soll-Werte mit Prozess-Ist-Werten verglichen. Falls der Ist-Wert zu stark vom Soll-Wert abweicht, findet eine neue Soll-Wert-Einschätzung statt und die Prozesssimulation und –optimierung wird erneut gestartet. Falls der Ist-Wert in Ordnung ist und keine weiteren Prozesse vorhanden sind, findet eine abschließende Bewertung der Prozesse statt, die dann in der Wissensbasis abgespeichert werden.

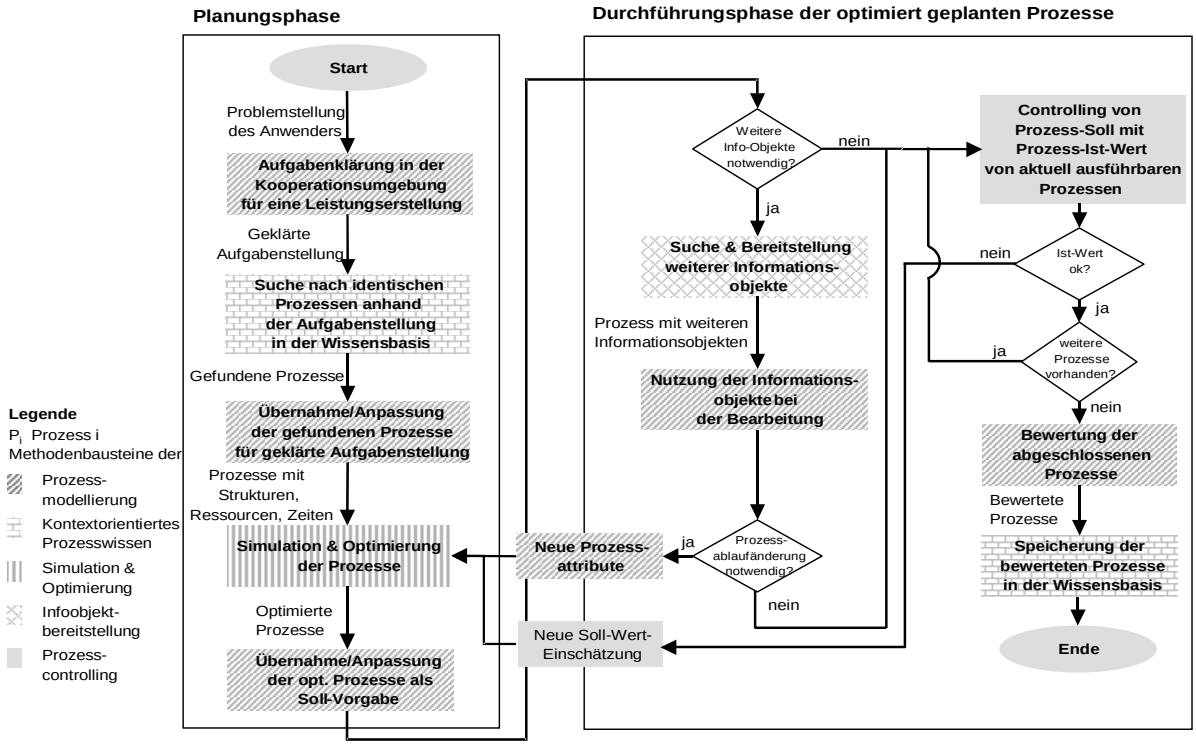


Bild 5.1.2 Ablaufmodell für die Planungs- und Durchführungsphase interdisziplinärer Kooperationen

Der folgende Abschnitt beschreibt zunächst ein Beispielszenario. Im Anschluss daran wird der Einsatz der Komponente Prozesssimulation und –optimierung für das Prozessmanagement interdisziplinärer Kooperationen aufgezeigt.

5.2 Beispielszenario

Im Beispiel ist ein Unternehmen mit der Aufgabe konfrontiert zu expandieren. Die angestrebte Ausweitung der Produktion, die Vergrößerung der Marketingabteilung und die damit verbundene Neustrukturierung der Verwaltung führen zu der Entscheidung, ein neues Verwaltungsgebäude zu planen und zu bauen. Bei der Planung des Bauvorhabens soll verstärkt Wert darauf gelegt werden, dass die beteiligten Fachplaner möglichst früh in den Planungsprozess integriert werden und untereinander ein Höchstmaß an Kommunikation und Kooperation verwirklichen. Um die Komplexität des Beispielszenarios in Grenzen zu halten, wird im Folgenden der Fokus auf den Bereich der Konzeptplanung gelegt. Die Strukturierung und der Ablauf der Planung wird am Beispiel der *Fassadenplanung* dargestellt. Bild 5.2.1 zeigt einen Projektstrukturplan, der den Gesamtzusammenhang des betrachteten Planungsbereichs verdeutlicht. In diesem Beispiel wird explizit die Objektplanung innerhalb der Konzeptplanungsphase untersucht. Die Objektkomponente stellt hierbei das Gebäude dar, dessen Fassade bearbeitet wird.

Mit der Planung der Fassade ist ein Team von Fachplanern betraut, das sich je nach Art und Komplexität der Aufgabe aus unterschiedlichen Experten zusammensetzt, die unter engem Austausch an Informationen untereinander die sie betreffenden Aufgaben bearbeiten. Die Aufgabenstellung für das Team ist die Konzeption einer repräsentativen Fassade mit hohem Wärme- und Schallschutz. Hierzu müssen die inhaltlich eng miteinander verbundenen und wechselseitig aufeinander wirkenden Bereiche wie die Gestaltung, das Tragwerk, die Bauphysik etc. untereinander abgestimmt werden. Dazu werden für die Fachplaner im Team einzelne *Arbeitspakete* erstellt. Diese enthalten die Beschreibung der Aufgaben, die nötigen Strukturinformationen, die Anforderungen und den Arbeitsaufwand.

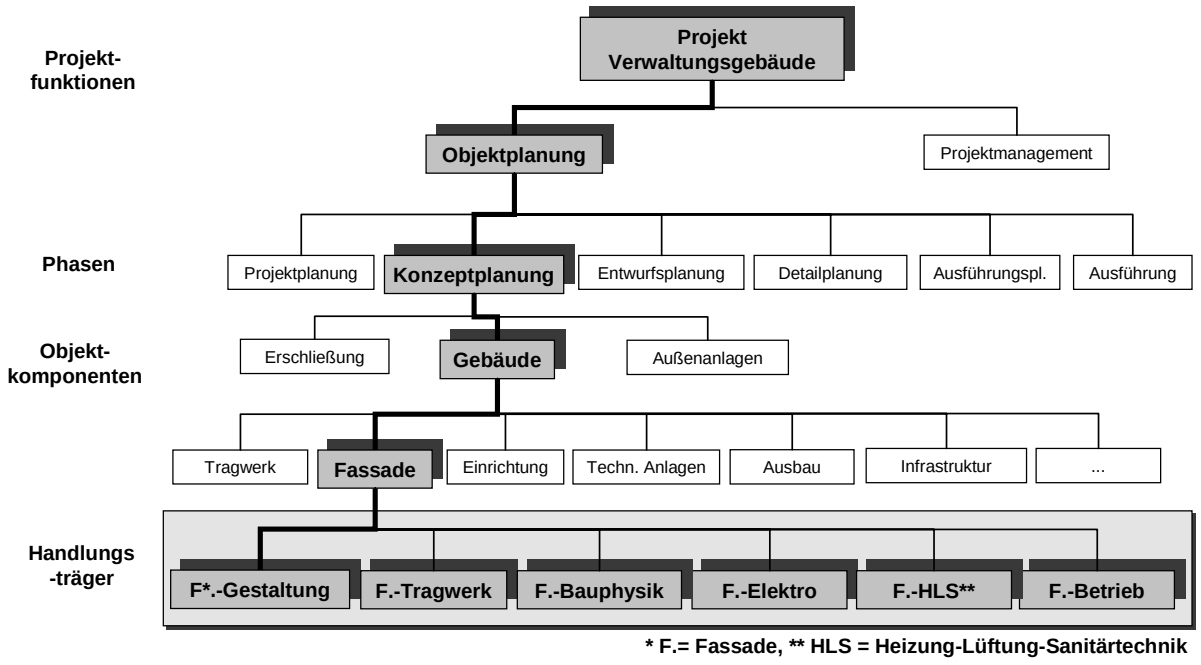


Bild 5.2.1 Projektstrukturierung

Diese Arbeitspakete werden dann in einzelne *Prozesse* überführt und in eine Ablauflogik gebracht, welche die Dauer und Anordnungsbeziehungen der Prozesse darstellt und die Kapazitäten der Bearbeiter bereits enthält. Im Beispiel wird der Rahmenprozess „Fassadenplanung“ des Teams „Fassade“ betrachtet. In diesem sind die einzelnen Prozesse der Fachplaner enthalten (Bild 5.2.2). Dabei stehen die eingesetzten Planer für das Projekt in der Regel alle Vollzeit zur Verfügung (8h/Tag entspricht 100%).

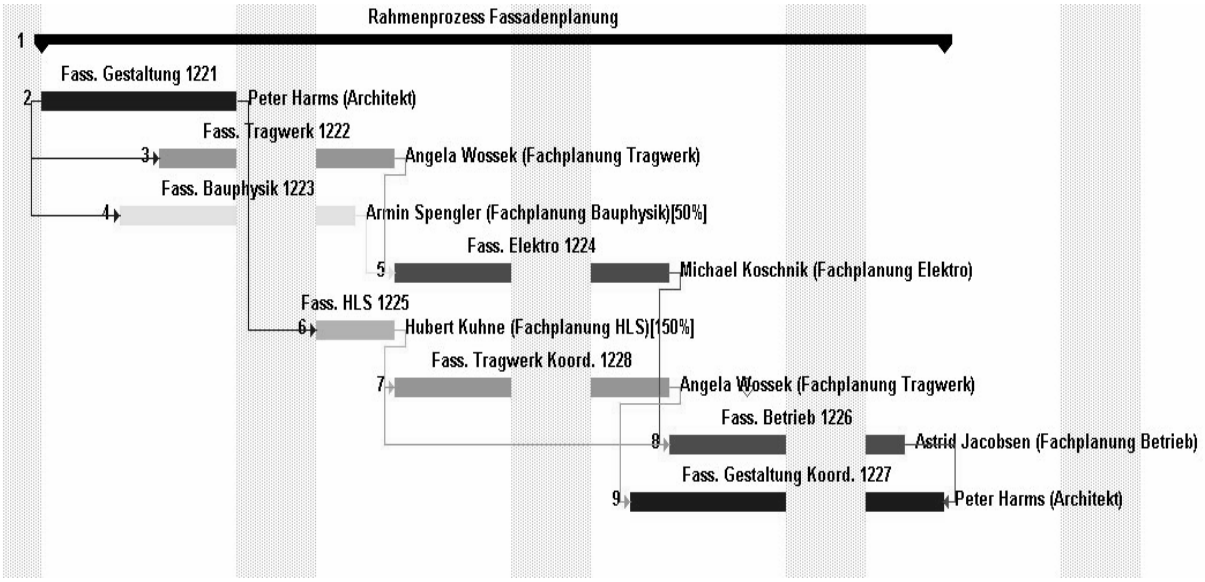


Bild 5.2.2 Rahmenprozess „Fassadenplanung“

Der Architekt beginnt im Prozess *Fassade Gestaltung* mit den grundsätzlichen Vorgaben für die Fassade. Mit einer gewissen Zeitüberlappung beginnen die Fachplaner mit dem Tragwerk und der Bauphysik und legen darauf aufbauend die weiteren Strukturen für den Baukörper fest. Anschließend ergänzen die Fachplaner der technischen Gebäudeausstattung (Elektro, HLS) ihre grundsätzlichen Festlegungen zur Leitungsführung. Die Tragwerksplanung koordiniert die Auswirkungen auf die Statik und die Fachplanung „Betrieb“ (Bsp. Fassadenreinigung) entwickelt ein Betriebskonzept. Aufbauend auf dem Ergebnis der Fachplaner wird das erste Gestaltungskonzept von dem Architekten nochmals überarbeitet.

5.3 Vorgehensweise bei der „dynamischen Prozessoptimierung“

Um Entscheidungsverantwortlichen umfangreiche Informationen über ihre Handlungsmöglichkeiten geben zu können, werden die entworfenen Prozessmodelle implementiert, systematisch analysiert und optimiert /Syrj-03/. Dabei sind folgende Randbedingungen zu beachten:

- Da das erstellte Prozessmodell sehr viele Einflussgrößen aufweist und nur für eine eng begrenzte Projektlaufzeit gültig ist, müssen die Modellbildung sowie die im Anschluss daran durchgeführten Modellanalysen und –optimierungen weitestgehend automatisiert und möglichst schnell erfolgen. Zusätzlich muss flexibel und innerhalb kurzer Zeit auf sich ändernde Randbedingungen reagiert werden können.

- Aufgrund der kurz- und mittelfristig angelegten Prozesse müssen Analyse- und Optimierungsmethoden eingesetzt werden, deren Bedienung auch von Nichtexperten ohne längeres Einlernen durchgeführt werden kann.
- Die hochgradig parallel ablaufenden Teilprozesse müssen hinsichtlich ihrer Funktionsfähigkeit und Korrektheit durch formale Methoden überprüft werden, da durch Simulation nicht alle auftretenden Situationen untersucht werden können.

Die derzeit in der Praxis eingesetzten Sprachen zur Prozessmodellierung sind zwar intuitiv verständlich, aber nicht unmittelbar formaler Analyse bzw. Simulation zugänglich. Der Einsatz von formalen Modellierungsmethoden auf der Basis von Petrinetzen verspricht dagegen viele Vorteile im Bereich der Analyse von Verklemmungen oder der schnellen Optimierung. Des Weiteren bieten Petrinetze die Möglichkeit, Prozessabläufe dynamisch darzustellen. Wie in Bild 5.3.1 dargestellt, besteht die allgemeine Vorgehensweise zur „dynamischen Prozessoptimierung“ insgesamt aus einer Modellierungs-, einer Analyse- und einer Optimierungsphase, die im Folgenden näher erläutert werden:

- *Modellierungsphase (Schritt 1):* In dieser Phase wird ein Prozessmodell automatisch in ein formales Modell basierend auf Petrinetzen überführt (wie in Abschnitt 4.3 beschrieben), da für eine quantitative Analyse und Auswertung von Planungs- und Entwicklungsprozessen eine formale Modellierungsmethode benötigt wird.
- *Analysephase (Schritte 2, 3 und 4):* In der anschließenden Analysephase wird das erzeugte Petrinetzmodell hinsichtlich der Struktur, des Verhaltens und der Parametersensitivität (wie in Abschnitt 2.2 beschrieben) untersucht. In diesem Zusammenhang ist es beispielsweise wichtig sicherzustellen, dass weder Verklemmungen noch Ressourcenengpässe auftreten.
- *Optimierungsphase (Schritt 5):* Nachdem sichergestellt wurde, dass das erstellte Prozessmodell ablauffähig ist und eine korrekte Grundstruktur aufweist, erfolgt die Optimierungsphase. Diese dient dazu, das Petrinetzmodell hinsichtlich bestimmter Zielgrößen mit Hilfe von direkten Optimierungsverfahren (wie in Abschnitt 2.2.4 beschrieben) zu optimieren. Optimierungsziele sind beispielsweise die Verringerung von Durchlaufzeiten oder eine bessere Ressourcenausnutzung.

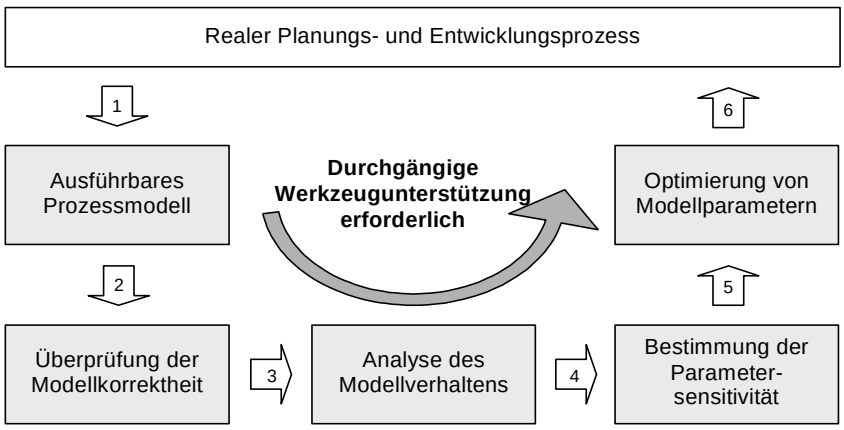


Bild 5.3.1 Vorgehensweise bei der dynamischen Prozessoptimierung

Für die Durchführung der einzelnen Arbeitsschritte wird eine durchgängige Werkzeugunterstützung benötigt. Im folgenden Abschnitt wird dazu ein Werkzeug zur „dynamischen Prozessoptimierung“ beschrieben, welches auf der in dieser Arbeit vorgeschlagenen Architektur zur Integration heterogener Komponenten basiert.

5.4 Architektur eines Werkzeugs zur dynamischen Prozessoptimierung

Die Komponenten-orientierte Werkzeugarchitektur zur dynamischen Prozessoptimierung wurde basierend auf den in dieser Arbeit beschriebenen Entwurfskonzepten entwickelt. Bei den integrierten Modellierungswerkzeugen handelt es sich um die bewährten Petrinetz-Werkzeuge GreatSPN /GSPN-03/, DSPNexpress /Lind-03/ und TimeNet /TNET-03/, die leistungsfähige Module zur analytischen und simulativen Auswertung von stochastischen Petrinetzen bereitstellen. Wie in Bild 5.4.1 dargestellt, unterstützt jedes dieser Modellierungswerkzeuge ein eigenes proprietäres Modellbeschreibungsformat. Um die dadurch hervorgerufene Inkompatibilität aufzuheben, wurde ein einheitliches *XML-basiertes Modellbeschreibungsformat für stochastische Petrinetze* entwickelt /Schi-02, SySy-03/, das sich weitgehend an Vorschlägen /PNMe-00, JüKW-00a, JüKW-00b/ orientiert, die bislang im Rahmen der von der ISO vorangetriebenen Standardisierung eines XML-basierten Modellaustauschformats für erweiterte Petrinetze unterbreitet wurden /PNSt-03/. Der Austausch von Modellbeschreibungen zwischen den integrierten Werkzeugen wird durch bidirektionale *Konverter* ermöglicht /SySy-03/, welche die proprietären Modellbeschreibungsformate von GreatSPN, DSPNexpress und TimeNet in das XML-basierte Modellbeschreibungsformat für stochastische Petrinetze umwandeln und umgekehrt.

Um die für die dynamische Prozessoptimierung erforderliche Optimierungsfunktionalität bereitzustellen, wurde eine *Experimentierkomponente* /SySy-01/ eingebunden, die eine Bausteinbibliothek bestehend aus direkten Optimierungsverfahren zur Verfügung stellt. Diese haben als allgemein anwendbare Heuristiken zur Black-Box Optimierung sowie zur Lösung NP-harter Problemstellungen aus der Kombinatorik große Bedeutung erlangt. Die Experimentierkomponente verfügt sowohl über globale probabilistische Optimierungsmethoden als auch über lokale deterministische Hill-Climbing Strategien. Darüber hinaus werden auch hybride Optimierungsverfahren zur Verfügung gestellt, welche die Vorteile globaler und lokaler Suchstrategien vereinen.

Die Einbindung der Experimentierkomponente erfordert neben losen Integrationskonzepten auf Basis von Dokumentenaustausch auch den direkten Zugriff auf die Modellauswertungsmodule. Zur Realisierung dieser Zugriffe wurde mit CORBA ein Middlewarestandard eingesetzt, der eine flexible Verteilung der beteiligten Komponenten auf heterogene physikalisch möglicherweise weit voneinander entfernt lokalisierte Plattformen erlaubt. Um den CORBA-Zugriff der Experimentierkomponente auf die Modellauswertungsmodule zu ermöglichen, wurden die dazu notwendigen Aufrufschnittstellen durch entsprechende *Adapter* nachgebildet /SySy-01/.

Der Austausch von deskriptiven Modellbeschreibungen mit externen Systemen erfolgt durch eine XML-basierte Planungsprozessbeschreibung. Die Transformation dieser Beschreibung in das XML-basierte Austauschformat für stochastische Petrinetze wird durch einen Konverter ermöglicht. Für die Bewerkstelligung dieser Transformation wurde auf den bereits

beschriebenen Petrinetz-basierten Modulbaukasten zur Darstellung von Planungsprozessen und deren Kopplungsmöglichkeiten zurückgegriffen. Des Weiteren wird ein Konverter benötigt, um die Ergebnisse der Optimierung wieder in das XML-basierte Format der deskriptiven Planungsprozessbeschreibung zu überführen.
Auf die Funktionalität der entwickelten Architektur kann über eine *Web-basierte Benutzungsschnittstelle* /SySy-02/ zugegriffen werden, wobei der Benutzer bei der Planung und Durchführung von Experimenten sowie der Auswertung und Darstellung der Ergebnisse durch Assistenten (Wizards) unterstützt wird.

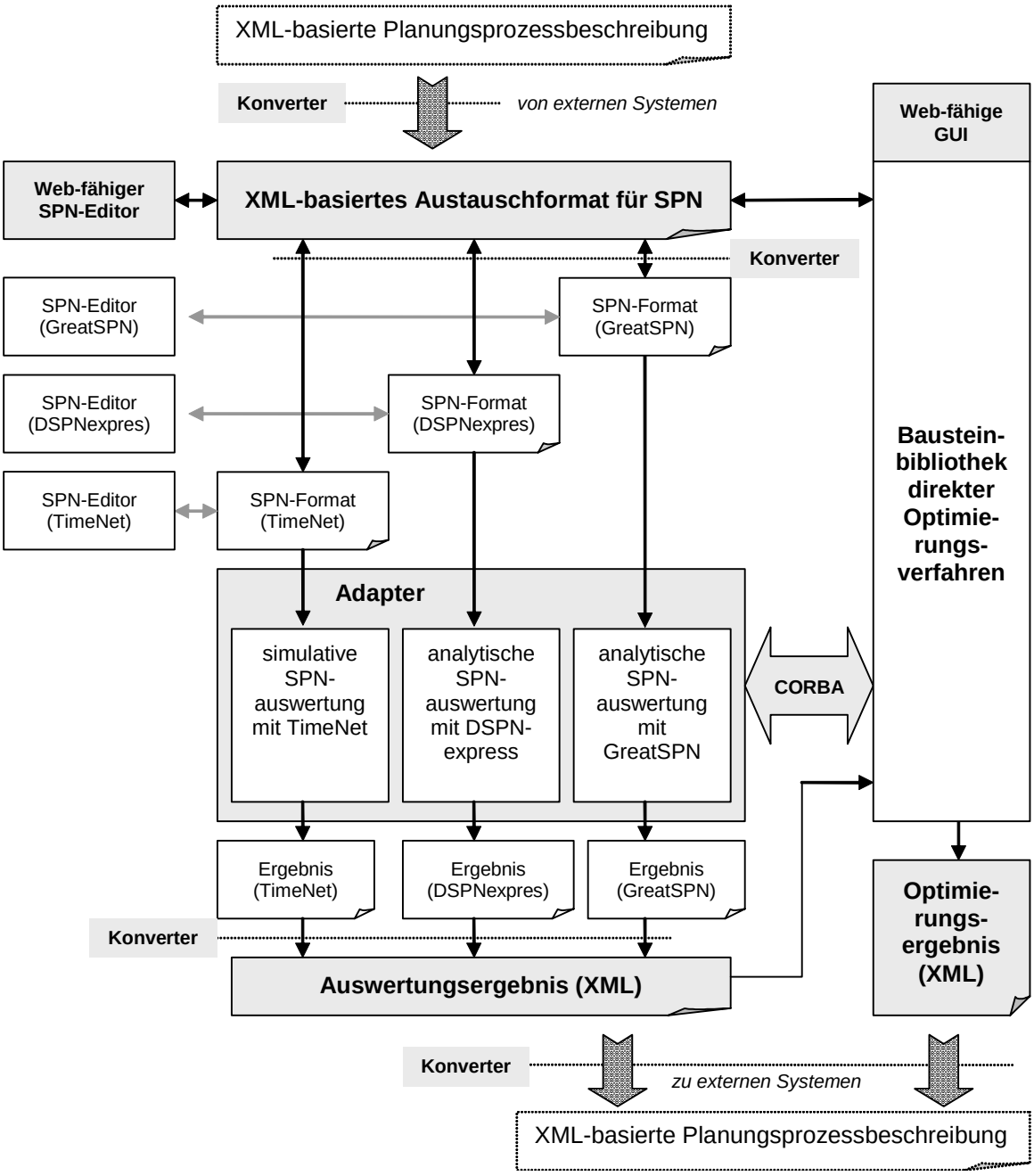


Bild 5.4.1 Komponentenorientierter Aufbau eines Web-basierten Werkzeugs zur dynamischen Prozessoptimierung

5.5 Ein XML-basiertes Modellaustauschformat für SPN

XML-Dokumente stellen baumartige Strukturen dar, die aus ineinander geschachtelten Auszeichnungsbefehlen und Daten bestehen. Dabei wird zwischen Elementen und Attributen unterschieden. Um die Anzahl der Sub-Elemente anzeigen zu können, existieren unterschiedliche Element-Deklarationssymbole. Tabelle 5.5.1 gibt eine Übersicht über die verwendeten XML-Elemente und –Attribute des in dieser Arbeit entwickelten XML-basierten Modellaustauschformats für SPN sowie deren Wertebereich. Die erste Spalte gibt dabei die Hierarchieebene an, die zweite und dritte Spalte enthält die Eltern- und deren Sub-Elemente und in Spalte vier und fünf sind die Attribute sowie deren Wertebereich angegeben.

Um XML zum Datenaustausch verwenden zu können, ist es wichtig zu definieren, wie XML-Dokumente strukturiert werden sollten. Zur Strukturierung von XML-Dokumenten können sowohl DTD's als auch XML-Schemas verwendet werden. In einer DTD werden die verschiedenen Arten von Elementen, die in einem validen Dokument auftreten dürfen und die erlaubten Muster von verschachtelten Elementen definiert. DTDs definieren außerdem die Attribute, die in einem Element beinhaltet sein können mit Hilfe einer so genannten ATTLIST. Obwohl eine DTD in das Dokument, dessen Syntax sie definiert, eingebettet sein kann, werden DTDs typischerweise in externen Dateien gespeichert und durch das XML-Dokument mit Hilfe eines Universal Resource Identifiers (URI) referenziert. XML-Schema ist eine weitere Initiative des World Wide Web Consortiums (W3C), mit deren Hilfe XML-Daten genauer beschrieben werden können als mit DTDs, da zahlreiche vordefinierte Datentypen verfügbar sind. Im Gegensatz zu DTDs stellen XML-Schemas selbst ein XML-Dokument dar, während DTDs in einer anderen Syntax gehalten sind.

Eine DTD bzw. ein XML-Schema stellen eine formale Spezifikation der Grammatik für ein spezifisches XML-Vokabular dar. Sie dienen dazu, den Inhalt eines XML-Dokuments hinsichtlich der spezifizierten Grammatik zu validieren. Des Weiteren wird der strukturierte Informationsaustausch zwischen kollaborierenden Anwendungen in einer Plattform- und Middleware-neutralen Art und Weise ermöglicht. Sowohl Validation als auch Verarbeitung können durch XML-Parser ausgeführt werden. Dazu werden zwei Alternativen unterschieden: DOM (Document Object Model) und SAX (Simple API for XML). DOM ist eine vom W3C unterstützte Standard-Anwendungsprogrammierschnittstelle (API), die eine Plattform- und Sprach-unabhängige Schnittstelle anbietet, um den Zugang zu Inhalt und Struktur baumbasierter Dokumente zu ermöglichen /DoOM-03/. Im Gegensatz zu DOM handelt es sich bei SAX um einen ereignisbasierten Parser. Durch ein benutzerdefiniertes Ereignissteuerungsprogramm wird gemeldet, wenn Elemente in einem XML-Dokument gefunden werden. Das Steuerungsprogramm nutzt diese Information, um anwendungsspezifische Verarbeitungsziele auszuführen /Saxp-03/.

E.	XML-Eltern-Element	XML-Sub-Element	XML-Attribut	Wertebereich
0	<spn>			
		<generalInformation>	netName (required)	string
			netType (required)	{GSPN,DSPN,CDSPN, EDSPN}
			description (implied)	string
		<place>	-	-
		<transition>	enablingDependence (required)	{IS, SS}
			kind (required)	{EXP,DET,GEN,IM}
			firingPolicy (required)	{RS, RA, RE}
			priority (required)	decimal
			phase (implied)	decimal
			group (implied)	decimal
			groupWeight (implied)	decimal
		<label>	kind (required)	{markingParameter, delayParameter}
1	<generalInformation>			
		<numberOfPlaces>	-	-
		<numberOfTransitions>	-	-
		<numberOfDelayParameters>	-	-
		<numberOfMarkingParameters>	-	-
	<place>			
		<name>	-	-
		<marking>	-	-
		<graphics>	kind (required)	{positionPlace, positionTag}
	<transition>			
		<name>	-	-
		<delay>	-	-
		<graphics>	kind (required)	{positionTransition, positionTag, positionDelay}
		<orientation>	-	-
		<arc>	-	-
	<label>			
		<name>	-	-
		<value>	-	-
		<graphics>	-	-
2	<arc>			
		<multiplicity>	-	-
		<connectedPlace>	-	-
		<number>	-	-
		<kind>	-	-
		<graphics>	-	-
3	<graphics>			
		<coordinate>	x (required)	decimal
			y (required)	decimal

Tabelle 5.5.1 XML-Elemente und –Attribute des Modellaustauschformats für SPN

5.6 Prototypische Implementierung

In diesem Abschnitt wird auf die prototypische Umsetzung der in Abschnitt 5.4 beschriebenen konzeptionellen Architektur zur dynamischen Prozessoptimierung eingegangen. Dazu wird zunächst deren konkrete Abbildung auf eine Software- bzw. System-technische Architektur erläutert. Im Anschluss daran werden die entwickelten Werkzeugkomponenten näher beschrieben. Dabei handelt es sich zum einen um eine Modelleditorkomponente und zum anderen um eine Experimentierkomponente.

5.6.1 Überblick über die gewählte Systemarchitektur

Wie in Bild 5.6.1 dargestellt, wurde zur Realisierung eine moderne 3-schichtige Client/Server-Architektur gewählt, die aus einer Präsentations-, Fachlogik- und Datenhaltungsschicht besteht. Über die Präsentationsschicht hat der Benutzer die Möglichkeit, mit Hilfe eines Java-Applets, das in einem Web-Browser wie beispielsweise Netscape oder Internet Explorer ausgeführt wird, auf die einzelnen Werkzeugkomponenten zuzugreifen. Dies ermöglicht den Einsatz des Systems in einer räumlich verteilten Umgebung und unterstützt damit eine effiziente Zusammenarbeit der beteiligten Personen.

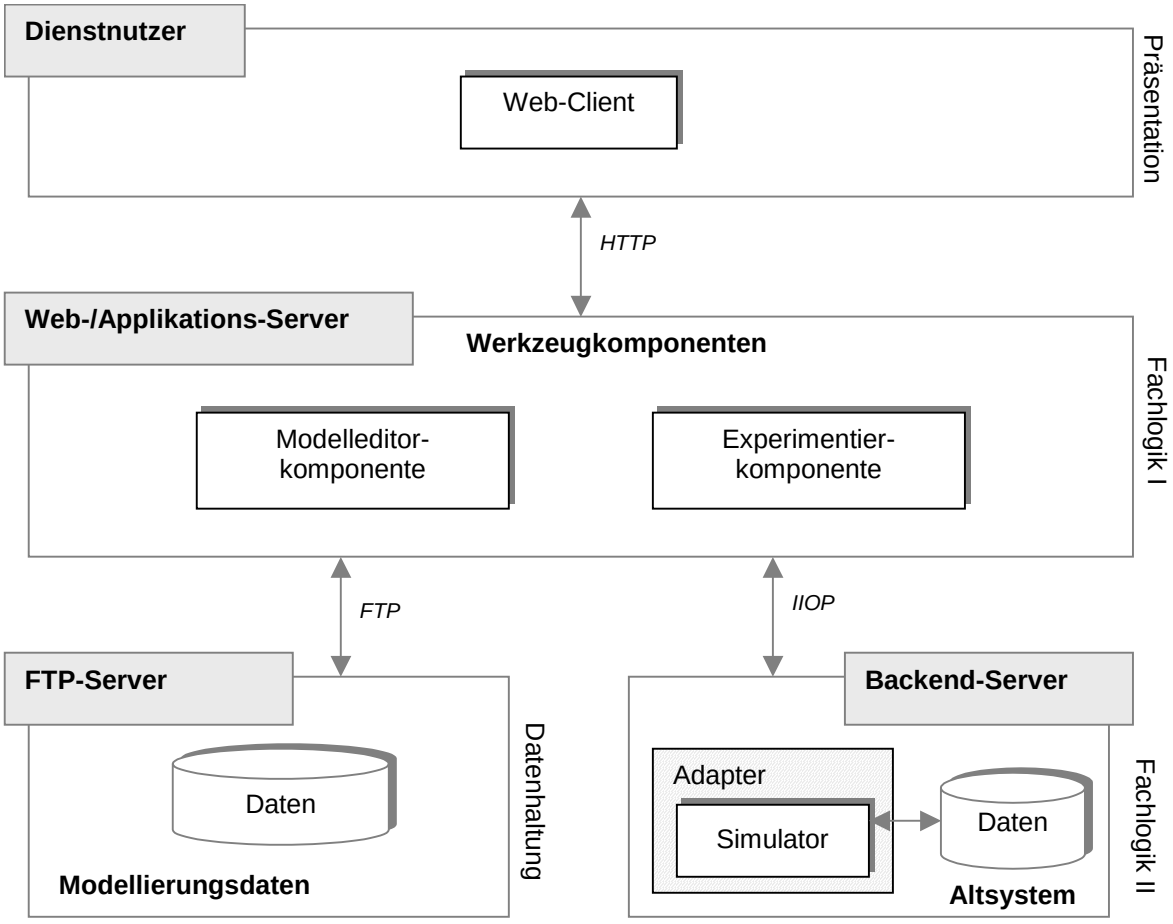


Bild 5.6.1 Mögliche Software- bzw. System-technische Realisierung der dynamischen Prozessoptimierung

Der Web-/Applikations-Server bietet eine Modelleditor- und eine Experimentierkomponente an. Die Experimentierkomponente stellt unterschiedliche Möglichkeiten zum Experimentieren mit Modellen zur Verfügung. Bei den implementierten Experimentierstrategien handelt es sich um die in Abschnitt 2.2 beschriebenen direkten Optimierungsverfahren. Der Modelleditor bietet als Funktionalitäten den Import und Export von XML-Dateien von bzw. zum FTP-Server, verschiedene Modelltransformationen und Präsentationsmöglichkeiten sowie die Möglichkeit zur Spezifikation einer Zielfunktion an.

Zur Modelloptimierung werden neben den Experimentierstrategien auch Algorithmen zur Modellauswertung benötigt, die auf einem Backend-Server lokalisiert sind. Dabei handelt es sich um Methoden zur simulativen und analytischen Modellauswertung. Zu diesem Zweck wurde auf existierende SPN-Auswertungsmodule aus den Werkzeugen TimeNet, GreatSPN und DSPNexpress zurückgegriffen. Die Datenbasis des Backend-Servers enthält zum einen Petrinetzmodelle und zum anderen Ergebnisse der Modellauswertung.

Die Implementierung der Client/Server-Architektur basiert hauptsächlich auf den folgenden drei Technologien: Java, XML und CORBA. Die graphische Benutzungsschnittstelle wurde mit Hilfe der Java Swing Klassen /Gear-99/ realisiert, da Java eine objektorientierte Realisierung, Plattformunabhängigkeit, Client/Server-Interoperabilität und Erreichbarkeit über das World Wide Web ermöglicht. Ein weiterer Vorteil von Java besteht darin, dass Java momentan die besten Voraussetzungen für den Umgang mit XML-Dokumenten bietet. Zur XML-Dateiverarbeitung in Zusammenhang mit Java wurde der XML-Parser Aelfred von Microstar Software Ltd. verwendet. Dieser Parser basiert auf SAX. Um Server- und Backend-Server-Komponenten miteinander zu verbinden, wurde der Middleware-Standard CORBA, aufgrund der in Abschnitt 3.4 beschriebenen Vorteile, eingesetzt.

5.6.2 Der Modell-Editor

In diesem Abschnitt wird die Modell-Editor Komponente erläutert. Dazu wird zuerst die graphische Benutzungsschnittstelle beschrieben. Im Anschluss daran wird näher auf die einzelnen Funktionalitäten des Modell-Editors eingegangen.

5.6.2.1 Die graphische Benutzungsschnittstelle

Wie in Bild 5.6.2 dargestellt, besteht die graphische Benutzungsschnittstelle des Modell-Editors aus den folgenden Teilen:

- einer Menüleiste mit den Menüs „File“, „Convert“, „View“, „Options“ und „Help“.
- einem Bereich mit den Feldern: „xmlImport“, „xmlExport“ und „xmlFiles“.
- einem Textfeld, um die XML-Datei bzw. das daraus generierte Petrinetz anzeigen zu können.

Das Menü „File“ besitzt die Menü-Punkte „Import“ und „Export“ und ermöglicht damit sowohl den Import als auch den Export von XML-Dateien von bzw. zum FTP-Server. Im Menü „Convert“ kann ausgewählt werden, welche Konvertierung gewünscht ist. Zur Verfügung stehen die folgenden Möglichkeiten: „to XML“, „to TimeNET“, „to DSPNexpress“ und „to

GreatSPN“. Wird eine Konvertierungsmöglichkeit ausgewählt, erscheint ein Dialog-Fenster, in dem der Name der Ergebnis-Datei angegeben werden kann. Danach kann mit der Konvertierung begonnen werden. Im Textfeld erscheint daraufhin die erzeugte Datei. Falls im Menü „Options“ der Menü-Punkt „show Report“ selektiert ist, wird zusätzlich ein Konvertierungsbericht erzeugt.

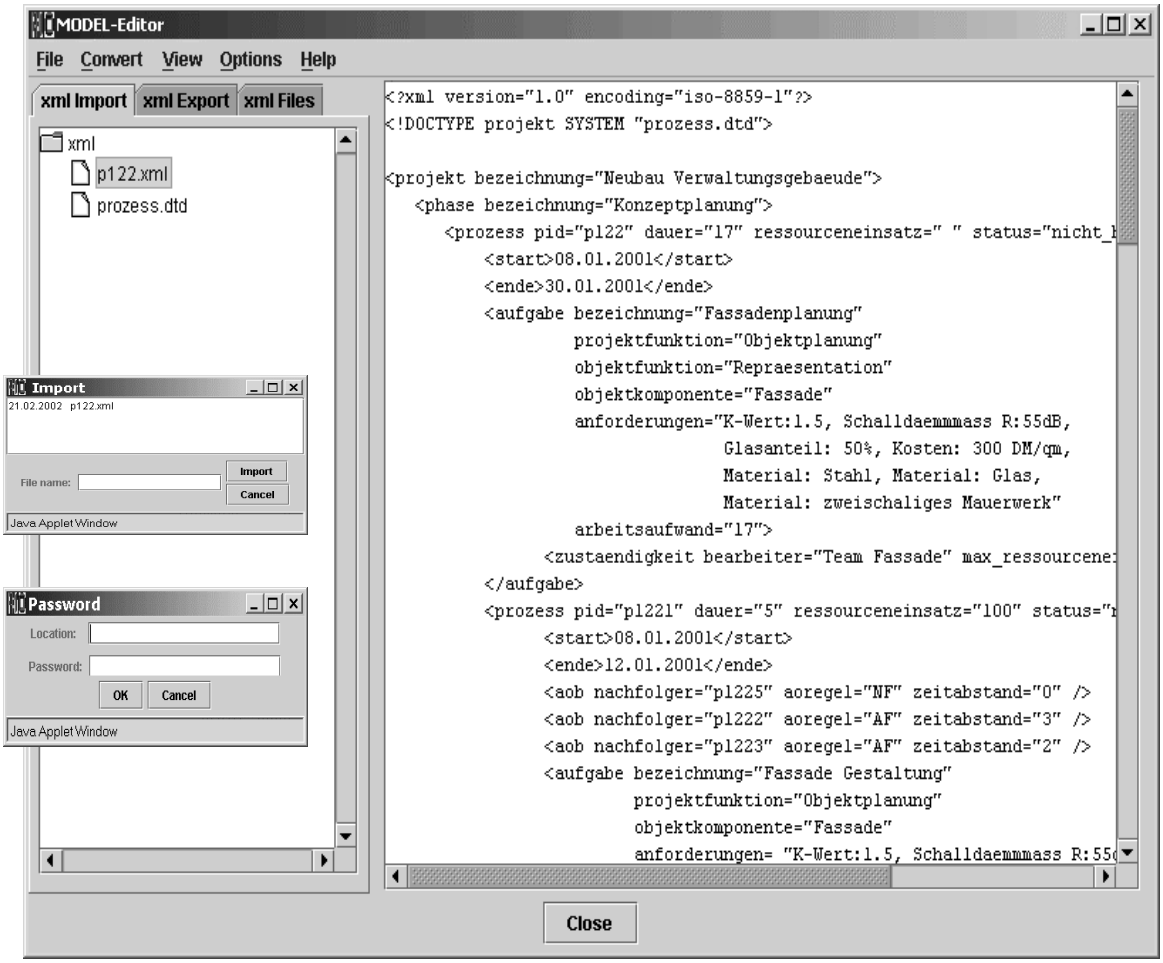


Bild 5.6.2 Die graphische Oberfläche des Modell-Editors

Das Menü „View“ bezieht sich auf die Visualisierung eines stochastischen Petrinetzes. Der erste Menü-Punkt „XML to SVG“ ist aktiv, wenn eine XML-Datei selektiert wurde, und ermöglicht eine Transformation von einer XML-Datei in eine SVG-Datei. Dadurch werden zwei weitere Menü-Punkte aktiv: „SVG to JPEG“ und „SVG Viewer“. Ersterer ermöglicht eine Transformation einer SVG-Datei in eine JPEG-Datei. Letzterer führt dazu, dass ein neues Fenster geöffnet wird, in dem die SVG-Graphik angezeigt wird. Das Menü „Options“ enthält neben dem bereits erwähnten Menü-Punkt „show Report“ den Menü-Punkt „Goal Function“, mit dem der Benutzer eine Zielfunktion, wie in Abschnitt 2.2 beschrieben, spezifizieren kann.

5.6.2.2 Import und Export von XML-Dateien

Jede XML-Datei eines in Abschnitt 5.2 beschriebenen Rahmenprozesses basiert auf der in Bild 5.6.3 dargestellten DTD. Jeder Prozess wird durch eine Projektphase und seine Bezie-

hungen beschrieben. Er besitzt einen definierten Start und ein definiertes Ende, Anordnungsbeziehungen, eine eindeutige Zuordnung zu einem Sachkontext sowie Verknüpfungen zu Informationsobjekten. Des Weiteren werden die Dauer, der Ressourceneinsatz und der Status berücksichtigt.

```
<!ELEMENT projekt (phase+)>
<!ATTLIST projekt
      bezeichnung CDATA #IMPLIED
>
<!ELEMENT phase (prozess+)>
<!ATTLIST phase
      bezeichnung CDATA #IMPLIED
>
<!ELEMENT prozess (start?, ende?, aob*, aufgabe, informationsobjekt*,
prozess*)>
<!ATTLIST prozess
      pid ID #REQUIRED
      dauer CDATA #IMPLIED
      ressourceneinsatz CDATA #IMPLIED
      status
      (nicht_begonnen|in_Vorbereitung|wird_bearbeitet|unterbrochen|
      abgeschlossen) "nicht_begonnen"
>
<!ELEMENT start (#PCDATA)>
<!ELEMENT ende (#PCDATA)>
<!ELEMENT aufgabe (zustaendigkeit)>
<!ATTLIST aufgabe
      bezeichnung CDATA #IMPLIED
      projektfunktion CDATA #IMPLIED
      objektfunktion CDATA #IMPLIED
      objektkomponente CDATA #IMPLIED
      anforderungen CDATA #IMPLIED
      arbeitsaufwand CDATA #REQUIRED
>
<!ELEMENT zustaendigkeit EMPTY>
<!ATTLIST zustaendigkeit
      bearbeiter CDATA #IMPLIED
      max_ressourceneinsatz CDATA #IMPLIED
>
<!ELEMENT informationsobjekt EMPTY>
<!ATTLIST informationsobjekt
      id ID #IMPLIED
      bezeichnung CDATA #IMPLIED
      url CDATA #IMPLIED
>
<!ELEMENT aob EMPTY>
<!ATTLIST aob
      nachfolger IDREF #REQUIRED
      aoregel (NF|AF|EF) "NF"
      zeitabstand CDATA #REQUIRED
>
```

Bild 5.6.3 Die DTD der Rahmenprozesse

Die Import- und Export-Funktionalität wird im Modell-Editor unter dem Menü „File“ angeboten. Für den XML-basierten Datenaustausch dient ein gemeinsamer FTP-Server. Die Import-Funktionalität ermöglicht dem Benutzer, ein zu optimierendes Prozessmodell vom FTP-

Server zu laden. Die Export-Funktionalität erlaubt es, eine XML-Datei auf den FTP-Server zu legen. Diese enthält das Ergebnis der Modelloptimierung. Der Export-Vorgang erfolgt analog zum Import-Vorgang. Der Benutzer wird über ein Passwort-Fenster aufgefordert, sich zu authentifizieren. Nachdem sein Passwort akzeptiert wurde, hat er die Möglichkeit, eine XML-Datei auszuwählen und auf den FTP-Server zu legen.

5.6.2.3 Erzeugen von Petrinetz-Dateien

Ausgangspunkt für eine Transformation in verschiedene proprietäre Petrinetz-Dateiformate ist das in Abschnitt 5.5 beschriebene XML-basierte Modellaustauschformat für stochastische Petrinetze, dessen zu Grunde liegende DTD in Anhang A1 angefügt ist. Um einem Benutzer eine automatische Transformation von einer XML-basierten Planungsprozessbeschreibung in eine XML-basierte Petrinetzbeschreibung zu ermöglichen, wird neben einer Stilvorlage, die beschreibt, wie ein Dokument in ein anderes XML-Dokument transformiert wird, eine Methode zum Lesen der XML-Datei vorausgesetzt. Das heißt, es werden XML-Bearbeitungsfähigkeiten benötigt. Die XML-Bearbeitung wird von einem XML-Parser durchgeführt, der die SAX-API, DOM-API oder beide implementiert. Die DOM-API ist eine Baum-basierte API, die ein XML-Dokument auf eine Menge von Objekten in einer baumartigen Struktur abbildet, die von der DTD festgelegt wird. Die DOM-API erlaubt dem Programmierer Objekte zu lesen (XML-Knoten auf dem Baum), XML-Knoten zu modifizieren und dem Baum neue XML-Knoten hinzuzufügen. Die Baumstruktur kann navigiert werden, um nach Informationen zu suchen, Informationen zu extrahieren oder Informationen zu ändern. Im Gegensatz dazu arbeiten SAX-Parser schneller, weniger speicheraufwändig und liefern eine einfache effiziente Art und Weise zur Bearbeitung von XML-Dateien. Die Aufgabe, eine XML-Datei zu lesen und daraus Instanzen der entsprechenden Modelltransformationsklassen zu erzeugen, ist für einen SAX-Parser gut geeignet. Der SAX-Parser liest sequentiell durch die XML-Dateien und löst Ereignisse aus, sobald die gewünschten Strukturen, wie beispielsweise der Anfang eines Elements, gefunden wurden. Diese Ereignisse stoßen dann die gewünschte Informationsverarbeitung an.

Bild 5.6.4 zeigt einen Konverter, der eine lose Kopplung von Altsystemen ermöglicht. Dazu wird eine bidirektionale Transformation zwischen dem entwickelten XML-basierten Modellaustauschformat für stochastische Petrinetze und den proprietären Dateiformaten der drei zu integrierenden SPN-Werkzeuge TimeNET, DSPNexpress und GreatSPN angeboten. Jedes dieser drei Werkzeuge unterstützt dabei seine eigene Version von stochastischen Petrinetzen und sieht dazu ein proprietäres Dateiformat vor, das ausschließlich auf die Bedürfnisse der unterstützten SPN-Version abgestimmt ist. Um die Heterogenität bewältigen zu können, welche die funktionelle Vielfalt der betrachteten Werkzeuge mit sich bringt, ist die Ausgabe eines Transformationsberichts vorgesehen, der den Benutzer darüber informiert, ob eine Transformation vollständig durchgeführt werden konnte. Dies ist abhängig davon, ob eine Transformation von einer einfacheren zu einer komplexeren Version stattgefunden hat oder umgekehrt. Im letzteren Fall ist eine Transformation nur mit Datenverlust möglich.

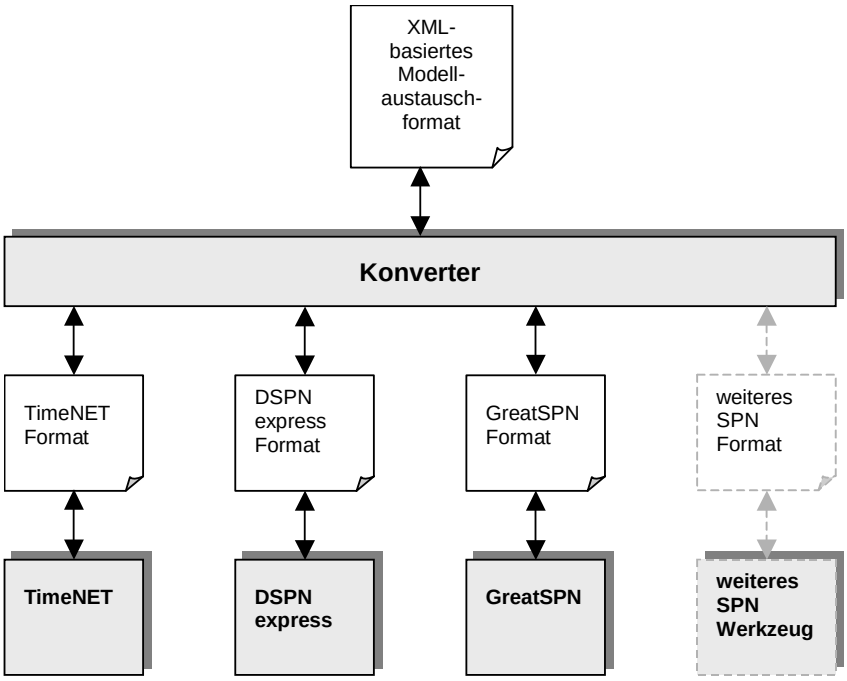


Bild 5.6.4 Transformationen zur Integration von Altsystemen

5.6.2.4 Unterstützung unterschiedlicher Benutzersichten und Endgeräte

Dateiformate zur Unterstützung unterschiedlicher Benutzersichten sind beispielsweise SVG (Scalable Vector Graphics), HTML (Hyper Text Markup Language) und WML (Wireless Markup Language). SVG ist eine Sprache zur Beschreibung zweidimensionaler Graphiken in XML und eignet sich zur graphischen Darstellung von stochastischen Petrinetzen. Für eine Web-Publikation sind sowohl HTML als auch SVG geeignet. WML ermöglicht eine Anbindung an drahtlose Endgeräte durch das Wireless Application Protocol (WAP). Bei WML handelt es sich um eine auf XML basierende Auszeichnungssprache, die für die Spezifikation von Inhalt und Benutzerschnittstelle schmalbandiger Endgeräte wie Mobiltelefone und Mobilfunkempfänger geeignet ist. WML unterstützt Text- und Bild-Darstellungen mit einer Vielfalt an Formatierungs- und Layout-Befehlen.

Die Integration unterschiedlicher Benutzersichten und Endgeräte erfolgt mit Hilfe eines Konverters wie in Bild 5.6.5 dargestellt, der ausgehend vom XML-basierten Modellaustauschformat für stochastische Petrinetze entsprechende Transformationen in die verschiedenen Dateiformate durchführt.

Das XML-basierte Modellaustauschformat enthält keine Information zur Präsentation der Daten. Diese sind in so genannten Stilvorlagen enthalten. Durch Modifikation der Stilvorlagen hat der Benutzer die Möglichkeit, ein Dokument auf unterschiedliche Art und Weise darzustellen. Eine Sprache, um solche Stilvorlagen beschreiben zu können ist der W3C-Standard XSL (eXtensible Stylesheet Language) /W3Co-03b/. Eine Stilvorlage besteht aus einer Menge von Regeln, die spezifizieren, welche Ausgabe erzeugt werden soll, wenn ein Muster im XML-Dokument passt. Wurde beispielsweise eine Stilvorlage für eine HTML-basierte Ausgabe hinsichtlich eines XML-Dokuments spezifiziert, dann erfolgt beim Laden des XML-Dokuments in einen XSL-fähigen Browser eine Transformation in ein HTML-Format.

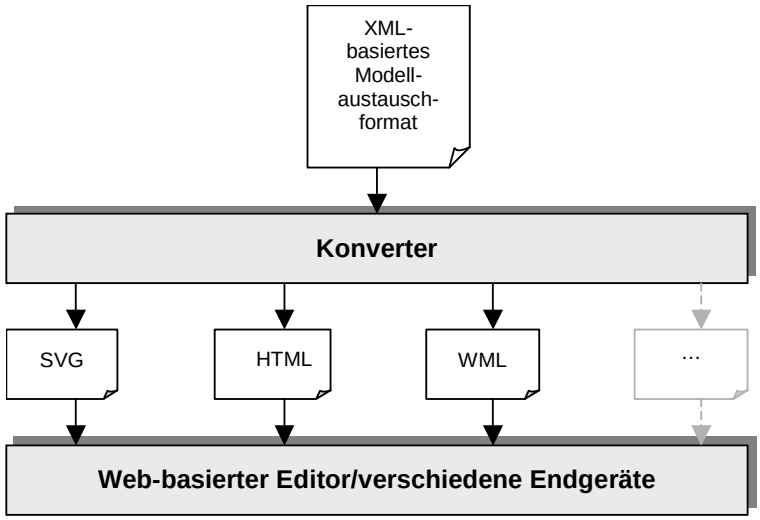


Bild 5.6.5 Transformationen zur Integration unterschiedlicher Benutzersichten und Endgeräte

5.6.2.5 Visualisierung eines Petrinetzmodells mit Hilfe von SVG

Petrinetzmodelle können sowohl textlich als auch graphisch dargestellt werden. Textliche Darstellungen sind beispielsweise ein XML-Format oder ein spezielles Modellbeschreibungsformat. Eine graphische Darstellung hat gegenüber der Textdarstellung den Vorteil, dass sie dem Benutzer eine bessere Übersicht über das entwickelte Modell bietet und eine bessere Diskussionsgrundlage darstellt. Für eine graphische Darstellung eignet sich das SVG-Format. Dieses unterstützt drei verschiedene Graphikarten: Vektorgraphikformen (z.B. Pfade, die aus geraden Linien und Kurven bestehen), Pixelbilder und Text. Graphische Objekte können gruppiert gestaltet, transformiert und aus zuvor geänderten Objekten zusammengesetzt werden. SVG-Zeichnungen können interaktiv und dynamisch sein. Animationen können entweder deklarativ (z.B. durch die Einbindung eines SVG-Animationselements in den SVG-Inhalt) definiert und ausgelöst werden oder über Skripte. Ein Überblick über die laufenden Aktivitäten hinsichtlich SVG ist verfügbar unter /W3Co-03c/.

Elementare Formen für Graphiken sind Rechtecke, Kreise, Ellipsen, Linien, Polylinien und Polygone. Eine graphische Repräsentation eines Petrinetzes /Schi-02/ kann damit einfach aufgebaut werden. Bild 5.6.6 zeigt eine graphische Darstellung des Rahmenprozesses „Fassadenplanung“ aus Abschnitt 5.2 in Form eines Petrinetzmodells, welche mit Hilfe von SVG realisiert wurde. Beispielsweise wird ein Kreis zur Darstellung einer Stelle mit Hilfe von SVG wie folgt beschrieben:

```
<circle    cx=”20” cy=”50” r=”10” fill=”white” stroke=”black”
           stroke-width=”1”/>
```

Dabei geben *cx* und *cy* die Koordinaten für den Mittelpunkt des Kreises und *r* den Radius an. Das Aussehen des Kreises wird mit *fill* für die Füllfarbe, *stroke* für die Strichfarbe und *stroke-width* für die Strichdicke festgelegt.

Zur Darstellung einer Transition wird ein Rechteck wie folgt definiert:

```
<rect  x="80" y="40" width="4" height="20" fill="black" stroke="black"
          stroke-width="1"/>
```

Es werden Angaben zu den Koordinaten der linken oberen Ecke sowie zur Breite und Höhe des Rechtecks gemacht. Analog zum Kreis werden Werte für Füllfarbe, Strichfarbe und Strichdicke festgelegt.

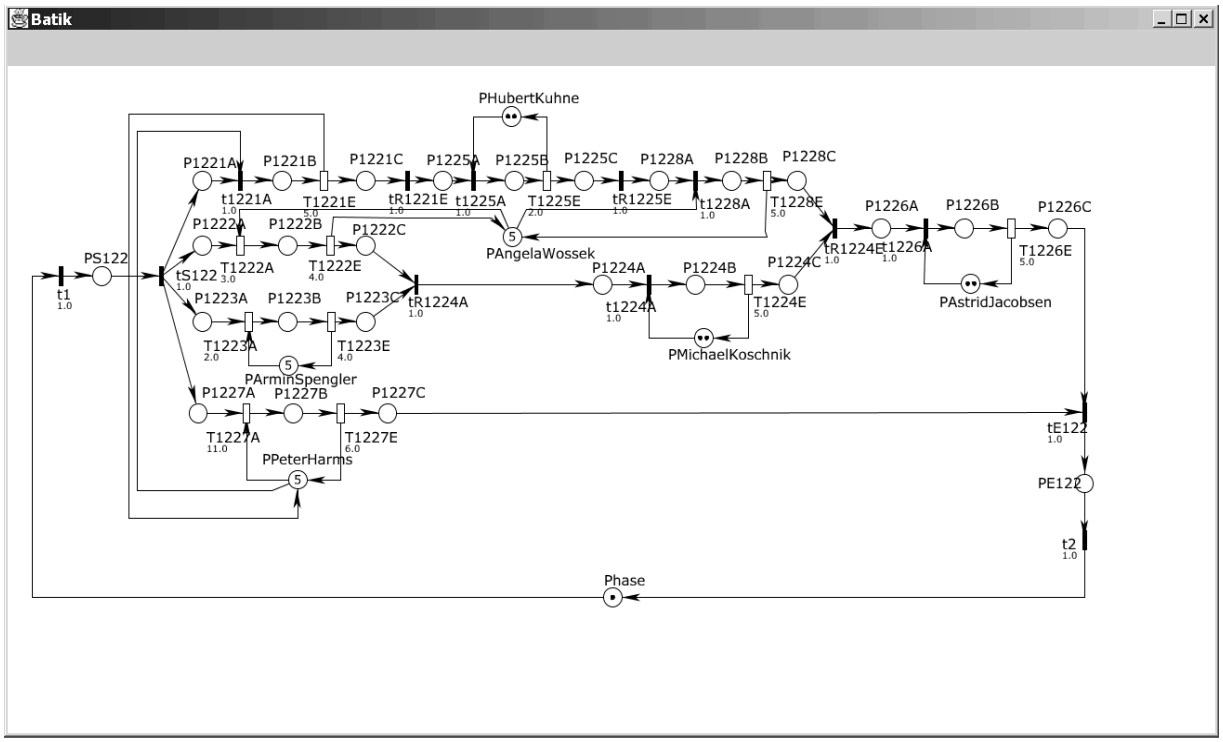


Bild 5.6.6 Visualisierung eines Petrinetzmodells mit SVG

Für die Darstellung einer gerichteten Kante zwischen einer Transition und einer Stelle wird die Definition einer Pfeilspitze benötigt. Mittels eines Markers können in SVG Pfeilspitzen oder auch andere Formen beschrieben werden. Dieser Marker kann dann an den Anfang oder an das Ende einer Linie oder einer Polylinie verankert werden. Die SVG-Beschreibung eines solchen Markers sieht wie folgt aus:

```
<marker    orient="auto" markerHeight="8" markerWidth="20"
       markerUnits="strokeWidth" refY="4" refX="20" id="ArrowHead">
<path  d="M 0 0 L 20 4 L 0 8 L 5 4 z"/>
</marker>
```

Hierbei legt *orient="auto"* fest, dass die Orientierung des Markers an der Polylinie oder Linie ausgerichtet wird. Die Höhe und Breite wird durch *markerHeight* und *markerWidth* festgelegt, *markerUnits="strokeWidth"* gibt an, dass die Einheit des Markers an der Strichbreite der Linie bzw. Polylinie ausgerichtet wird, *refY* und *refX* geben den Ankerpunkt und *id="ArrowHead"* gibt den Identifikator an, mit welcher der Marker referenziert werden kann. Innerhalb des *marker-Tags* wird dann die Form beschrieben, die der Marker haben soll. Hier wird ein Pfad angegeben. Das Attribut *d* enthält die Punktfolge, die den Pfad beschreibt. Da-

bei kennzeichnet *M* den Startpunkt, *L* die Zwischenpunkte und mit *z* wird der Pfad geschlossen.

Der SVG-Code für eine Polylinie, welche die zuvor definierte Pfeilspitze an ihrem Ende verankert, lautet wie folgt:

```
<polyline  fill=”none” stroke=”black” stroke-width=”1”
           points=”30, 50, 80, 50” marker-end=”url(#ArrowHead)”/>
```

Die Angaben zum Aussehen der Polylinie sind analog zum Kreis. In *points* werden die Punkte angegeben, denen die Polylinie folgt und mit *marker-end=“url(#ArrowHead)“* wird definiert, dass am Ende der Polylinie der Marker mit dem Identifikator *ArrowHead* verankert wird.

5.6.2.6 Eingabe einer Zielfunktion

Bild 5.6.7 zeigt das Eingabefenster zur Spezifikation einer zu optimierenden Zielfunktion. Es besteht aus den folgenden Teilen:

- einer Optionsschaltfläche, um zwischen der Optimierung nach Zeit bzw. Ressourcen wählen zu können.
- einer Tabelle, die für jeden Prozess die zu optimierenden Zeit- bzw. Ressourcenparameter enthält. Zu jedem Parameter ist ein Intervall angegeben, innerhalb dessen die gesuchte optimale Parametereinstellung liegen muss. In der letzten Spalte der Tabelle hat der Benutzer die Möglichkeit, die Parameter, nach denen er optimieren möchte, zu selektieren.
- einem Textfeld, in dem der Benutzer die Berechnungsvorschrift für den Zielfunktionswert eingeben muss.
- einer Optionsschaltfläche, um zwischen einem Maximierungs- und einem Minimierungsproblem unterscheiden zu können.

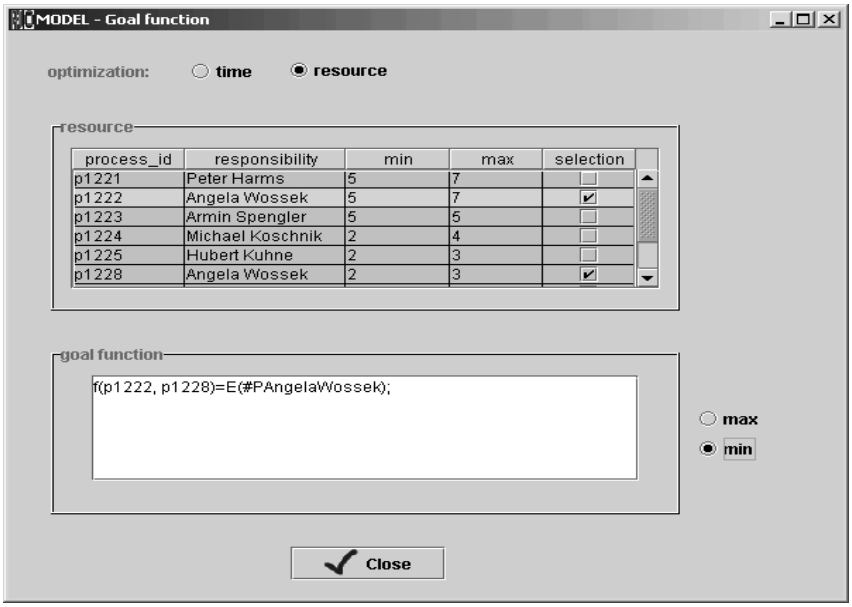


Bild 5.6.7 Spezifikation einer Zielfunktion

5.6.3 Die Experimentierkomponente

Dieser Abschnitt beschreibt die prototypische Realisierung der Experimentierkomponente, die zur Modelloptimierung eingesetzt werden kann. Dazu wird ausgehend von der graphischen Benutzungsschnittstelle auf die verschiedenen Optimierungsstrategien und entsprechenden Kontrollparameter eingegangen. Des Weiteren wird, um dem Anwender die Auswahl einer geeigneten Optimierungsstrategie zu erleichtern, ein Assistenzsystem vorgestellt, das ihn bei der Bearbeitung seiner Optimierungsaufgaben unterstützt.

5.6.3.1 Die graphische Benutzungsschnittstelle

Wie in Bild 5.6.8 dargestellt, besteht das Hauptfenster der graphischen Benutzungsschnittstelle /Somm-99/ aus den folgenden Teilen:

- einer Menüleiste mit den Menüs „File“, „Options“, „Model“ und „Help“.
- einem Auswahlkasten, um eine Optimierungsstrategie (GA, HC, GA&HC) auswählen zu können.
- einer Optionsschaltfläche, um zwischen einer Modell-basierten und einer vordefinierten mathematischen Zielfunktion auswählen zu können.
- einem Fortschrittsbalken, um den Fortschritt des laufenden Optimierungsprozesses anzeigen zu können.
- einem Textfeld, um die während des Optimierungsprozesses berechneten Daten anzeigen zu können.
- einem Kontrollteil, um den Optimierungsprozess starten, unterbrechen und anhalten zu können.
- einer „Change parameters“-Taste, um die Kontrollparameter der gewählten Optimierungsstrategie ändern zu können.

Das Menü „File“ ermöglicht dem Benutzer die spezifizierten Parametereinstellungen zu speichern bzw. zu laden sowie das Beenden der Anwendung. Das Menü „Options“ erlaubt dem Benutzer, den Anfangswert für den Pseudozufallszahlengenerator anzugeben sowie den gewünschten Protokollumfang des angezeigten Optimierungslaufs zu spezifizieren.

Bild 5.6.8 Die graphische Oberfläche der Experimentierkomponente

5.6.3.2 Assistenz bei der Selektion einer geeigneten Optimierungsstrategie

Entscheidend für die Benutzerfreundlichkeit der Experimentierkomponente ist ein geeigneter Entwurf der graphischen Benutzungsschnittstelle. Die oft schwierige Auswahl einer Optimierungsstrategie wird durch ein Assistenzsystem vereinfacht, das dem Benutzer einige Fragen über das Optimierungsproblem stellt. Basierend auf den gegebenen Antworten schlägt das Assistenzsystem dem Benutzer eine oder mehrere geeignete Optimierungsmethoden vor. Solch ein Benutzerinterview kann wie folgt aussehen: Zuerst fragt das Assistenzsystem, ob a priori Wissen über das Optimierungsproblem verfügbar ist. Um die richtige Optimierungsstrategie auswählen zu können, ist vorhandenes Wissen über die Anzahl der Extremstellen der zu optimierenden Zielfunktion oder über charakteristische Merkmale der Zielfunktionsoberfläche sehr hilfreich. Ist beispielsweise bekannt, dass es sich um eine unimodale Zielfunktion handelt, das heißt eine Zielfunktion mit nur einer Extremstelle, dann können alle globalen Optimierungsstrategien ausgeschlossen und ein einfacher Hill-Climbing Algorithmus angewandt werden. Wenn eine multimodale Zielfunktion erwartet wird oder überhaupt kein Wissen über das gegebene Optimierungsproblem vorliegt, dann lautet die nächste Frage des Assistenzsystems, wie viele Extremstellen lokalisiert werden sollen. Falls nur nach einer Extremstelle gesucht wird, schlägt das Assistenzsystem eine globale Optimierungsstrategie vor. Um herauszufinden welche Strategie, fragt das Assistenzsystem nach der erwarteten Optimierungsqualität, dem gewünschten Optimierungserfolg und der zur Verfügung stehenden Rechenzeit. Falls nur eine schnelle und grobe Approximation einer Extremstelle benötigt wird,

schlägt das Assistenzsystem den Genetischen Algorithmus vor. Im Falle einer ε-genauen Lokalisation wäre eine kombinierte 2-Phasen-Strategie die beste Wahl. Ist der Benutzer nicht nur an einer, sondern an einer Folge von verschiedenen Extremstellen interessiert, dann schlägt das Assistenzsystem eine mehrstufige Optimierungsstrategie vor. Zusammenfassend hängt die Wahl einer geeigneten Optimierungsstrategie also von folgenden Benutzerinformationen ab:

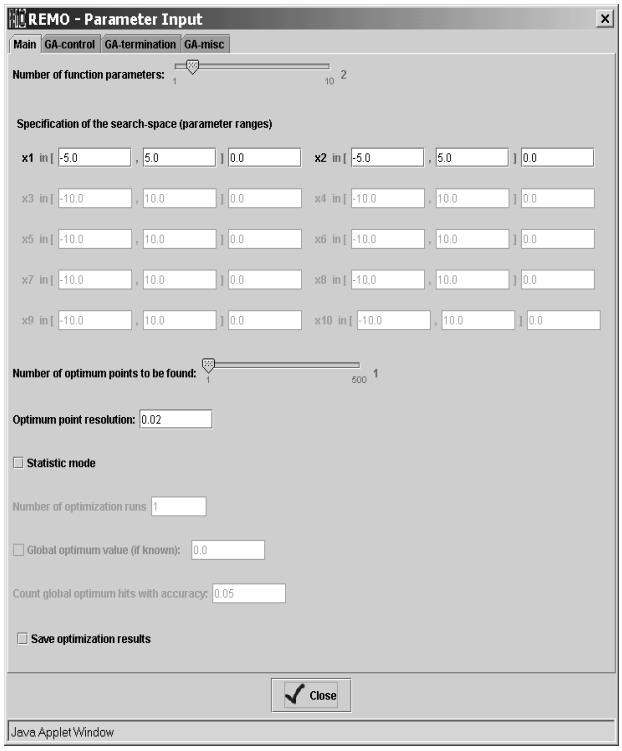
- A priori Wissen über die Zielfunktion
  - Anzahl existierender Extremstellen
  - Zielfunktionsoberflächeneigenschaften
- Eigenschaften des Suchraums
  - Dimension des Problems (Anzahl der Zielfunktionsparameter)
  - Art der zu optimierenden Parameter (ausschließlich diskret, ausschließlich kontinuierlich, gemischt)
- Optimierungsziele
  - Fortschritt, im Vergleich zu einer bereits bekannten Lösung
  - Auffinden einer lokal-optimalen Lösung
  - Auffinden einer global-optimalen Lösung
  - Ermittlung der markantesten Extremstellen eines Optimierungsproblems
- Randbedingungen
  - Maximale Höhe des Optimierungsaufwandes (Anzahl der Modellauswertungen)
  - Zeitbeschränkungen
  - Genauigkeitsanforderungen

5.6.3.3 Festlegung der Kontrollparameter

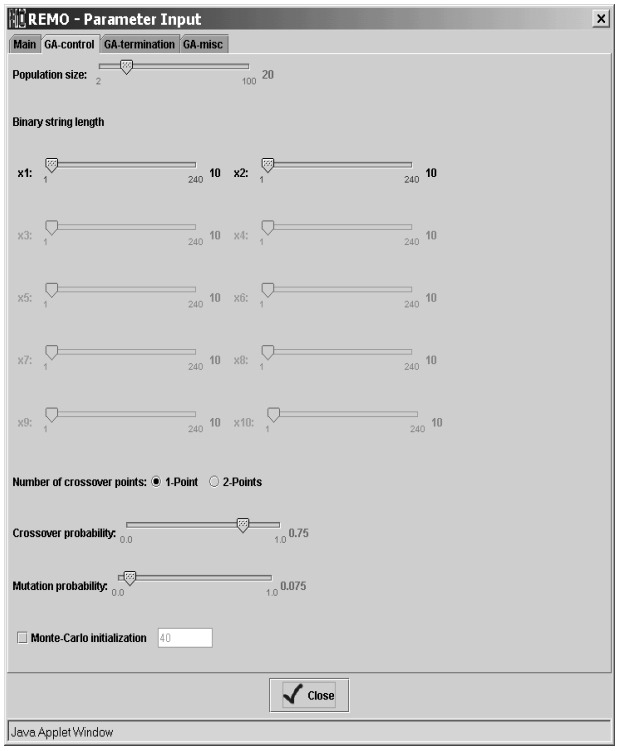
Um Fehler bei der Bedienung zu reduzieren, wird für jede Optimierungsstrategie eine gebräuchliche Voreinstellung der Kontrollparameter angeboten. Erfahrene Benutzer haben die Möglichkeit, die Parameter individuell einzustellen. Im Folgenden werden exemplarisch einige Fenster zur Parameterspezifikation vorgestellt.

Bild 5.6.9 zeigt links oben das Hauptfenster zur Spezifikation der Parameter. Es erlaubt dem Benutzer neben der Spezifikation der Dimension des gegebenen Optimierungsproblems (Anzahl der Zielfunktionsparameter), eine Angabe über die Anzahl der gesuchten Extremstellen und über die gewünschte Genauigkeit zu machen. Darüber hinaus wird ein Statistik-Modus zur Leistungsbewertung der implementierten Optimierungsverfahren und die Möglichkeit zur Ergebnisspeicherung angeboten. Des Weiteren können die Kontrollparameter der gewählten Optimierungsstrategie (in diesem Beispiel der Genetische Algorithmus) über folgende Bedienungsfelder eingestellt werden: „GA-control“, „GA-termination“ und „GA-misc“. Bild 5.6.9 zeigt rechts oben das Fenster zur Spezifikation der GA-Kontrollparameter. Dabei handelt es sich um Angaben über die Populationsgröße, die Länge der Binärvektoren, die Anzahl der Crossover-Punkte sowie die Crossover- und Mutations-Wahrscheinlichkeit.

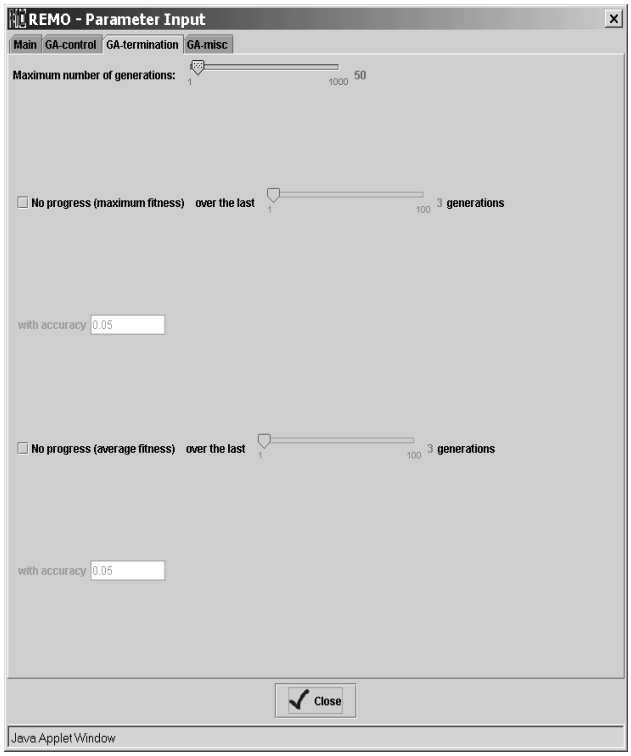
Die Startpopulation des Genetischen Algorithmus kann durch eine parametrisierbare Monte-Carlo Initialisierung festgelegt werden.



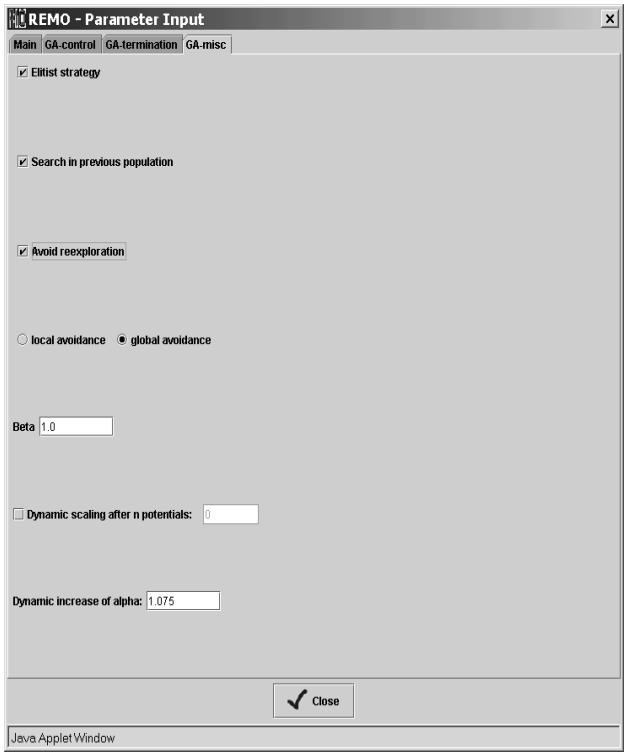
Hauptfenster zur Parameterspezifikation



GA-Kontrollparameter



GA-Terminierungsparameter



Maßnahmen zur Reduzierung des Optimierungsaufwands

Bild 5.6.9 Spezifikation von Kontrollparametern

Bild 5.6.9 zeigt links unten das Fenster zur Spezifikation der GA-Terminierungsparameter. Der GA terminiert, wenn eine zuvor spezifizierte maximale Anzahl an Generationen erreicht wurde oder wenn kein Fortschritt mehr über eine bestimmte Anzahl an Generationen erzielt wurde. Hierbei kann der Fortschritt anhand der maximalen oder der durchschnittlichen Fitness berechnet werden. Um den Optimierungsaufwand zu reduzieren, werden verschiedene Strategien /Syrj-97/ angeboten, die in Bild 5.6.9 rechts unten dargestellt sind. Dazu zählt die Vermeidung von wiederholter Evaluation bereits ausgewerteter Suchpunkte, die Vermeidung von wiederholtem Auffinden bereits gefundener optimaler Lösungen sowie die Elitist-Strategie.

5.7 Bewertung

Als Basis für eine Bewertung der Komponentenarchitektur werden die in Abschnitt 2.5 aufgestellten Ziele der Arbeit herangezogen. Im Hinblick auf den Aspekt *Offenheit* erfüllt der Komponenten-basierte Architekturansatz die geforderte Flexibilität. Die einzelnen Komponenten lassen sich aufgrund ihrer klar voneinander abgegrenzten Funktionalität getrennt entwickeln, warten und erweitern sowie auf unterschiedliche Rechner verteilen. Die Modell-Editor-Komponente unterstützt den Import/Export von XML-Dateien sowie verschiedene Konvertierungs- und Präsentationsmöglichkeiten. Die Experimentierkomponente bietet, wie in Abschnitt 2.2.4 beschrieben, eine Auswahl verschiedener direkter Optimierungsstrategien an. Darüber hinaus wird, um Fehler bei der Bedienung zu reduzieren und Experimente gezielt planen und vorbereiten zu können, der Benutzer durch ein Assistenzsystem bei der Bearbeitung von Modellierungs- und Optimierungsaufgaben unterstützt.

Eine der wesentlichen Forderungen aus Kapitel 2 besteht in der *Integrationsmöglichkeit bestehender Systeme* in die Modellierungsumgebung, da es zu aufwändig bzw. zu teuer wäre, sämtliche Altsysteme durch Neuimplementierungen zu ersetzen. Im Rahmen dieser Arbeit wurden die Werkzeuge TimeNET /TNET-03/, GreatSPN /GSPN-03/ und DSPNexpress /Lind-03/ integriert. Diese bieten, wie in Abschnitt 2.2.2 beschrieben, analytische und simulative Verfahren zur Analyse von stochastischen Petrinetzen an. Eine einfache Integration wird durch die entwickelten Integrationsschnittstellen basierend auf Dokumentenaustausch bzw. (entfernten) Methodenaufruf ermöglicht. Um den Integrationsaufwand möglichst gering zu halten, wurde ein generisches Modellaustauschformat für stochastische Petrinetze entwickelt. Dies hat den Vorteil, dass der Aufwand an notwendigen Modellformatkonvertierungen mit zunehmender Anzahl an zu integrierenden Altsystemen nicht quadratisch, sondern lediglich linear ansteigt.

Um die Anforderung nach weitestgehender *Verwendung von Standards* zu erfüllen, wurde für den Datenaustausch zwischen den einzelnen Komponenten die Auszeichnungssprache XML eingesetzt. Dabei handelt es sich um einen W3C-Standard, der heute große Bedeutung erlangt hat. Zur Metamodellierung wurden die OMG-Standards UML, MOF und XMI vorgeschlagen. Diese Standards zeichnen sich dadurch aus, dass sie in ihrem jeweiligen Einsatzgebiet weit verbreitet sind und somit auf eine breite Akzeptanz stoßen. Des Weiteren unterstützt das entwickelte System ein orts-, zeit- und plattformunabhängiges Arbeiten, da sämtliche implementierten Komponenten mittels gängiger *Internettechnologien* aufgebaut wurden und somit über das World Wide Web zugänglich sind. Dies stellt einen wichtigen Aspekt im

Rahmen einer interdisziplinären Aufgabenstellung dar. Aufwändige Softwareinstallationen an jedem Arbeitsplatz werden damit vermieden.

Hinsichtlich der Forderung, unterschiedliche *Sichten* auf ein Modell berücksichtigen zu können, erfüllt der entwickelte Konverter die geforderte Funktionalität. Er ermöglicht ausgehend vom XML-basierten Modellaustauschformat für stochastische Petrinetze automatische Transformationen in verschiedene Dateiformate. Zur Beschreibung graphischer Darstellungen von Petrinetzen hat sich in diesem Zusammenhang das Dateiformat SVG aufgrund von Eigenschaften wie Web- und Animationsfähigkeit als besonders geeignet herausgestellt. Somit unterstützt das System eine unterschiedliche Aufbereitung und Präsentation von Modellierungsdaten.

Insgesamt konnte durch den Einsatz der entwickelten Komponentenarchitektur im Forschungsschwerpunkt Informationslogistik eine durchgängige Modellierung, Simulation und Optimierung von Planungs- und Entwicklungsprozessen erzielt werden. Durch die weitgehende Automatisierung der einzelnen Arbeitsschritte kann zum einen schneller auf Änderungen des Prozessmodells reagiert werden. Zum anderen können Fehler bei den notwendigen Modelltransformationen vermieden werden. Betrachtet man die in Abschnitt 2.5 vorgestellten Ziele der Arbeit, so lässt sich insgesamt feststellen, dass alle Anforderungen an das zu entwickelnde System berücksichtigt und erfüllt wurden.

6 Zusammenfassung

In dieser Arbeit wurde eine Architektur zur Integration heterogener Komponenten aus den Bereichen Modellierung, Simulation und Optimierung entworfen und realisiert. Diese Bereiche haben in den letzten Jahren aufgrund der stark angestiegenen Komplexität und Heterogenität technischer Systeme große Bedeutung erlangt. Im Bereich der Modellierung und Simulation werden heute in zunehmendem Maße Web- und die damit eng verbundenen Komponententechnologien eingesetzt. Dieser Trend spiegelt sich unter anderem in zahlreichen zu diesem Thema stattfindenden Konferenzen wider /FiHS-98/, /BrUP-99/, /SiBl-00/, /SiWM-01/, /SiWi-02/. Beispielsweise wird die eXtensible Markup Language (XML), welche vom World Wide Web Consortium (W3C) im Februar 1998 verabschiedet wurde, häufig zur Speicherung von Dokumenten und für den Datenaustausch verwendet /Brad-98/. Für Online-Simulationen und -Animationen sowie zur dreidimensionalen Visualisierung von simulierten Abläufen werden mehr und mehr Java-basierte Simulationspakete bzw. Web3D-Technologien entwickelt und eingesetzt. Komponenten-orientierte Techniken sollen primär zu einer Vereinfachung der Entwicklung, Wartung, Erweiterung und Verteilung von Simulationssoftware führen. Die grundlegende Idee hinter diesem Ansatz besteht darin, große Softwaresysteme aus genormten Einzelkomponenten baukastenartig zusammenzusetzen. Komponententechnologien werden prinzipiell auf unterschiedlichen Abstraktionsebenen eingesetzt, wobei die Art der Komponenten auf den jeweiligen Ebenen sehr unterschiedlich ausfallen kann. Während in der Vergangenheit der Fokus hauptsächlich auf den unteren Ebenen lag, auf denen Komponenten im Wesentlichen als ausführbare Codemodule angesehen werden, setzen sich heute Softwareentwickler zunehmend mit der abstrakten Softwarearchitektur-Ebene auseinander, da dies eine klare Trennung von architektur- und implementierungsspezifischen Aspekten erlaubt.

Aus dem Bereich der verteilten Komponenten-orientierten Modellierung und Simulation wurden in dieser Arbeit zwei unterschiedliche Aspekte untersucht: zum einen eine Verteilung von Simulationsmodellen und zum anderen eine Verteilung der Funktionalität von Modellierungswerkzeugen. Beide Aspekte erfordern jeweils eine geeignete Infrastruktur zur Kopplung entsprechender Komponenten. Für den ersten Fall eignet sich die vom US-amerikanischen Verteidigungsministerium entwickelte High Level Architecture (HLA), welche Infrastrukturdienste zum Aufbau und Ablauf verteilter Simulationsanwendungen zur Verfügung stellt. Im zweiten Fall wurde dagegen ein Mangel an geeigneten Integrationskonzepten festgestellt. Eine Analyse klassischer Modellierungswerkzeuge ergab, dass jedoch ein großer Bedarf an solchen Integrationskonzepten vorhanden ist, da zur Planung und Entwicklung komplexer Systeme in der Regel unterschiedliche Modellierungstechniken und Werkzeuge verwendet werden.

Zu den klassischen Modellierungstechniken für nebenläufige und verteilte Systeme zählen heute Petrinetze. Inzwischen existiert zu dieser Modellierungstechnik eine reichhaltige Auswahl an Werkzeugen für die verschiedensten Petrinetzarten, die eine Spezifikation, Simulation und Analyse von Petrinetzen ermöglichen. Wesentliche Schwachstellen dieser Werkzeuge liegen in ihrem monolithischen Softwaredesign, in ihren vielfältigen Modellbeschreibungsformaten sowie in ihrer mangelnden Interoperabilität, ihrer fehlenden Web-Fähigkeit und den eingeschränkten Präsentationsmöglichkeiten.

Um diesen sehr unbefriedigenden Zustand zu verbessern, verfolgt das in dieser Arbeit entwickelte Lösungskonzept einen Komponenten-basierten Ansatz, der darauf basiert, die Funktionalität von Modellierungswerkzeugen in unabhängig voneinander arbeitende Softwarekomponenten zu zerlegen, um sie je nach Anforderungen individuell zu einer so genannten Modellierungsumgebung zusammenstellen zu können. Der Vorteil dieser Vorgehensweise liegt darin, dass sich in solch einer offenen Umgebung Komponenten leichter hinzufügen, entfernen und modifizieren lassen. Zur Integration dieser heterogenen Komponenten wurde eine Architektur entwickelt, welche die einzelnen Komponenten sowie ihre Aufgaben und ihr Zusammenwirken über eine verteilte Kommunikationsinfrastruktur beschreibt. Bei den spezifizierten Komponenten handelt es sich im Einzelnen um Modell-Editoren, Komponenten zur Modellauswertung sowie um Spezialkomponenten zum zielgerichteten Experimentieren mit Modellen. Darüber hinaus werden auch externe Systeme wie beispielsweise Alt-Werkzeuge, Dateiverwaltungs- bzw. Datenbanksysteme sowie verschiedene Arten von Endgeräten für unterschiedliche Benutzergruppen berücksichtigt. Weitere Schwerpunkte dieser Arbeit liegen in der Verwendung von Standards zur Spezifikation generischer Modellaustauschformate, der Berücksichtigung unterschiedlicher Benutzersichten sowie der Verwendung von Web-Technologien.

Im Einzelnen wurden bei der Entwicklung der Komponentenarchitektur folgende Arbeiten durchgeführt. Zunächst wurden die wesentlichen in der Fachdomäne der Modellierungswerkzeuge vorkommenden Komponenten herausgearbeitet und deren Interaktionen beschrieben. Im Anschluss daran wurden zur Kopplung der einzelnen Komponenten flexible Schnittstellen entworfen, die auf losen und engen Kopplungsmechanismen basieren. Unter loser Kopplung wird der Austausch standardisierter Dokumente zwischen den Komponenten verstanden. Dazu wurde ein generisches Modellaustauschformat für stochastische Petrinetze mit Hilfe der Auszeichnungssprache XML entwickelt. Um bei der Spezifikation eines solchen Formats syntaktische und semantische Unstimmigkeiten vermeiden zu können, wurde ein Metamodell-basierter Ansatz vorgestellt, der es erlaubt, die Syntax und zum Teil auch die Semantik eines Modells formal zu beschreiben. Bei den Metamodellen handelt es sich um Objektmodelle, die mit der Unified Modeling Language (UML) dargestellt werden. Einzelne Objektmodelle können in ein gemeinsames Objektmodell zusammengeführt werden, das als Grundlage für eine integrierte Modellierungsumgebung dient. Dies ermöglicht anwendungsspezifischen Werkzeugen auf Sichten zu arbeiten, die unidirektional mit dem Gesamtmodell gekoppelt sind. Eine Berücksichtigung unterschiedlicher Sichten und ihrer unterschiedlichen Präsentationsmöglichkeiten kommt den Wahrnehmungs- und Konzeptualisierungsmustern verschiedener Anwendergruppen entgegen. Durch Verwendung eines Meta-Metamodells können einzelne Modellierungssprachen integriert werden.

Eine enge Kopplung von Komponenten ermöglicht (entfernte) Prozedur- bzw. Methodenaufrufe, wie sie beispielsweise zur Durchführung einer Modelloptimierung benötigt werden. Dazu eignen sich standardisierte Kommunikationsinfrastrukturen wie beispielsweise CORBA oder Web Services. Zur Integration externer Systeme wurden entsprechende Konverter bzw. Adapter entwickelt.

Um eine möglichst durchgängige Modellierung, Simulation und Optimierung gewährleisten zu können, wurden in dieser Arbeit unterschiedliche Konzepte zur automatischen Modelltransformation vorgestellt. Dabei handelt es sich zunächst um eine automatisierte Gewinnung einer XML-DTD bzw. eines XML-Schemas aus einem UML-Modell, da zur Spezifikation von Domänen-spezifischen Metamodellen die UML verwendet wird. Der Austausch und die Verarbeitung von Modellierungsdaten erfolgt jedoch auf Basis von XML. Ausgehend von einem XML-basierten Modellaustauschformat befasst sich eine weitere Transformationsart mit der automatischen Erzeugung unterschiedlicher Modell- und Präsentationsformate zur einfachen Integration verschiedener Endsysteme. Für die Umsetzung dieser Transformationen wurde ein Konverter auf Basis der Transformationssprache XSLT entwickelt. Die dritte Transformationsart bezieht sich auf eine Integration heterogener Modellierungsformalismen. Dazu wurde am Beispiel von Geschäftsprozessen aufgezeigt, wie mit Hilfe eines Modul-basierten Ansatzes ein Geschäftsprozessmodell in ein Petrinetzmodell überführt werden kann.

Die Validation der entwickelten Konzepte erfolgte im Rahmen des Forschungsschwerpunkts Informationslogistik anhand eines komplexen Beispielszenarios aus dem Bereich der Ingenieurwissenschaften mit der Zielsetzung, komplexe Planungsprozesse interdisziplinärer Kooperationen durch den Einsatz von Methoden aus der Modellierung, Simulation und Optimierung wirkungsvoll zu unterstützen. Dazu wurden Prozessmodelle entworfen, implementiert, systematisch analysiert und optimiert. Das in dieser Arbeit entwickelte Integrationskonzept ermöglichte dabei eine durchgängige Modellierung, Simulation und Optimierung der spezifizierten Planungsprozesse. Die abschließende Bewertung zeigte, dass sämtliche in Abschnitt 2.5 vorgestellten Anforderungen an die zu entwickelnde Architektur berücksichtigt und erfüllt wurden. Damit leistet die vorliegende Arbeit einen wesentlichen Beitrag zur Integration heterogener Modellierungswerkzeuge.

7 Literatur

/AaKo-89/ Aarts, E.; Korst, J.: Simulated Annealing and Boltzmann Machines; John Wiley & Sons, 1989

/Andr-03/ Andresen, A.: Komponentenbasierte Softwareentwicklung mit MDA, UML und XML; Carl Hanser Verlag, 2003

/Baum-90/ Baumgarten, B.: Petri-Netze Grundlagen und Anwendungen; Wissenschaftsverlag, 1990

/Bäum-98/ Bäumer, D.: Softwarearchitekturen für die rahmenwerkbasierte Konstruktion großer Anwendungssysteme; Dissertation an der Universität Hamburg im Fachbereich Informatik, 1998

/Baus-96/ Bause, F.: Stochastic Petri nets – an introduction to the theory; Vieweg, 1996

/BiAl-97/ Bischoff, J.; Alexander, T.: Data Warehouse – Practical Advice from the Experts; Prentice Hall, 1997

/Brad-98/ Bradley, N.: The XML companion; Addison-Wesley, 1998

/BrUP-99/ 1999 International Conference on Web-based Modeling and Simulation (part of the 1999 SCS Western MultiConference on Computer Simulation WMC'99); Bruzzone, A.G.; Uhrmacher, A.; Page, E.H. (editors); San Francisco, USA, January 17-20, 1999, Society for Computer Simulation (Publisher)

/Deit-03/ Deitel, H.M.: Web Services – A Technical Introduction; Prentice Hall, 2003

/DeJa-95/ Desrochers, A.; Al_Jaar, R.: Applications of Petri Nets in Manufacturing Systems; IEEE PRESS Marketing, 1995

/DIN-69900/ DIN-Deutsches Institut für Normung e.V. – Projektwirtschaft Netzplantechnik Begriffe, Teil 1. Berlin: Beuth Verlag, 1987

/DoOM-03/ Document Object Model (DOM), URL: http://www.w3.org/DOM/, Stand: 2003

/FiHS-98/ 1998 International Conference on Web-Based Modeling and Simulation (part of the 1998 SCS Western MultiConference on Computer Simulation WMC'98); Fishwick, P.A.; Hill, D.R.C.; Smith, R. (editors); San Diego, California, USA, January 11-14, 1998, Society for Computer Simulation (Publisher)

/Gear-99/ Geary, D.: graphic Java™ – Mastering the JFC; Sun Microsystems Press, A Prentice Hall Title, 1999

/Gold-89/ Goldberg, D.E.: Genetic Algorithms in Search, Optimization and Machine Learning; Addison-Wesley, 1989

/GrKl-03/ Grabowski, H.; Klimesch, C. (Hrsg.): Informationslogistik und Prozessmanagement – Bausteine für interdisziplinäre Kooperationen; Logos Verlag Berlin, 2003

/Grif-98/ Griffel, F.: Componentware: Konzepte und Techniken eines Softwareparadigmas; dpunkt-Verlag, 1998

/GrTh-00/ Gruhn, V.; Thiel, A.: Komponentenmodelle – DCOM, JavaBeans, Enterprise JavaBeans, CORBA; Addison-Wesley, 2000

/GSPN-03/ GreatSPN, Graphical Editor and Analyzer for Timed and Stochastic Petri Nets, URL: http://www.di.unito.it/~greatspn, Stand: 2003

/Hafn-98/ Hafner, S.: Industrielle Anwendungen Evolutionärer Algorithmen; R. Oldenbourg Verlag München Wien, 1998

/HiLA-03/ High Level Architecture, URL: https://www.dmso.mil/public/transition/hla/, Stand: 2003

/HoJe-61/ Hooke, R.A.; Jeeves, T.A.: Direct Search Solution for Numerical and Statistical Problems; Journal ACM 8, pp. 212-221, 1961

/JaSi-03/ JavaSim, Department of Computing Science, University of Newcastle upon Tyne, URL: http://javasim.ncl.ac.uk/, Stand: 2003

/Java-03/ Java, URL: http://www.net-lexikon.de/Java.html, Stand: 2003

/JüKW-00a/ Jüngel, M.; Kindler, E.; Weber, M.: Towards a Generic Interchange Format for Petri Nets; in Proceedings of the Meeting on XML/SGML based Interchange Formats for Petri Nets, a satellite event at 21st International Petri Net Conference, Aarhus, Denmark, June 27, 2000, pp. 1-6

/JüKW-00b/ Jüngel, M.; Kindler, E.; Weber, M.: The Petri Net Markup Language; in Proceedings of the 7th Workshop on “Algorithmen und Werkzeuge für Petrinetze”, Fachberichte Informatik, No. 7-2000, Koblenz, Germany, October 2-3, 2000, pp. 47-52

/KaZY-97/ Kaylan, A.R.; Ziya, S.; Yilmaz, O.: Web-based Architecture for Simulation Modeling and Analysis; in Proceedings of the 1997 European Simulation Multiconference, Istanbul, Turkey, June 1-4, 1997

/KuWD-00/ Kuhl, F.; Weatherly, R.; Dahmann, J.: Creating Computer Simulation Systems – An Introduction to the High Level Architecture; Prentice Hall, 2000

/Lind-03/ Lindemann, C.: DSPNexpress-NG (Next Generation), URL: http://www.Dspnexpress.de/, Stand: 2004

/MaBC-86/ Marsan, M., Balbo, G., Conte, G.: Performance Models of Multiprocessor Systems; MIT Press Series in Computer Systems, 1986

/Müll-97/ Müller, B.: Modellierung von Geschäftsprozessen; Studienarbeit am Institut für Rechnerentwurf und Fehlertoleranz, Fakultät für Informatik, Universität Karlsruhe, 1997

/NaWe-03/ Nagl, M.; Westfechtel, B.: Modelle, Werkzeuge und Infrastrukturen zur Unterstützung von Entwicklungsprozessen – Symposium, WILEY-VCH Verlag GmbH & Co. KgaA, Weinheim, 2003

/Ober-96/ Oberweis, A.: Modellierung und Ausführung von Workflows mit Petri-Netzen; Teubner, 1996

/OMGr-04a/ OMG, Object Management Group, URL: http://www.omg.org, Stand: 2004

/OMGr-04b/ OMG-MOF Meta Object Facility, Version 1.4, URL: http://www.omg.org/technology/documents/formal/mof.htm, Stand: 2004

/OMGr-04c/ OMG-UML Unified Modeling Language, URL: http://www.uml.org/, Stand: 2004

/OMGr-04d/ OMG-XML Metadata Interchange (XMI), URL: http://www.omg.org/technology/documents/formal/xmi.htm, Stand: 2004

/OMGr-04e/ OMG, Object Management Group, Catalog Of OMG Modeling And Metadata Specifications, URL: http://www.omg.org/technology/documents/modeling_spec_catalog.htm, Stand: 2004

/OrHa-98/ Orfali, R.; Harkey, D.: Client/Server Programming with JAVA and CORBA; John Wiley & Sons Inc., 1998

/Page-91/ Page, B.: Diskrete Simulation – Eine Einführung mit Modula-2; Springer-Verlag, 1991

/Petr-62/ Petri, C.A.: Kommunikation mit Automaten; Schriften des Instituts für Instrumentelle Mathematik, Bonn, Nr.3, 1962

/PGEE-00/ Padberg, J.; Gajewsky, M.; Ermel, C.; Ehrig, H.: Petrinetze – Modellierung, Strukturierung und Kompositionalität; Skript zur Lehrveranstaltung FB Informatik, TU Berlin, März 2000

/PNMe-00/ Meeting on XML/SGML based Interchange Formats for Petri Nets, URL: http://www.daimi.au.dk/pn2000/Interchange/, Stand: 2004

/PNSt-03/ Petri Nets Standard, URL: http://www.daimi.au.dk/PetriNets/standardisation, Stand: 2003

/PNTS-03/ Petri Nets Tools and Software, URL: http://www.daimi.au.dk/PetriNets/tools, Stand: 2003

/Reis-85/ Reisig, W.: Systementwurf mit Netzen; Springer-Verlag Berlin Heidelberg, 1985

/RuJB-99/ Rumbaugh, J.; Jacobsen, I.; Booch, G.: The Unified Modeling Language Reference Manual; Reading, Massachusetts, Addison-Wesley, 1999

/Rump-99/ Rump, F.: Geschäftsprozeßmanagement auf der Basis ereignisgesteuerter Prozeßketten; Teubner-Reihe Wirtschaftsinformatik, 1999

/Saue-99/ Sauerbier, T.: Theorie und Praxis von Simulationssystemen; Vieweg Verlag, 1999

/Saxp-03/ The simple api for xml, URL: http://www.saxproject.org/, Stand: 2003

/Sche-01/ Scherber, S.: Modellierung und Simulation software-intensiver eingebetteter Systeme; Dissertation, Universität Hannover, Shaker Verlag, 2001

/Schi-02/ Schillinger, S.: Entwicklung eines XML-basierten Austauschformates für Stochastische Petri-Netze; Studienarbeit, Universität Karlsruhe, Fakultät für Informatik, Institut für Rechnerentwurf und Fehlertoleranz, 2002

/Schn-92/ Schnieder, E.: Petrinetze in der Automatisierungstechnik; Oldenbourg-Verlag, 1992

/Schn-03/ Schnieder, E.: Integration heterogener Modellwelten der Automatisierungstechnik; in Modelle, Werkzeuge und Infrastrukturen zur Unterstützung von Entwicklungsprozessen, WILEY-VCH Verlag GmbH & Co. KgaA, Weinheim, 2003

/Schn-97/ Schneider, H.-J.: Lexikon Informatik und Datenverarbeitung, R. Oldenbourg Verlag, 1997

/SeSB-98/ Seibt, F.; Schumann, M.; Beikirch, J.: Concept and Components for a Web-based Simulation Environment (WBSE); in Proceedings of the 1998 International Conference on Web-Based Modeling and Simulation (part of the 1998 SCS Western MultiConference on Computer Simulation WMC'98); San Diego, California, USA, January 11-14, 1998

/ShGa-96/ Shaw, M.; Garlan, D.: Software Architecture – Perspectives on an Emerging Discipline; London, Prentice Hall, 1996

/SiBl-00/ 2000 International Conference on Web-Based Modeling and Simulation (part of the 2000 SCS Western Multi Conference on Computer Simulation WMC'00); Signorile, R.; Blais, C. (editors); San Diego, California, USA, January 23-27, 2000, Society for Computer Simulation (Publisher)

/Silk-03/ Silk, ThreadTec Incorporation, St. Louis, USA, URL: http://www.threadtec.com/, Stand: 2003

/SimJ-03/ Simjava, Institute for Computing Systems Architecture, Division of Informatics, University of Edinburgh, URL: http://www.dcs.ed.ac.uk/home/hase/simjava/, Stand: 2003

/SiWi-02/ 2002 International Conference on Web-Based Modeling and Simulation (part of the 2002 SCS Western Multi Conference on Computer Simulation WMC'02); Signorile, R.; Wilsey, P.A. (editors); San Antonio, Texas, USA, January 27-31, 2002, Society for Computer Simulation (Publisher)

/SiWM-01/ 2001 International Conference on Web-Based Modeling and Simulation (part of the 2001 SCS Western Multi Conference on Computer Simulation WMC'01); Signorile, R.; Wilsey, P.A.; Miller, J.A. (editors); Phoenix, Arizona, USA, January 7-11, 2001, Society for Computer Simulation (Publisher)

/Somm-99/ Sommer, T.: CORBA-basierte Parameteroptimierung von Simulationsmodellen in verteilter Umgebung; Diplomarbeit, Universität Karlsruhe, Institut für Rechnerentwurf und Fehlertoleranz, Fakultät für Informatik, 1999

/SoMa-02/ Song, F.; Mackworth, A.: CNJ – A Visual Programming Environment for Constraint Nets; in Proceedings of the 2002 AI, Simulation and Planning in High Autonomy Systems (AIS 2002), Lisbon, Portugal, April 7-10, 2002

/StCe-94/ Standridge, C.R.; Centeno, M.A.: Concepts for Modular Simulation Environments; in Proceedings of the 1994 Winter Simulation Conference, Lake Buena Vista, FL, USA, December 11-14, 1994, pp. 657-663

/StKl-98/ Straßburger, S.; Klein,U.: Integration des Simulators SLX in die High Level Architecture, Tagung Simulation und Visualisierung, Magdeburg, SCS Europe Publishing House, 1998

/Syrj-97/ Syrjakow, M.: Verfahren zur effizienten Parameteroptimierung von Simulationsmodellen; Dissertation am Institut für Rechnerentwurf und Fehlertoleranz der Universität Karlsruhe, Berichte aus der Informatik, Shaker Verlag, 1997

/Syrj-99/ Syrjakow, E.: Analyse und Steuerung von verteilten Arbeits- und Kommunikationsprozessen in branchenübergreifenden Kooperationen mit formalen Modellierungskonzepten; in Informationslogistik für unternehmens- und branchenübergreifende Kooperation, Abschlussbericht, Universität Karlsruhe, Dezember 1999

/Syrj-02/ Syrjakow, E.: Unterstützung der branchenübergreifenden kooperativen Planung durch Methoden aus der Modellierung, Simulation und Optimierung; in Informationslogistik für die internetbasierte Prozessinteraktion bei der branchenübergreifenden Kooperation, Abschlussbericht, Universität Karlsruhe, September 2002

/Syrj-03/ Syrjakow, E.: Prozesssimulation und –optimierung in der Planung und Durchführung; in Informationslogistik und Prozessmanagement; H. Grabowski, C. Klimesch (Hrsg.); Logos Verlag Berlin (ISBN 3-8325-0182-7), Mai 2003

/SySy-01/ Syrjakow, E.; Syrjakow, M.: Parameter Optimization of Complex Simulation Models; in System Design Automation – Fundamentals, Principles, Methods, Examples; R. Merker, W. Schwarz (eds.); Kluwer Academic Publishers, March 2001, pp. 233-246

/SySS-02/ Syrjakow, M.; Syrjakow, E.; Szczerbicka, H.: Towards a Component-Oriented Design of Modeling and Simulation Tools; Proceedings of the 2002 AI, Simulation and Planning in High Autonomy Systems (AIS 2002), Lisbon, Portugal, April 7-10, 2002, pp. 59-64

/SySy-02/ Syrjakow, E.; Syrjakow, M.: Web-based Business Process Modeling and Optimisation; in Proceedings of the International Conference on Internet and Multimedia Systems and Applications (IMSA 2002); Kauai, Hawaii, USA, August 12-14, 2002, pp. 124-129

/SySy-03/ Syrjakow, E.; Syrjakow, M.: XML for Data Representation in Modeling and Simulation Environments; in Proceedings of the International Conference on Modelling, Simulation, and Optimization (MSO 2003); Banff, Alberta, Canada, July 2-4, 2003, pp. 100-107

/SzUt-00/ Szczerbicka, H.; Uthmann, T. (Hrsg.): Frontiers in Simulation – Modellierung, Simulation und Künstliche Intelligenz; SCS-Europe BVBA, Ghent, Belgium, 2000

/Turo-99/ Turowski, K.: Ordnungsrahmen für komponentenbasierte betriebliche Anwendungssysteme; in Tagungsband 1, Workshop Komponentenbasierte betriebliche Anwendungssysteme (WKBA 1), Otto-von-Guericke-Universität Magdeburg 1999; URL: http://www-wi.cs.uni-magdeburg.de/workshops/komponenten

/Vers-97/ Versteegen, G.: Vorgehensmodell zur Geschäftsprozessmodellierung – Immer der Reihe nach; iX Magazin für professionelle Informationstechnik, Heft 6, 1997

/Wied-99/ Wiedemann, T.: A New Approach for an Open Modeling and Simulation Environment by Using Actual Software Technologies; in Proceedings of the 13th European Simulation Multiconference ESM 99, Warsaw, Poland, June 1-4, 1999

/Wiki-04/ Wikipedia, die freie Enzyklopädie, URL: http://de.wikipedia.org/wiki/, Stand: 2004

/W3Co-03a/ W3C, World Wide Web Consortium, URL: http://www.w3.org, Stand: 2003

/W3Co-03b/ World Wide Web Consortium, The Extensible Stylesheet Language, URL: http://www.w3.org/Style/XSL/, Stand: 2003

/W3Co-03c/ World Wide Web Consortium, Scalable Vector Graphics (SVG) 1.0 Specification, URL: http://www.w3.org/TR/SVG/, Stand: 2003

/Zieg-98/ Ziegler, P.: Strukturbasierte Dekomposition von allgemeinen stochastischen Petrinetzen (GSPN); Dissertation, Universität Karlsruhe, Logos-Verl., 1999

/Zimm-01/ Zimmermann, A.: TimeNET 3.0 User Manual – A Software Tool for the Performability Evaluation with Stochastic Petri Nets, Performance Evaluation Group, TU Berlin, 2001

/Zimm-03/ Zimmermann, A.: TimeNET, URL: http://pdv.cs.tu-berlin.de/~timenet, Stand: 2003

/Züll-98/ Züllighoven, H.: Das objektorientierte Konstruktionshandbuch nach dem Werkzeug-& Material-Ansatz; Heidelberg dpunkt-Verlag, 1998

Bilder- und Tabellenverzeichnis

Anhang

A1 DTD des Modellaustauschformats für SPN

```
<!ELEMENT spn (generalInformation, place*, transition*, label*)>

<!ELEMENT generalInformation (numberOfPlaces, numberOfTransitions,
numberOfMarkingParameters, numberOfDelayParameters)>
<!ATTLIST generalInformation
    netName CDATA #REQUIRED
    netType (GSPN|DSPN|CDSPN|EDSPN) #REQUIRED
    description CDATA #IMPLIED>
<!ELEMENT numberOfPlaces (#PCDATA)>
<!ELEMENT numberOfTransitions (#PCDATA)>
<!ELEMENT numberOfMarkingParameters (#PCDATA)>
<!ELEMENT numberOfDelayParameters (#PCDATA)>

<!ELEMENT place (name, marking, graphics*)>
<!ELEMENT name (#PCDATA)>
<!ELEMENT marking (#PCDATA)>

<!ELEMENT transition (name, delay, graphics*, orientation, arc+)>
<!ATTLIST transition
    enablingDependence (IS|SS) #REQUIRED
    kind (EXP|DET|GEN|IM) #REQUIRED
    firingPolicy (RS|RA|RE) #REQUIRED
    priority CDATA #REQUIRED
    group CDATA #IMPLIED
    groupWeight CDATA #IMPLIED
    phase CDATA #IMPLIED>
<!ELEMENT name (#PCDATA)>
<!ELEMENT delay (#PCDATA)>
<!ELEMENT orientation (#PCDATA)>
<!ELEMENT arc (multiplicity, number, connectedPlace, graphics*)>
<!ATTLIST arc
    kind (inputArc|outputArc|inhibitorArc) #REQUIRED>
<!ELEMENT multiplicity (#PCDATA)>
<!ELEMENT number (#PCDATA)>
```

```
<!ELEMENT connectedPlace (#PCDATA)>

<!ELEMENT label (name, value, graphics*)>
<!ATTLIST label
    kind (markingParameter|delayParameter) #REQUIRED>
<!ELEMENT name (#PCDATA)>
<!ELEMENT value (#PCDATA)>

<!ELEMENT graphics (coordinate+)>
<!ATTLIST graphics
    kind (positionPlace|positionTag|positionTransition|positionDelay)
    #REQUIRED>
<!ELEMENT coordinate EMPTY>
<!ATTLIST coordinate
    x CDATA #REQUIRED
    y CDATA #REQUIRED>
```

A2 Semantik der Klassenattribute des Metamodells für SPN

Klasse	Attribut	Bedeutung
Place	name	Name des Platzes
	marking	Anzahl der Marken des Platzes
Transition	name	Name der Transition
	delay	Schaltverzögerung
	orientation	Orientierung der Transition: (0=vertikal, 1=horizontal)
	enablingDependence	Die Schalttaktik exponentiell verteilter Transitionen bestimmt die Art und Weise, wie mit mehreren „Kunden“ verfahren wird: • IS (infinite server): Transitionen können gleichzeitig schaltbereit sein. • SS (single server): die Schaltzeiten werden sequentiell bestimmt.
	kind	Art der Transition (IM, EXP, DET, GEN)
	firingPolicy	Im Falle nicht exponentiell verteilter Verzögerungen muss eine der folgenden Schalttaktiken spezifiziert werden: • RS (race with resampling): Bei jedem Markierungswechsel benötigt die Transition eine neue Schaltzeit-Stichprobe aus ihrer Verteilung. • RA (race with age memory): Die Transition benötigt eine neue Schaltzeit-Stichprobe nur nachdem sie geschaltet hat. Die Zeit, die von einer Transition verbraucht wurde, während sie geschaltet hat, geht nicht verloren. • RE (race with enabling memory): Die Transition benötigt eine neue Schaltzeit-Stichprobe, nachdem sie geschaltet hat oder wenn sie wieder schaltbereit ist. Die verbrachte Zeit während die Transition schaltbereit war geht verloren.
	priority	Die Priorität definiert die Rangfolge zwischen gleichzeitig aktivierten zeitlosen Transitionen.
	phase	Anzahl der Phasen, um deterministische Verteilungen zu approximieren.
	groupWeight	Das Gewicht gibt die relative Schaltwahrscheinlichkeit einer zeitlosen Transition in Bezug zu anderen gleichzeitig schaltbereiten zeitlosen Transitionen an.
Arc	multiplicity	Multiplizität der Kante
	number	Anzahl der Kanten, die von einer Transition ausgehen
	connectedPlace	Name des mit der Transition verbundenen Platzes
	kind	Art der Kante: • inputArc (Eingangskante) • outputArc (Ausgangskante) • inhibitorArc (Hemmendekante)
GeneralInfo	netName	Name des SPN
	netType	Art des SPN (GSPN, DSPN, CDSPN, EDSPN)
	description	Beschreibung des SPN
	numberOfPlaces	Anzahl der Plätze
	numberOfTransitions	Anzahl der Transitionen
	numberOfDelayParameters	Anzahl der Verzögerungsparameter
	numberOfMarkingParameters	Anzahl der Markierungsparameter
Label	kind	Art der Kennzeichnung (Markierungs- oder Verzögerungsparameter)
	name	Bezeichnung
	value	Wert des Bezeichners
Coordinate	x	x-Koordinate
	y	y-Koordinate

Tabelle A2.1 Semantik der Klassenattribute des Metamodells für SPN

Glossar

API (Application Programming Interface)

Schnittstelle, die ein Betriebssystem oder auch ein anderes Softwaresystem anderen Programmen zur Verfügung stellt. Eine API stellt Routinen, Protokolle und Dienstprogramme für das Erstellen von Software dar.

Applet

Kleines Computerprogramm, das in einem Web-Browser läuft und in der Programmiersprache Java geschrieben ist.

CGI (Common Gateway Interface)

Eine Möglichkeit, um Programme oder Scripts im WWW bereitzustellen, die von HTML-Dateien aus aufgerufen werden können und die selbst HTML-Code erzeugen und an einen Web-Browser senden können.

CORBA (Common Object Request Broker Architecture)

OMG-Standard für verteilte objektorientierte Systeme; alle Schnittstellenbeschreibungen werden im IDL (Interface Definition Language)-Format vorgenommen.

CWM (Common Warehouse Metamodel)

Ein von der OMG entwickelter Standard für die Beschreibung, den Zugriff und den Austausch von im Data-Warehouse-Prozess anfallenden Metadaten. CWM soll Interoperabilität zwischen verschiedenen Data-Warehouse-Systemen und Werkzeugen ermöglichen.

DIN (Deutsches Institut für Normung e.V.)

Nationale Normungsorganisation in Deutschland mit Sitz in Berlin. Der Verein entwickelt in Zusammenarbeit mit Handel, Industrie, Wissenschaft, Verbrauchern und Behörden technische Standards (Normen) zur Rationalisierung und Qualitätssicherung.

DOM (Document Object Model)

DOM ist eine API und beschreibt, wie man programmiersprachenunabhängig auf HTML- oder XML-Dokumente zugreifen kann. Das DOM-Modell besteht aus Knoten, die Dokumente, Elemente und Attribute enthalten, die in einer Baumstruktur durch Zeiger miteinander verbunden sind.

DTD (Document Type Definition)

Formale Spezifikation aller in einem Dokumenttyp erlaubten Strukturen. Man spricht auch von einer formalen Grammatik, in der zulässige Tags, Attribute und deren Verschachtelung definiert sind.

DWH (Data Warehouse)

Datenbanksystem zur Sammlung sämtlicher auswertungsrelevanter Unternehmensinformationen ohne spezielle Zielgruppenausrichtung.

Homonym

Wort, das in verschiedenen Bedeutungen für mehrere Begriffe stehen kann (z.B. Ton als Material oder Ton in der Musik).

HTML (Hypertext Markup Language)

Publikationssprache von Dokumenten für das WWW.

ISO (International Organization for Standardization)

Internationale Vereinigung der Standardisierungsgremien von 148 Ländern. Die ISO verabschiedet internationale Standards in allen technischen Bereichen (außer in Elektrik und Elektronik, für welche die IEC zuständig ist), darunter technische (z.B. MP3 oder Telefonkarten), klassifikatorische (z.B. Ländercodes wie .de, .nl, .jp) und Verfahrensstandards (z.B. Qualitätsmanagement nach ISO 9000). Das Deutsche Institut für Normung e.V. (DIN) ist seit 1951 Mitglied der ISO. Die USA sind durch ANSI (American National Standards Institute) bei der ISO vertreten.

Java

Objektorientierte, plattformunabhängige Programmiersprache. Java-Programme laufen üblicherweise auf einer Java Virtual Machine (JVM), das heißt einer virtuellen Maschine, welche die konkreten Details von Hardware und Betriebssystem für das Programm verbirgt.

Metadaten

Strukturen von Daten und Datenkomplexen.

Metamodellierung

Erstellung und Pflege von Metadaten enthaltenden Modellen.

Middleware

Software, die auf den Einzelrechnern lokal installiert wird und unter Verwendung von Betriebssystemfunktionen die Dienste und Eigenschaften eines verteilten Systems für Anwendungen zur Verfügung stellt. Dies geschieht über eine API, die unabhängig vom Rechnertyp und dem lokalen Betriebssystem ist.

MOF (Meta Object Facility)

Vorgaben der OMG für die Erstellung und Nutzung von Metamodellen.

OCL (Object Constraint Language)

Bestandteil der UML. Sie dient u.a. der textuellen Spezifikation von Invarianten in Klassendiagrammen, von Bedingungen in Sequenzdiagrammen oder der Formulierung von Vor- und Nachbedingungen für Methoden.

OMG (Object Management Group)

Herstellerübergreifendes Konsortium, das es sich zur Aufgabe gesetzt hat, informationstechnische Standards zu schaffen. Federführend u.a. bei CORBA, UML, OCL, XML, XMI, MOF und CWM.

Petrinetze

Mathematische Repräsentation von verteilten Systemen. Petrinetze wurden durch Carl Adam Petri in den 1960er Jahren definiert.

Rahmenwerk (Framework)

Rahmen für die Entwicklung von Modellen, Programmcode oder Datenbasen.

SAX (Simple API for XML)

Bibliothek mit Grundfunktionen zur Interpretation und Auswertung von XML-Daten. Die Bibliothek gibt es für verschiedene Programmiersprachen und Betriebssysteme.

Servlet

Stellt im Rahmen der „Java 2 Platform Enterprise Edition“ (J2EE) das Pendant zu einem CGI-Skript oder anderen Konzepten dar, mit denen Web-Inhalte erstellt werden können. Stellt eine Weiterentwicklung von CGI dar.

SMIL (Synchronized Multimedia Integration Language)

Ein auf XML basierender, vom W3C entwickelter Standard für eine Auszeichnungssprache für zeitsynchronisierte, multimediale Inhalte. SMIL ermöglicht die Einbindung und Steuerung von Multimedia-Elementen wie Audio, Video, Text und Grafik in Webseiten. SMIL-Dateien können mit Java-Applets und -Servlets oder CGI-Skripten verknüpft werden und so beispielsweise auf eine Datenbank zugreifen.

SOAP (Simple Object Access Protocol)

Softwarekommunikationsprotokoll zur Verbindung heterogener Anwendungen.

SVG (Scalable Vector Graphics)

Sprache zur Beschreibung zweidimensionaler Vektorgrafiken in XML. SVG unterstützt mittels SMIL Animationen. SVG wurde im September 2001 zur W3C-Empfehlung erhoben.

UML (Unified Modeling Language)

Etablierte objektorientierte Modellierungs- und Spezifizierungsnotation mit hohem graphischem Anteil.

URI (Uniform Resource Identifier)

Zeichenfolge, die zur Identifizierung einer abstrakten oder physikalischen Ressource dient. URI wird zur Bezeichnung von Ressourcen im WWW eingesetzt.

Web Service

Ein u.a. von Hewlett-Packard, IBM, Microsoft und SAP initiierter Standard für den Datenaustausch zwischen Applikationen über das Internet. Jeder Web-Service wird eindeutig über einen URI identifiziert, während seine Schnittstellen über XML basierende Nachrichten angesprochen werden.

WSDL (Web Services Description Language)

Eine XML-Datei, die beschreibt, welche Dienste und Operationen ein Server anbietet. Sie definiert Strukturen, die für den Klienten bindend sind, wenn er einen Dienst benutzt.

WWW (World Wide Web)

Bezeichnung für ein weltweites Informationsnetz, das aus miteinander verknüpften HTML-Seiten besteht.

W3C (World Wide Web Consortium)

Gremium zur Standardisierung von Techniken, die das Internet betreffen.

XMI (XML Metadata Interchange Format)

XML-Bildungsvorschriften zum Austausch von MOF-konformen Metadaten.

XML (Extensible Markup Language)

Schema zur Informationsübertragung mit erweiterbaren Strukturen.

XSL (Extensible Stylesheet Language)

Definiert das Layout für XML-Dokumente. XSLT dient der Transformation von XML-Dokumenten, beispielsweise in HTML.

Index