Eine agentenbasierte Architektur zur Anwendung semantischer Netze im Rapid Product Development

**Von der Fakultät Maschinenbau der Universität Stuttgart
zur Erlangung der Würde eines
Doktor-Ingenieurs (Dr.-Ing.) genehmigte Abhandlung**

**Vorgelegt von
Dipl.-Inf. Michael Karsten Diederich
aus Stuttgart**

Hauptberichter:	**Prof. Dr.-Ing. Dieter Spath**
Mitberichter:	**Prof. Dr.-Ing. Engelbert Westkämper**
Tag der mündlichen Prüfung:	**07.02.2007**

**Institut für Arbeitswissenschaft und Technologiemanagement
der Universität Stuttgart**

2006

Vorwort

Die vorliegende Arbeit entstand während meiner Tätigkeit am Institut für Arbeitswissenschaft und Technologiemanagement (IAT) der Universität Stuttgart. Die zugrundeliegenden Forschungsarbeiten wurden durch das von der Deutschen Forschungsgemeinschaft (DFG) geförderte Teilprojekt „Integrationsplattform für aktive Wissenskommunikation“ des Sonderforschungsbereich 374 „Entwicklung und Erprobung innovativer Produkte – Rapid Prototyping“ ermöglicht.

Herrn Prof. Dr-Ing. Dieter Spath, Leiter des Instituts für Arbeitswissenschaft und Technologiemanagement (IAT) der Universität Stuttgart und des Fraunhofer Instituts für Arbeitswirtschaft und Organisation (IAO) in Stuttgart, gilt für die wissenschaftliche Unterstützung und die wohlwollende Förderung dieser Arbeit mein herzlicher Dank.

Herrn Prof. Dr.-Ing. Engelbert Westkämper, Leiter des Instituts für Industrielle Fertigung und Fabrikbetrieb (IFF) der Universität Stuttgart und Leiter des Fraunhofer Instituts für Produktionstechnik und Automatisierung (IPA), danke ich für die Übernahme des Mitberichts.

Herrn Prof. Dr.-Ing. habil. Joachim Warschat, Institutsdirektor des Fraunhofer IAO, gilt mein besonderer Dank für die zahlreichen Anregungen und die kontinuierliche Betreuung.

Danken möchte ich auch allen meinen Kolleginnen und Kollegen und wissenschaftlichen Hilfskräften, die zum Gelingen dieser Arbeit beigetragen haben. Mein herzlicher Dank gilt Herrn Dr.-Ing. Oliver Schumacher, der mir mit wertvollen Hinweisen zum Thema zur Seite stand.

An dieser Stelle möchte ich mich vor allem bei meinen Eltern herzlich bedanken, ohne deren tatkräftige Unterstützung und deren Geduld auch in schwierigen Phasen diese Arbeit nicht möglich gewesen wäre.

Stuttgart im Februar 2007 Michael Diederich

Inhaltsverzeichnis

Abbildungsverzeichnis

Tabellenverzeichnis

Abkürzungsverzeichnis

Abkürzung	Erklärung
ABLE	Agent Building and Learning Environment
ACL	Agent Communication Language
ASN	Aktives Semantisches Netz
ASN-QL	Aktives Semantisches Netz – Query Language
ASZI	Aktives Semantisches Zuverlässigkeits-Informationsnetz
AMA	Agent Monitoring Agent
AMP	Active Monitor Present
AV	Annahme Vorschlag
BDI	Belief-Desire-Intension
CAL	Communicative Act Library
CASL	Cognitive Agents Specification Language
CNP	Contract Net Protocol
COOL	Cooperation Language
DAT	Duplicate Address Test
DCE	Distributed Computational Economy
DSD	Document Structure Description
DTD	Data Type Definition
EBNF	Erweiterte Backus Naur Form
ECA-Regel	Event-Condition-Action-Regel
FIPA	Foundation for Intelligent Physical Agents
FLWOR	For Let Where Or Return
FSM	Finite State Machine
HMS	Holonic Manufacturing System
ICL	Interagent Communication Language
IEC	International Electrotechnical Commission
IEEE	Institute of Electrical and Electronics Engineers
IP	Internet Protocol
ISO	International Organization for Standardization

Abkürzung	Erklärung
JADE	Java Agent Development Framework
JXTA	Java XML Transfer Architecture
KIF	Knowledge Interchange Format
KQML	Knowledge Query and Manipulation Language
KR	Kostenrechner
LAN	Local Area Network
LRMP	Light-weight Reliable Multicast Protocol
MAN	Metropolitan Area Network
MAS	Multi Agent System
OAA	Open Agent Architecture
ODMG	Object Database Management Group
PB	Prototypenbauer
PP	Projektplaner
QFunction	Query Function
QM	Qualitätsmanager
RDF	Resource Description Framework
RELAX	REgular LAnguage description for XML
RETSINA	Reusable Environment for Task-Structured Intelligent Networked Agents
RMP	Reliable Multicast Protocol
RP	Ring Purge
RPD	Rapid Product Development
RQL	RDF Query Language
Sfb 374	Sonderforschungsbereich 374 *Entwicklung und Erprobung innovativer Produkte – Rapid Prototyping*
SMP	Standby Monitor Present
SQL	Structured Query Language
SRM	Social Reasoning Mechanism
TCP	Transmission Control Protocol
UAI	Unified Agent Identifier

Abkürzung	Erklärung
VP	Verhandlungspartner
WAN	Wide Area Network
XML	eXtensible Markup Language
XQuery	XML Query Language

1 Einleitung

In den heutigen Märkten nimmt die schnelle Entwicklung neuer Produkte für den Verbraucher und die Unternehmen einen zentralen Stellenwert ein [Agh 85]. Der Erfolg eines Produktes ist dabei nicht nur ausschließlich vom Innovationsgrad, Qualität und Preis des Produktes abhängig, sondern auch vom Zeitpunkt an dem das Produkt auf den Markt kommt. Zusätzlich wird durch den Trend zur Verkürzung der Entwicklungszeiten auf die Unternehmen Druck aufgebaut, der sich in Zukunft noch verschärfen wird [Bul 96b]. Um am Markt konkurrenzfähig zu bleiben, müssen Unternehmen daher schnell und flexibel auf Markt- und Kundenforderungen reagieren. Dies wird aber durch vorherrschende starre, sequentielle und arbeitsteilige Abläufe behindert [Wes 94]. Durch die immer komplexer werdenden Produkte und eingesetzten Technologien ist eine Spezialisierung von Experten notwendig. Dieses verlangt wiederum eine Wissens- und Kompetenzorientierung der gesamten Produktentwicklung [Spa 03a]. Es führt zu einem stark erhöhten Abstimmungsbedarf und Koordinationsaufwand und erschwert die Integration von neuen Produktfunktionen [Spa 03b]. Deshalb werden in der Zukunft vermehrt Methoden zur Wissensintegration für eine effiziente Teambildung benötigt [Bul 00]. Lösungspotentiale für die genannten Herausforderungen liegen in der Organisation der an der Produktentstehung beteiligten Bereiche im Sinne einer auf Markt und Kunden ausgerichteten, evolutionären Produktentwicklung. Sie wird auch als Rapid Product Development (RPD) bezeichnet [Bul 96a].

Das RPD zeichnet sich durch dezentrale Zusammenarbeit von Experten aus unterschiedlichen RPD-Domänen, wie Kostenrechnung, Projektplanung, Konstruktion, Prototypenbau, etc. aus. Um diese Zusammenarbeit zu unterstützen wurde der Ansatz des *Aktiven Semantischen Netzes (ASN)* im Rahmen des Sonderforschungsbereichs 374 *Entwicklung und Erprobung innovativer Produkte – Rapid Prototyping* konzipiert und realisiert [Eck 00a]. In der Produktentstehung hat sich gezeigt, dass die objektorientierte Datenmodellierung gegenüber dem relationalen Ansatz in Bezug auf die immer komplexer werdenden Produkte entscheidende Vorteile aufweist [Spa 02].

Das Aktive Semantische Netz erlaubt das Ablegen von strukturierten Informationen und deren semantischen Zusammenhängen in einem objektorientierten Umfeld. Zusätzlich besitzt das ASN eine aktive Komponente in Form eines Regelwerks, um Informationen einer Wissensdomäne in eine andere innerhalb des ASN propagieren zu können.

Die Funktionen des ASN sind auf eine datenorientierte Ebene beschränkt. Der Zugriff auf das ASN wird durch eine objektorientierte Programmierschnittstelle ermöglicht. Bei der Anwendung des ASN hat sich jedoch gezeigt, dass diese nahezu auf das Funktionsspektrum traditioneller Datenbanksysteme beschränkt bleibt, solange für die erweiterten Repräsentationskonstrukte keine adäquaten Anfragestrukturen

bereitgestellt werden. Somit entsteht die Notwendigkeit eine erweiterte Funktionalität, zur Integration von interdisziplinären Anwendungen mit dem ASN, in Form einer RPD-Middleware aufzubauen.

Zur effektiveren Ausnutzung der Möglichkeiten des ASN sind erweiterte Zugriffs- und Information-Retrieval-Methoden notwendig, die Zugriffe auf ASN-Modelle auf derselben Abstraktionsstufe gestatten, wie sie zur Modellierung von Informationen im ASN verwendet werden. Daraus leitet sich die Anforderung an die RPD-Middleware eines datenstrukturunabhängigen Zugriffs auf Informationen im ASN ab.

Neben einer vereinfachten und komfortableren Formulierung von Anfragen besteht andererseits das Ziel der RPD-Middleware, die Ausdrucksmächtigkeit von Anfragen zu erhöhen. Eine höhere Ausdrucksmächtigkeit im Zugriff auf Produktmodelle soll nicht nur die Bedienbarkeit des Systems erleichtern, sondern hat auch eine bessere Performance in der Auswertung von Zugriffen zur Folge. Statt einer größeren Menge einfacherer Anfragen soll die erweiterte Schnittstelle eine geringere Anzahl komplexerer Anfrageformulierungen erlauben, wodurch der Kommunikationsaufwand in einer verteilten Umgebung minimiert werden soll.

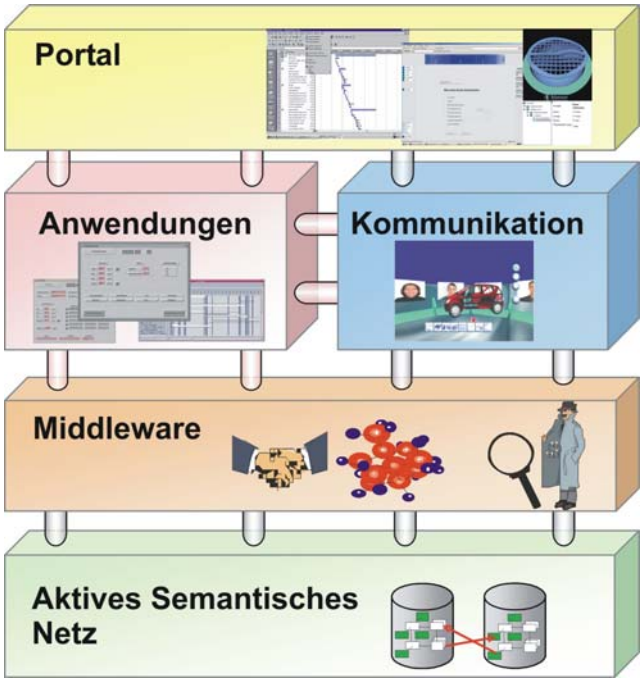


Bild 1: Einordnung der Middleware in die RPD-Infrastruktur

Zudem soll durch asynchrone Kommunikation das ASN aktive Elemente aufweisen. Ereignisse, die beispielsweise die Interaktion eines Experten mit dem System erfordern, müssen durch intelligente Weiterleitung der Information an die entsprechende RPD-Anwendung gesendet und durch diese dem Experten nutzbar gemacht werden. Ein wesentlicher Bestandteil der RPD-Middleware besteht daher in der Bereitstellung von Funktionalität zur Informationsvermittlung (siehe Bild 1).

Für die Realisierung von Middleware-Systemen existieren in der Informatik verschiedene Ansätze. Zum einen wird versucht, Verteiltheit von Anwendungen und der dazugehörigen Komponenten durch Nachrichtenaustausch (Messaging) zu erreichen. In diesem Bereich haben sich insbesondere Multiagentensysteme als geeignet erwiesen, da neben dem reinen Informationsaustausch bereits Anwendungsfunktionalität in den Agenten selbst realisiert werden kann [Coh 95]. Als Beispiel für die erfolgreiche Umsetzung dieser Technologie kann hier das Projekt ARCHON angeführt werden [Jen 95]. Zum anderen konzentrieren sich die Middleware-Ansätze auf die Standardisierung von Programmierschnittstellen und Übertragungsprotokollen [Ber 96].

Ziel dieser Arbeit ist die Entwicklung einer agentenbasierten RPD-Middleware, die die funktionale Lücke zwischen dem ASN, als Datenhaltung, und den RPD-Anwendungen, als Werkzeuge des Produktentstehungsprozesses, schließt, um eine Integration der RPD-Experten entlang des RPD-Prozesses zu erreichen.

In Bild 2 ist die Gliederung der Arbeit graphisch dargestellt. Auf Basis der Problemstellung des Rapid Product Development und dem Stand der Technik (siehe Kapitel 2) wird in Kapitel 3 die Zielsetzung und die Anforderungen an die RPD-Middleware spezifiziert. Aufbauend auf den Anforderungen werden die benötigten Methoden zur Informationsbeschaffung und -aufbereitung, sowie für die Informationsüberwachung und die Koordination in Kapitel 4 entwickelt. Parallel zu Kapitel 4 wird in Kapitel 5 die RPD-Middleware-Architektur entwickelt und in die bestehende RPD-Umgebung eingepasst. Die RPD-Middleware-Methoden werden im Kapitel 6 in RPD-Agenten überführt und in die Architektur integriert. Kapitel 7 stellt einen Abriss über die prototypische Realisierung dar. In Kapitel 8 wird anhand eines Anwendungsbeispiels der Gebrauch und der Nutzen der RPD-Middleware vorgestellt. Die Arbeit schließt mit einer Zusammenfassung (siehe Kapitel 9) und einem Ausblick (siehe Kapitel 10) über mögliche Weiterentwicklungen.

Im Anhang ist ein Abriss über bestehende Agentensysteme (Anhang A) dargestellt. Die verwendeten Programmiertechniken und Darstellungshilfsmittel finden sich im Anhang B. Die Sprachdefinitionen der Anfragesprachen (siehe Anhang C) für die RPD-Agenten, sowie die entwickelten Algorithmen (siehe Anhang D) sind ebenso im Anhang dargestellt, wie die Nachrichtenformate der zwischen den RPD-Anwendungen und der RPD-Middleware ausgetauschten Nachrichten (siehe Anhang E). Ein möglicher Durchlauf des Anwendungsbeispiels aus Kapitel 8 wird im Anhang F detailliert vorgestellt.

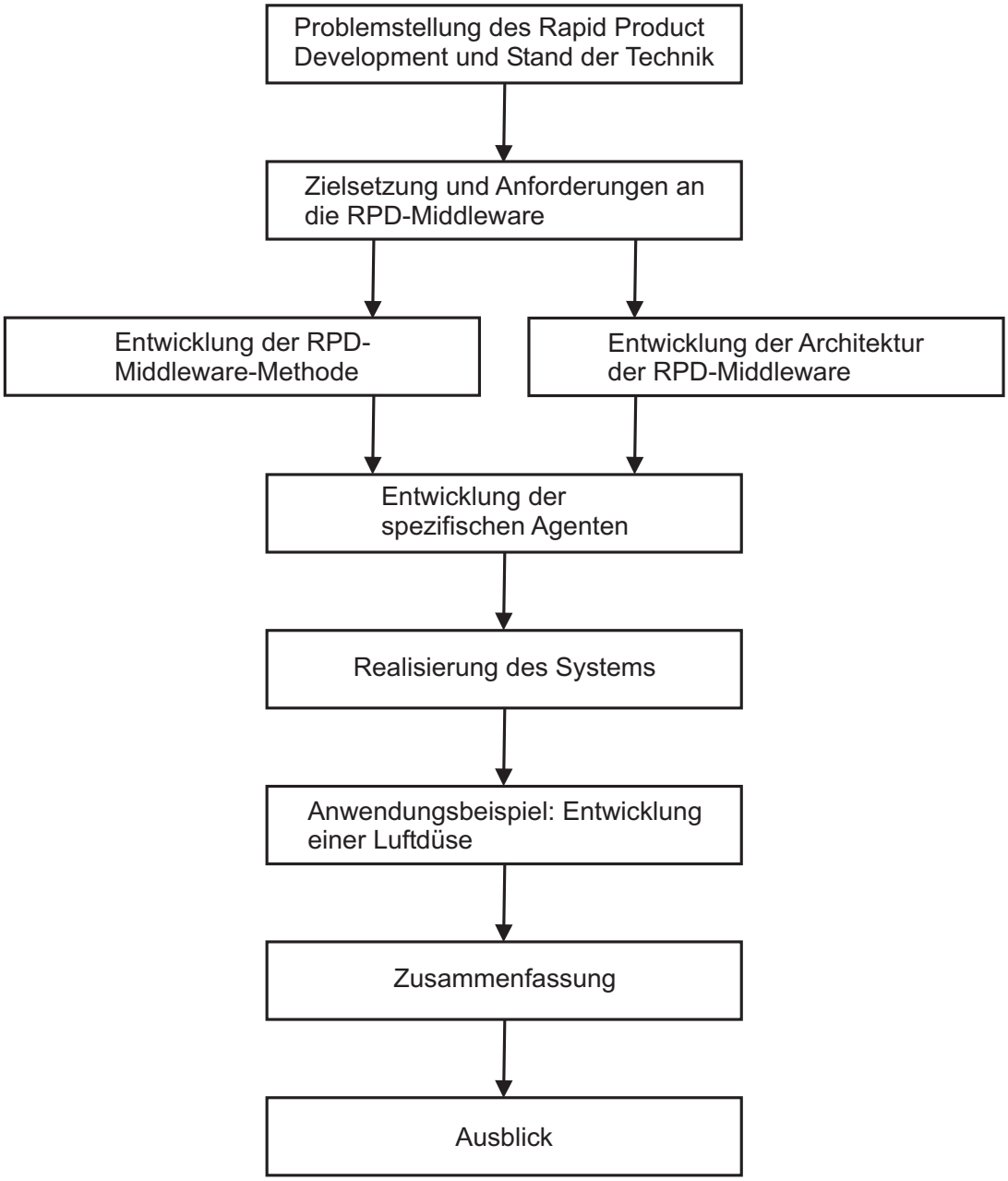


Bild 2: Aufbau der Arbeit

2 Problemstellung und Stand der Technik

2.1 Problemstellung

Der stark arbeitsteilig organisierte Produktentstehungsprozess wird bestimmt durch die wachsende Zahl von hoch spezialisierten Experten. Dadurch ist neben der ablaufbedingten Aufteilung von Aufgaben eine Verteilung der Kompetenzen auf Experten zu beobachten. Dies erschwert, hervorgerufen durch einen erhöhten Abstimmungsbedarf, die Integration von neuen Produktfunktionen, wodurch sich eine erhöhte Kommunikation der Experten ergibt, um eine verbesserte Koordination des Produktentstehungsprozesses (RPD-Prozess) zu erzielen.

Für eine iterative, evolutionäre Produktentwicklung ist die Nutzung von fachübergreifenden Informationen in allen Phasen des Produktentwicklungsprozesses eine große Herausforderung [War 00], [Kat 99]. Ein Problem stellt immer noch die situationsgerechte Bereitstellung von Prozess-, Konstruktions- und Technologiewissen aber auch von Wissen über Zeiten, Kosten und Qualität dar [Khu 98], [Wes 02].

Eine wesentliche Herausforderung zukünftiger Systemgenerationen zur Unterstützung der Produktentwicklung ist daher, von bisher einzelplatzorientierten zu intelligenten und kooperationsorientierten Bearbeitungsstrukturen zu gelangen. Hierzu wurden in einem ersten Schritt eine Vielzahl von hoch spezialisierten einzelplatzorientierten aber auch verteilten RPD-Anwendungen entwickelt, die einzelne Aspekte des RPD-Prozesses abdecken. In einem zweiten Schritt müssen nun diese RPD-Anwendungen zu einem Gesamtkonzept zusammengeführt werden.

Die RPD-Anwendungen haben eines gemeinsam: sie sind auf ihren Anwendungsbereich fokussiert, sowohl inhaltlich als auch konzeptionell. Die inhaltliche Fokussierung bezieht sich dabei auf den Problemraum, wie beispielsweise die Projektplanung oder die Konstruktion. Die konzeptionelle Ausrichtung der RPD-Anwendungen orientiert sich unter anderem an den Bedürfnissen der Nutzer. So kommt ein Kostenrechner unter Umständen mit einem Einzelplatzsystem aus, während die Projektplanung, als das RPD-Prozess begleitende Instrument, eine verteilte Anwendung benötigt, um die Planung der einzelnen Teams koordinieren zu können.

2.1.1 Integration von RPD-Experten unterschiedlicher Domänen

Der RPD-Prozess umfasst alle benötigten Wissensdomänen und deren Experten, die für eine erfolgreiche Durchführung von RPD-Projekten notwendig sind. Dies sind vor allem die Bereiche Prototypenbau, Qualitäts- und Kostenmanagement, Projektplanung, Simulation und Bewertung von Prototypeigenschaften, Konstruktionsverfahren, sowie Verfahren zur optimalen Materialauswahl.

Die Integration muss zwei Arten von Verteiltheit überwinden. Die erste Art ist charakterisiert durch eine inhaltliche Dimension, bedingt durch die Beteiligung unter-

schiedlicher Wissensbereiche am RPD-Prozess. Das explizite Wissen der RPD-Experten liegt in unterschiedlicher Granularität und in unterschiedlicher Strukturierung vor. Die zweite Art besitzt eine räumliche Dimension. Aufgrund der Tätigkeiten sind die am RPD-Prozess beteiligten Experten räumlich getrennt, beispielsweise muss ein Prototypenbauer auf Fertigungsstätten zurückgreifen können, während für einen Kostenrechner ein Büroumfeld üblich ist. Beide Dimensionen stellen grundlegende Problematiken für die Integration der RPD-Domänen im RPD-Prozess dar.

Zum einen müssen einheitliche Schnittstellen zwischen den RPD-Anwendungen unterschiedlicher Domänen geschaffen werden, die es ermöglichen unterschiedlich strukturierte Informationen auszutauschen. Diese Unterschiede können zum einen verschiedene Bezeichnungen von ein und derselben Informationseinheit sein und zum anderen können Informationen auf unterschiedliche Art und Weise modelliert sein. So versteht ein Konstrukteur unter Umständen unter den Kosten eines Prototyps lediglich die Materialkosten, während der Kostenrechner zusätzlich die Arbeitsmittel und Rüstzeiten den Kosten zuordnet.

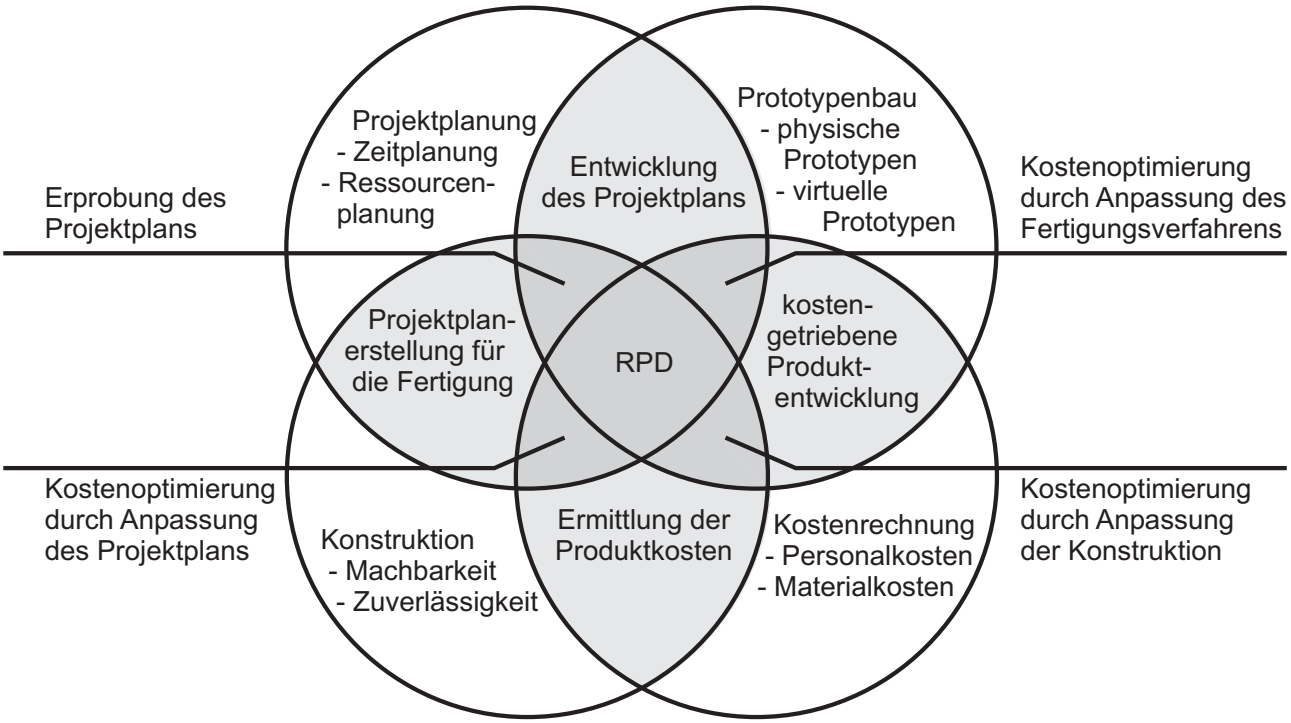


Bild 3: RPD-Wissensbereiche

Als erster Integrationsschritt wurde als gemeinsame Datenbasis das Aktive Semantische Netz (ASN) im Rahmen des Sonderforschungsbereichs 374 entwickelt. Die unterschiedlichen RPD-Domänen legen ihre Informationen, die für den RPD-Prozess von Bedeutung sind, im ASN ab. Dieses explizite Wissen ist dabei logisch nach den Zusammenhängen zwischen den Wissensdomänen strukturiert. Dadurch stellt das ASN den verteilten Entwicklungsteams des RPD eine gemeinsame Grundlage der Datenhaltung zur Verfügung, die alle Wissensdomänen des Pro-

duktentstehungsprozesses in einem integrierten Produktmodell repräsentiert und die Voraussetzung für Kommunikations- und Kooperationsmechanismen ist (siehe Bild 3).

Zusätzlich besteht der Bedarf an einer Kommunikationsumgebung als Basis für den Informationsaustausch zwischen den RPD-Anwendungen, denn die prozessbegleitenden Softwarewerkzeuge besitzen kein Wissen über die Kommunikationsmöglichkeiten der jeweils anderen RPD-Anwendung. Die einzige Möglichkeit zwischen RPD-Anwendungen zu kommunizieren, besteht durch den Informationsaustausch über das ASN. Dies ist eine einseitige Kommunikation, denn nur die RPD-Anwendung legt Informationen im ASN ab bzw. ruft explizites Wissen aus dem ASN ab. Das ASN ist nicht in der Lage aktiv Informationen der RPD-Anwendung zu übermitteln, ebenso wenig kann eine RPD-Anwendung über das ASN direkt mit einer anderen RPD-Anwendung kommunizieren.

2.1.2 Aufbereitung des RPD-Wissens für unterschiedliche Domänen

Die Informationselemente des ASN enthalten Informationen, die entweder nur für eine oder für mehrere RPD-Anwendungen oder gar für die gesamte RPD-Domäne von Interesse sind. So ist unter Umständen für einen Kostenrechner im Gegensatz zum Prototypenbauer nicht das Fertigungsverfahren eines Prototyps von Bedeutung, wenn er lediglich die Materialkosten bestimmen möchte. Umgekehrt stehen für den Prototypenbauer nicht unbedingt die Einzelkosten jedes Bauteils des Prototyps im Vordergrund, sondern vielmehr die Kosten in Abhängigkeit von der Werkstoffauswahl, sodass er eine andere Sicht auf die Kostendaten benötigt als der Kostenrechner.

Der Projektplaner versteht beispielsweise unter dem Begriff *Bearbeitungszeitraum* die Zeitspanne, die für die Herstellung eines Prototyps benötigt wird, begonnen mit der Werkzeugherstellung über Rüstzeiten, etc.. Dagegen beinhaltet der *Bearbeitungszeitraum* für den Konstrukteur unter Umständen lediglich die reine Fertigungszeit des Prototyps. Dieses Missverständnis kann gravierende Auswirkungen auf den Projektplan und damit auf den RPD-Prozess haben. Dies bedeutet die RPD-Domänen hinterlegen im ASN Informationen unterschiedlicher Bedeutung unter ein und demselben Begriff.

Zusätzlich ist die Granularität der Informationen je RPD-Domäne unterschiedlich. So stehen für den Prototypenbauer nicht unbedingt die Einzelkosten jedes Bestandteils des Prototyps im Vordergrund, sondern vielmehr nur die Gesamtkosten. Während der Kostenrechner für sein Kostenmodell alle Arten von Kosten benötigt, neben den Materialkosten auch Arbeitszeit und Einsatzkosten von Werkzeugen, und diese für alle Teile des Prototyps einzeln aufgeschlüsselt.

Durch den Einsatz des ASN können die Informationen der RPD-Domänen strukturiert abgelegt werden. Dabei ist die Möglichkeit geschaffen worden, Informationseinheiten um zusätzliche RPD-Domänen spezifischen Informationen zu erweitern.

Damit sind alle benötigten Informationen im ASN vorhanden, lediglich das Auffinden dieser Informationen gestaltet sich als schwierig, da der Zugriff auf Informationen im ASN nur durch die direkte Adressierung des Informationselements möglich ist. Dies bedeutet, wenn der Konstrukteur eine neue Informationseinheit mit dem Namen *Rüstzeitkosten* für seine Prototypen einbringt, so bleibt das dem Kostenrechner verborgen.

Eine Schwierigkeit Informationen aus dem ASN unter direkter Angabe des Informationselements aufzufinden, ist die Aufteilung der Informationen in unterschiedliche Informationseinheiten. Zum Beispiel kann die Adresse eines RPD-Mitarbeiters direkt an die Informationseinheit *Mitarbeiter* angeheftet oder aber als extra Informationseinheit abgebildet sein, die dann mit der Informationseinheit *Mitarbeiter* semantisch verknüpft ist.

Für diese Fragestellungen werden Informations-Retrieval-Mechanismen benötigt, die von der Datenstruktur abstrahieren. Unter Informations-Retrieval-Mechanismen werden Methoden verstanden, die es erlauben Informationen nach vorgegebenen Kriterien aus einem Datenbestand zu extrahieren, dessen interne Struktur nicht bekannt ist [Hea 93], [Nav 97]. Des Weiteren stellen diese Mechanismen eine Integration der RPD-Domänen innerhalb des RPD-Prozesses dar.

Eine weitere Herausforderung an die Informationsbeschaffung aus dem ASN entsteht, wenn eine RPD-Domäne Informationen aus zwei oder mehreren nicht direkt zusammenhängenden Wissensgebieten benötigt. Das ist beispielsweise der Fall wenn die Qualitätssicherung der Frage nachgeht, welcher Prototypenbauer am schnellsten einen Prototyp bei vorgegebener Qualität erstellen kann. Hierfür müssen für alle Prototypen mit der vorgegebenen Qualität und deren Hersteller ausfindig gemacht werden, um dann die Herstellungsdauer der gefundenen Prototypen anhand des Projektplans mit der Qualitätsrangliste in Verbindung bringen zu können.

Für diese Problematik werden Aggregationsmechanismen benötigt, die in Verbindung mit den Retrieval-Methoden, die Teilergebnisse zu einem Gesamtergebnis zusammenfassen. Dieses Gesamtergebnis stellt dann eine neue Information dar, die so nicht direkt im ASN modelliert wurde.

2.1.3 Überwachung des RPD-Wissens

Mit Hilfe einer Lösung für das Problem der Wissensaufbereitung ist es möglich Wissen zwischen Domänen verbessert auszutauschen bzw. aufzufinden, aber eine Wissenskommunikation als der direkte aktive Austausch von Informationen zwischen RPD-Domänen findet damit noch nicht statt. So wird beispielsweise ein Kostenrechner nicht automatisch darüber informiert, wenn sich das Material des Prototyps und damit die Kosten verändert haben.

Die Ergebnisübergabe beim Wechsel von einem Prozessschritt zum nächsten ist derzeit mit Hilfe des ASN nicht möglich. Hat der Prototypenbauer einen virtuellen Prototyp fertig gestellt, so muss er derzeit die RPD-Experten für die Simulation benachrichtigen, damit die Funktionssimulation beginnen kann. Diese Vorgehensweise verbraucht kostbare Arbeitszeitressourcen, denn im ungünstigsten Fall ist der Simulations-Experte zum Zeitpunkt der Fertigstellung nicht erreichbar, während zu einem späteren Zeitpunkt der Prototypenbauer für die Übergabe des Prototypen nicht zur Verfügung steht. Besser wäre es, wenn der Simulations-Experte direkt in seiner RPD-Anwendung eine Mitteilung über die Fertigstellung des Prototyps erhält, sodass er seinem Zeitplan entsprechend die Simulation durchführen kann. Hierfür wird eine Methode zur Überwachung des im ASN hinterlegten Wissens benötigt, die flexibel und konfigurierbar ist. Der RPD-Experte muss selbst entscheiden können, welche ASN-Informationen für ihn von Bedeutung sind.

Die Überwachung des RPD-Wissens hat damit eine synchronisierende Wirkung auf die RPD-Domänen entlang des RPD-Prozesses und fördert die Integration der unterschiedlichen RPD-Domänen. Wobei die Abstimmung der einzelnen Prozessschritte dadurch entsteht, dass Tätigkeiten entlang des RPD-Prozesses erst dann ausgeführt werden, wenn Vorgänger-Prozessschritte abgearbeitet sind.

2.1.4 Koordination von RPD-Anwendungen

Jede RPD-Domäne deckt einen bestimmten Bereich des RPD-Prozesses ab. Bei der Durchführung ihrer Aufgabe ist sie auf Ergebnisse von anderen Bereichen angewiesen und liefert als Grundlage für die Weiterarbeit anderen Domänen ihre Erkenntnisse, wodurch eine enge Verzahnung der Wissensdomänen innerhalb des RPD-Prozesses entsteht. Die Wissensdomänen müssen an den entstehenden Übergabepunkten koordiniert werden. Da die Ergebnisse der einzelnen Bereiche Sackgassen in der Prototypenentwicklung innerhalb des RPD-Prozesses aufdecken können, wodurch eine iterative Abarbeitung des RPD-Prozesses verursacht wird, müssen hierfür geeignete Koordinationsmechanismen entwickelt werden.

Die Koordinationsmechanismen müssen zum einen den Subprozess der Informationsübergabe steuern und zum anderen müssen beteiligte RPD-Domänen in die Lage versetzt werden, Veränderungen sofort zu erfahren, um geeignet darauf reagieren zu können. Die Steuerung der Informationsübergabe durch ein zwischen den RPD-Domänen gemeinsam abgestimmtes Informationsübergabeprotokoll verhindert, dass einzelne Bereiche ihre Ziele für so wichtig erachten, dass andere Domänen nicht mehr in der Lage sind, ihre Ziele in den RPD-Prozess einzubringen. Das Informationsübergabeprotokoll muss alle an der Koordination beteiligten Partner fair behandeln.

Dabei ist ein starres Synchronisationsmanagement, wie es im vorangegangenen Abschnitt bei der Überwachung des RPD-Wissens noch möglich war, nicht mehr ausreichend, denn dieses Modell geht davon aus, dass es zu keinen Komplikationen entlang des RPD-Prozesses kommt. Es werden daher Mechanismen benötigt,

die neben der synchronisierenden Wirkung auch einen verhandelnden Charakter besitzen. Wird beispielsweise ein Prototyp durch die Wahl eines Materials so teuer, dass die Produktion unrentabel wäre, und wird dieser Umstand durch die Kostenrechnung aufgedeckt, kann eine Freigabe beispielsweise für die Simulation erst erfolgen, wenn der Kostenrechner und der Prototypenbauer sich auf ein Material einigen, das sowohl die Vorgaben der gewünschten Eigenschaften als auch die Kosten des Prototyps berücksichtigt.

Durch die Schaffung eines Koordinationsmechanismus können besonders in Verbindung mit der Informationsüberwachung die Zeiten und damit die Kosten für die Verhandlung von Konfliktsituationen minimiert werden.

2.2 Stand der Technik

Auf der Basis der Problemstellung werden sowohl bekannte Multiagentenansätze als auch Kommunikations- und Kooperations-Techniken auf ihre Einsetzbarkeit in der RPD-Middleware untersucht.

2.2.1 Agentenarchitekturen und -modelle

In Wooldrige [Woo 95], [Woo 99] und Flores-Mendez [Flo 99] ist eine allgemeine Definition des Begriffs *Agent* gegeben:

„Ein Agent ist eine künstlich erstellte Einheit, die ihre Umgebung wahrnimmt [Bro 91], [Rus 95] und proaktiv oder reaktiv in dieser Umgebung handelt [Jen 98a], in der weitere Agenten existieren. Die Agenten interagieren untereinander [Sho 97] auf der Basis eines gemeinsamen Verständnisses von Kommunikation und Repräsentation von Informationen [Fin 97].“

Dabei ist unter dem Begriff der Umgebung entweder die reale Welt, ein Nutzer, der über eine Schnittstelle mit dem Agenten interagiert, eine Menge anderer Agenten, das Internet oder Kombinationen davon zu verstehen. Die Umgebung wirkt über Eingaben auf den Agenten ein und wird durch Ausgaben des Agenten beeinflusst.

Der Agent zeichnet sich durch vier Eigenschaften aus [Woo 95]:

- **Autonomes Handeln**: Agenten handeln selbständig ohne direkte Beeinflussung des Menschen oder anderer Systeme. Sie besitzen eine Art Kontrolle über ihr eigenes Handeln und ihren internen Zustand [Cas 95].
- **Sozialverhalten**: Agenten besitzen die Möglichkeit mit anderen Agenten oder dem Menschen mit Hilfe einer Agentenkommunikationssprache zu kommunizieren [Gen 94].
- **Reaktivität**: Agenten sind in der Lage ihre Umgebung zu beobachten und auf Veränderungen in der Umgebung zu reagieren.
- **Proaktivität**: Agenten werden aktiv ohne ihre Umgebung zu beobachten. Sie ergreifen selbständig die Initiative um ihre Ziele zu erreichen.

Agenten können dabei zustandslos oder zustandsbehaftet agieren. Im zustandslosen Fall wird auf eine Eingabe sofort über die Berechnung einer internen Funktion eine Ausgabe erzeugt. Im Gegensatz dazu werden im zustandsbehafteten Fall unter Umständen weitere Eingaben durch die Umgebung zur Berechnung der Ausgabe herangezogen (siehe Bild 4).

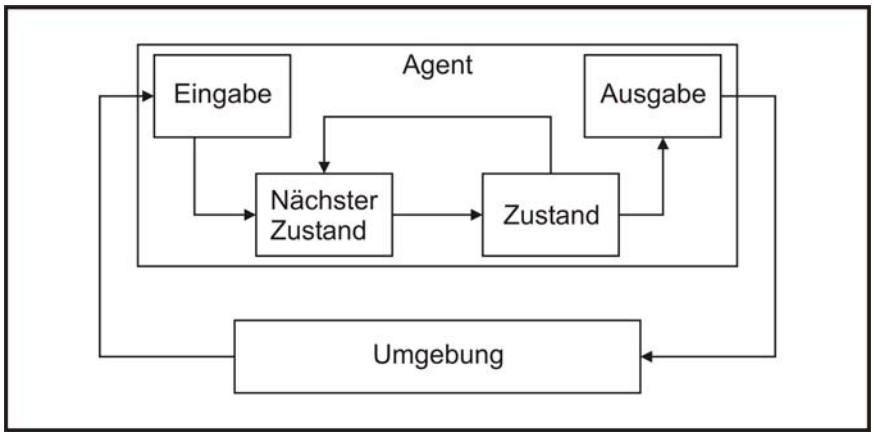


Bild 4: Schematische Darstellung eines zustandsbehafteten Agenten

In Fatima [Fat 98] ist der interne Aufbau eines Agenten beschrieben, wie er häufig in Multiagentensystemen verwendet wird. Der Agent besteht aus vier Komponenten (siehe Bild 5):

- **Kommunikationsmodul**: Das Kommunikationsmodul tauscht Nachrichten mit anderen Agenten aus.
- **Problemlösungsmodul**: Die Problemlösekomponente treibt das Lösen des lokalen Problems voran.
- **Verwaltungsmodul**: Das Verwaltungsmodul administriert die unterschiedlichen Nachrichtentypen und steuert die anderen Module des Agenten.
- **Datenbasis**: Die Datenbasis enthält zwei Arten von Informationen. Zum einen Informationen, um das lokale Problem zu lösen und zum anderen Informationen zur Synchronisation des globalen Problemlösungsprozesses, wenn mehrere Agenten parallel am gleichen Problem arbeiten.

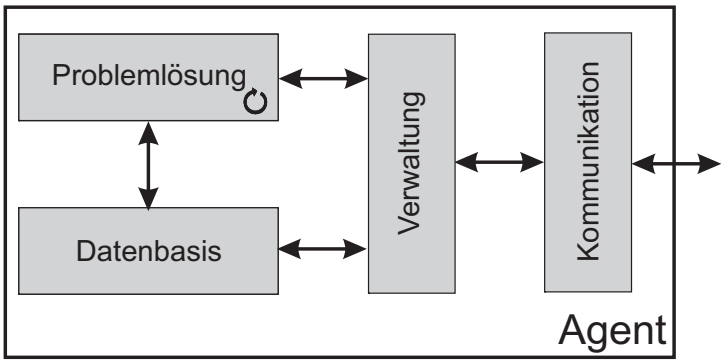


Bild 5: Agenten-Architektur nach Fatima [Fat 98]

Bei der Betrachtung der Agentenarchitekturen unterscheidet Wooldridge [Woo 99] zwischen vier Architekturtypen:

- **Logic based agents**: Diese Agenten bestimmen ihre Ausgabefunktion durch das logische Schließen auf der Basis von logischen Zusammenhängen, die in der Datenbasis des Agenten abgelegt sind [Gen 87], [Kon 86].
- **Reactive agents**: Dieser Agententyp besitzt einen Satz von Aktionen, die er abhängig von der Umgebung ausführt. Es besteht ein direkter Zusammenhang zwischen der aktuellen Situation der Umgebung und der Aktion des Agenten darauf [Bro 86], [Fer 96].
- **Belief-desire-intention (BDI) agents**: Das Schlussfolgern im BDI-Ansatz [Geo 99] basiert auf Datenstrukturen, die durch die Annahmen (beliefs), Wünsche (desires) und Ziele (intentions) repräsentiert werden [Bra 87], [Bra 88].

  Die sieben Komponenten des BDI-Ansatzes sind:

  - **Eine Menge aktueller Annahmen**: Diese bilden den aktuellen Zustand der Umgebung innerhalb des Agenten nach.
  - **Belief revision function**: Diese Funktion generiert auf der Basis einer Eingabe und den aktuellen Annahmen neue Annahmen.
  - **Option generation function**: Mit Hilfe dieser Funktion werden mögliche Optionen unter Berücksichtigung der aktuellen Annahmen und Ziele bestimmt.
  - **Eine Menge von Optionen**: Diese repräsentieren die Handlungsmöglichkeiten des Agenten.
  - **Filter function**: Der Agent passt mit dieser Funktion und auf der Basis der aktuellen Annahmen, Wünsche und Ziele seine Ziele an.
  - **Eine Menge von aktuellen Ziele**: Sie beschreiben die verfolgten Ziele des Agenten.
  - **Action selection function**: Diese Funktion bestimmt die Handlung des Agenten.

- **Layered architectures**: Die Entscheidungsfindung wird in Layered Architectures durch mehrere Softwareebenen realisiert, die jeweils einen bestimmten Teil zur Entscheidungsfindung beitragen. Es wird zwischen horizontaler und vertikaler Schichtbildung unterschieden. Die horizontale Schichtbildung (siehe Bild 6) bedeutet, dass der Agent in unterschiedliche Module aufgeteilt wird, die verschiedenen Aufgaben realisieren. Die Eingabe wird an alle Module gleichzeitig geleitet. Anschließend wird die Ausgabe aus den Ergebnissen aller Module bestimmt. Im Gegensatz dazu werden bei der vertikalen Schichtenbildung (siehe Bild 7) alle Module der Reihe nach durchlaufen, dabei kann die Ausgabe direkt nach der obersten Schicht erzeugt werden oder durch die unterste Schicht nach einem Rücklauf durch alle Schichten [Woo 99], [Jen 98b], [Mül 95].

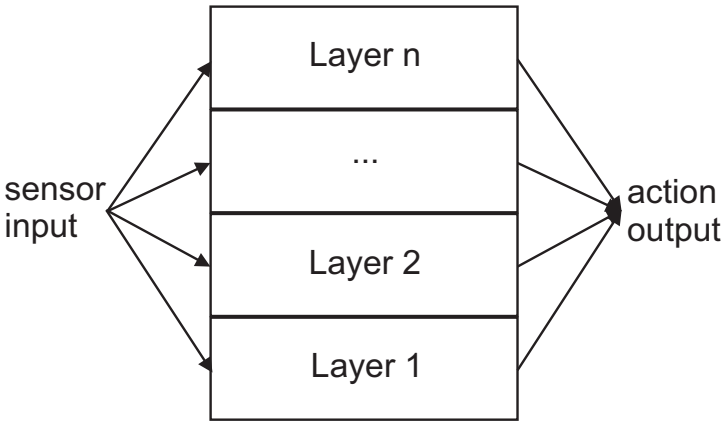


Bild 6: Schematische Darstellung der horizontalen Layered-Architektur

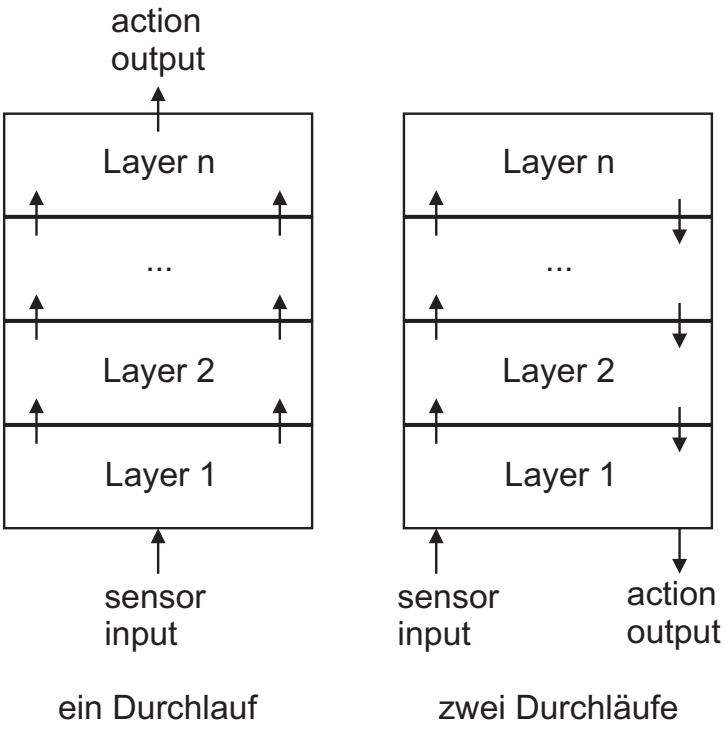


Bild 7: Schematische Darstellung der vertikalen Layered-Architektur

Eine mögliche Klassifizierung von Agententypen ist von Nwana [Nwa 96a] vorgestellt worden. Er definiert Agenten als Kombination von drei Einflussgrößen: die Autonomie, die Kooperation und die Lernfähigkeit. Diese drei Bereiche sowie die durch die gemeinsam überlappenden Bereiche entstehenden Möglichkeiten, liefern sieben Agententypen (Collaborative-, Interface-, Mobile-, Information/Internet-, Reactive-, Hybrid- und Smart-Agenten). Diese weisen unterschiedliche Abstufungen bezüglich Ihrer Intelligenz auf, so ist z. B. der Smart-Agent, der zu der obersten Intelligenzklasse gehört, für die Steuerung des Agentensystems zuständig.

In Bianchi [Bia 97] ist eine einfache Beschreibung eines Agenten dargestellt, die sich aus den folgenden Parametern zusammensetzt:

- **Regeln**: Verhaltensregeln des Agenten innerhalb des Agentensystems.
- **Andere Agenten**: Hier ist hinterlegt, welche Agenten sich im Agentensystem befinden und in welchem topologischen Zusammenhang sie untereinander stehen.
- **Zustand der Umgebung**: Der Zustand der Umgebung, den der Agent für die Erbringung seiner Aufgabe benötigt.
- **Kommunikationssprache**: Die Kommunikationssprache, die zur Interagentenkommunikation eingesetzt wird.
- **Basisagenten**: Die Basisagenten, mit denen der Agent für die Erbringung seiner Aufgabe kommunizieren muss.
- **Entscheidungsfähigkeiten**: Hier ist festgelegt, wie der Agent entscheiden kann, wer im Moment die Kontrolle über eine benötigte Ressource besitzt.

Die Foundation for Intelligent Physical Agents (FIPA) stellt Spezifikationen für das gesamte Spektrum der Agententechnologie bereit [FIP 02a]. Den Mittelpunkt der gesamten FIPA-Aktivitäten bilden die Beziehungen zwischen den Agenten mit dem Ziel, die Interoperabilität in komplexen Systemen zu erhöhen. Durch die Standardisierungen der FIPA ist eine weit verbreitete sowie allgemein akzeptierte Agenten-Architektur geschaffen worden. So stimmen wichtige Projekte, wie Agent Building and Learning Environment (ABLE) (siehe Anhang A) und Java Agent Development Framework (JADE) (siehe Anhang A) mit den Spezifikationen der FIPA überein. Aktuelle Agentenprojekte berücksichtigen bzw. greifen auf Teile des FIPA-Standards zurück.

2.2.2 Kommunikationsprotokolle für eine aktive RPD-Middleware

Kommunikation ist ein wesentliches Merkmal innerhalb eines Multiagentensystems und per Definition ein Mindestkriterium, das erfüllt werden muss [Gen 94]. Unter Kommunikationsprotokollen werden Vorschriften verstanden, die den Nachrichtenaustausch zwischen zwei oder mehreren Instanzen steuern. Diese Instanzen können sowohl Agenten als auch externe Anwendungen der Umgebung des Multiagentensystems sein. Die Kommunikation ist auf zwei Ebenen aufgeteilt. In der oberen Ebene werden die Kommunikationsarten und -wege behandelt, während in der unteren Ebene die Struktur und damit der Informationsgehalt der ausgetauschten Nachricht festgelegt wird [KI 01].

Kommunikationsbeziehungen und -wege

Für die Auswahl des richtigen Kommunikationsweges werden drei Arten von Kommunikationsbeziehungen unterschieden:

- **1-zu-1**: In dieser Beziehungsart kommuniziert genau ein Partner mit einem anderen Partner.
- **1-zu-n**: Hier kommuniziert ein Partner mit beliebig vielen anderen Partnern.
- **m-zu-n**: Es kommunizieren mehrere Partner mit beliebig vielen anderen Partnern.

Um diese Kommunikationsbeziehungen informationstechnisch abbilden zu können, gibt es in der Netzwerktechnologie [Tan 96] drei Arten von Kommunikationswegen:

- **Unicast**: Die Unicast-Technologie setzt die 1-zu-1 – Kommunikation um. Es wird genau ein Kommunikationskanal zwischen zwei Kommunikationspartner geschaltet.
- **Broadcast**: Die Broadcast-Technologie realisiert die 1-zu-n – bzw. die m-zu-n – Kommunikation. Die Kommunikationspartner gehören dabei einer Gruppe an und jeder erhält die Informationen, die ein Mitglied der Gruppe verschickt. Die Gruppenbildung erfolgt auf räumlicher Ebene und ist auf Netzwerke beschränkt. Eine Gruppenbildung nach inhaltlichen Gesichtspunkten über Routinggrenzen hinweg ist nicht möglich.
- **Multicast**: Die Multicast-Technologie bildet ebenso wie der Broadcast-Ansatz die 1-zu-n – bzw. m-zu-n – Kommunikation ab. Auch hier gehören die Kommunikationspartner einer Gruppe an, allerdings erfolgt die Gruppenbildung durch die Kommunikationspartner und ist auf keine räumliche Nähe, wie Netzwerke, beschränkt.

Die Multicast-Technologie weist in ihrer Grundform [Tan 96] die Schwäche des zuverlässigen Zustellens von Informationen auf, d. h. es wird nicht dafür garantiert, dass eine Informationseinheit den Empfänger erreicht bzw. in welcher Reihenfolge mehrere verschickte Informationseinheiten bei den Empfängern ankommen. Erweiterungen des Multicast-Protokolls, wie beispielsweise das *Light-weight Reliable Multicast Protocol* (LRMP) [Lia 00], gleichen diese Schwäche durch Mechanismen des Transmission Control Protocols (TCP) [RFC 793], [RFC 1122], [RFC 1323] aus. LRMP ist für heterogene Netzwerke mit Unterstützung mehrerer Sender aufgebaut. LRMP gewährleistet die Zustellung der Nachrichten und bietet mit Hilfe des Multicast-Protokolls die Möglichkeit, über verteilte Standorte Informationen auszutauschen. Im Allgemeinen wird das LRMP für die Implementierung von *Java Message Service*, welche die Kommunikations-Komponente für Verteilte Systeme (siehe Abschnitt 2.2.4) darstellen, eingesetzt [Hew 03], [Sun 00], [Swi 01], [Ube 02]. Auch für Realisierungen von Diensten der Peer-to-Peer-Spezifikation *Java XML Transfer Architecture* (JXTA) wurde LRMP benutzt [Sun 03].

Kommunikationssprachen

Die inhaltlichen Gesichtspunkte befassen sich mit dem Ziel der Kommunikation, also der beabsichtigten Wirkung, dem Inhalt der Nachricht und dem Ergebnis der Kommunikation [KI 01]. Da der reine Inhalt einer Nachricht oft missverständlich ist, müssen Regeln für die Interpretation des Inhalts der Nachricht definiert werden, dies ist zum Beispiel auf der Basis der Sprechakttheorie [Cov 96], [Dew 89], [Moo 93], [Huh 99] möglich. Dabei wurde untersucht, in wie weit der Aufbau der natürlichen Sprache auf die Kommunikation von Computersystemen untereinander übertragbar ist. In der Sprechakttheorie wurden hierbei Kommunikationsregeln erarbeitet und eine Grammatik sowie ein Basis-Wortschatz festgelegt. Die Sprechakttheorie ist auch bei der Teamkoordination im RPD von großer Bedeutung [Ceb 04].

Derzeit existieren Standardisierungsbestrebungen für zwei viel versprechende Agenten-Kommunikations-Sprachen, die die inhaltliche Strukturierung der Nachricht zum Ziel haben. Eine direkte Gegenüberstellung beider Ansätze findet sich in Labrou [Lab 99] und Luck [Luc 03b]. Dabei ist zwischen einer inneren und einer äußeren Sprache zu unterscheiden, wobei die innere Sprache in die Äußere eingebettet ist. Die Aufgabe der äußeren Sprache ist das Adressierungsmanagement der Nachricht, hier wird spezifiziert, an wen die Nachricht gerichtet ist und wie sie strukturiert ist. Die innere Sprache definiert die eigentliche Nachricht [Gen 94].

Ein Beispiel für die äußere Sprache ist die Knowledge Query and Manipulation Language (KQML) als eine Programmiersprache, die Sprechakte nutzt, um eine inhaltsorientierte Kommunikation zwischen Agenten aufzubauen [Fin 93], [Fin 94], [Fin 92], [Gen 92], [Lab 97], [Huh 99]. Mit KQML können strukturierte Anfragen an andere Agenten gestellt und der Informationsgehalt der Antwort interpretiert werden [Lab 94]. KQML weist aber auch Lücken auf, wie z. B. die Möglichkeit, dass ein Agent im Rahmen seiner Aufgabe ein Arbeitspaket an andere Agenten stellen kann [Cov 98]. Die eigentliche Nachricht wird in die KQML-Nachricht im Knowledge Interchange Format (KIF) [KIF 94] als innere Sprache eingebettet [Gen 92].

Die andere Standardisierungsbestrebung erfolgt durch die Foundation for Intelligent Physical Agents (FIPA). Die FIPA hat bereits als Agent Communication Language (ACL) für die Kommunikation eine *message structure* standardisiert [FIP 02b]. Auch die FIPA-ACL Semantik ist durch der FIPA-Communicative Act Library (CAL) spezifiziert worden [FIP 02c]. Zum Austausch von Informationen wird neben anderen Content Languages auch das KIF berücksichtigt und standardisiert. Die Standardisierungsbestrebungen der FIPA bauen dabei auf den Entwicklungen von KQML und KIF auf.

Aufbauend auf den beiden Standardisierungsbestrebungen existieren diverse Weiterentwicklungen. In Martin [Mar 99] wird eine Open Agent Architecture (OAA) vorgestellt, die unter anderem auch eine Weiterentwicklung von KQML und KIF zu ei-

ner Interagent Communication Language (ICL) beinhaltet. ICL behandelt die Nachrichten als Event. Die Events bestehen aus einem Typ, einer Parameterliste und dem Inhalt. Anhand des Typs entscheidet der Agent, welche Aufgabe er zu erfüllen hat. Die eventbasierte Vorgehensweise hat den Vorteil, dass die asynchron eintreffenden Nachrichten direkt im Agent als Events für die weitere Verarbeitung umgesetzt werden können. Ebenso wird in der in De Palma [DeP 00] vorgestellten agentenbasierten Middleware die Kommunikation ausschließlich durch den Austausch von Nachrichten vorgenommen.

Ein weiterer Ansatz, um Kommunikation von Agenten zu realisieren, wird in Barbuceanu [Bar 97] verfolgt. Es werden Kommunikationsregeln aufgestellt, die vom aktuellen Zustand des Agenten abhängen. Durch die Eingaben anderer Agenten wird ein Zustandswechsel verursacht und damit ein neuer aktueller Zustand bestimmt. Für den Agenten wird mit Hilfe eines Zustandübergangsdiagramms ein Konversationsplan aufgestellt, der festlegt, welche Kommunikation in welchem Zustand durch den Agenten erfolgen muss. Die strukturierten Nachrichten, des in Barbuceanu [Bar 97] vorgestellten Systems, werden in KQML verfasst. Auch in Moore [Moo 93] und d'Inverno [dIn 98] wird ein zustandsbezogenes Protokoll vorgestellt.

2.2.3 Koordinationsprotokolle für eine aktive RPD-Middleware

Im voran gegangenen Abschnitt wurden Möglichkeiten aufgezeigt, wie eine Information in Form einer Nachricht von einem Kommunikationspartner A zu einem Kommunikationspartner B gelangt. Dies ist jedoch für eine funktionierende Kommunikation nicht ausreichend; es müssen Koordinationsprotokolle definiert werden, die den Kommunikationsablauf zwischen den Bestandteilen der RPD-Middleware und den RPD-Anwendungen bestimmen. Die Koordinationsprotokolle haben folgende Ziele [Nwa 96b], [Nwa 94], [Jen 90]:

- **Vermeidung von Chaos**: Das Ziel ist es, durch die Zusammenführung aller Teilergebnisse, ein Gesamtziel zu erreichen. Würde jede Komponente, wie die Agenten und die RPD-Anwendungen ohne Koordination ihre Ziele verfolgen wäre das Gesamtziel nicht erreichbar.
- **Schaffung einer einheitlichen Umgebung**: Die Umgebung beinhaltet die Randbedingungen, in denen die Komponenten arbeiten. Eine Randbedingung ist z. B. die Fehlertoleranz des Gesamtsystems.
- **Koordination von verteilten Informationen, Ressourcen und Verantwortlichkeiten**: Die Nutzung von verteilten Informationen, Ressourcen und Verantwortlichkeiten muss zwischen den einzelnen Bestandteilen der RPD-Middleware abgestimmt werden.
- **Auflösung von Abhängigkeiten zwischen zwei Komponenten**: Beispielsweise können Informationen erst für die RPD-Anwendung aufbereitet werden, wenn sie aus dem ASN beschafft wurden.
- **Effizientes Arbeiten**: Das durch die Aufteilung auf spezialisierte Agententypen erreicht wird.

Es wird zwischen den folgenden Klassen von Koordinationsprotokollen unterschieden [Huh 99]:

- **Kooperationsprotokolle**: Kooperationsprotokolle arbeiten nach dem Prinzip *Teile-und-Herrsche*. Dabei werden Aufgaben aufgeteilt und die Ergebnisse wieder zusammengeführt. Das Aufteilen der Aufgaben kann zum Zeitpunkt des Systementwurfs stattfinden beispielsweise durch Aufteilung auf spezielle Agententypen oder während der Laufzeit durch Analyse der Aufgabe [Dur 87], [Kir 98].
- **Vertragsprotokoll**: Das Vertragsprotokoll, auch Contract Net Protocol (CNP) [Dav 83], [Smi 80] genannt, beruht auf dem Vertragsprinzip. Ein Agent bietet eine Leistung an und garantiert, dass die Leistung erbracht wird. Möchte ein anderer Agent diese Leistung nutzen, so schließt er mit dem anbietenden Agenten einen Vertrag ab. Eine Verbesserung des Contract Net Protocol ist in Rosenschein [Ros 94] beschrieben. In diesem Protokoll wird dem angebotssuchenden Agenten die Möglichkeit geboten, nicht nur das Angebot anzunehmen bzw. abzulehnen, sondern auch eine Nachbesserung des Angebots zu fordern.
- **Blackboard-Ansatz**: Unter dem Blackboard-Ansatz wird ein zentralistisches Verfahren verstanden, wo alle Informationen an einem Ort, dem Blackboard, abgelegt werden. Auf diesen Ort haben alle Koordinationsbeteiligte Zugriff. Die Koordination erfolgt durch Auswertung der vorhandenen Informationen auf dem Blackboard [Dur 88], [Jag 93], [Cor 88], [Hay 85], [Nii 86], [OGr 90].
- **Verhandlungsansatz**: Der Verhandlungsansatz ist ein verteiltes Verfahren, bei dem alle Koordinationspartner gleichberechtigt sind. Zwischen den Koordinationspartner wird ein Verhandlungsprotokoll beispielsweise regelbasiert [Bla 01] oder durch ein Zustandübergangsdiagramm spezifiziert. [Bar 99].
- **Marktmechanismen**: Marktmechanismen [Wel 95] funktionieren nach dem Prinzip der Marktwirtschaft. Die einzelnen Koordinationskomponenten bieten ihre Lösung, hinterlegt durch einen Preis, an. Eine gangbare Lösung wird anhand des Preis/Leistungsverhältnisses ausgewählt.

In Reed [Ree 98] werden fünf Koordinationstechniken für das Themenfeld der automatischen Problemlösung vorgestellt, wobei die ersten beiden Koordinationstechniken im Falle eines Konflikts eingesetzt werden, während die anderen drei Techniken zur Findung der Gesamtlösung dienen und Auswirkungen auf die Architektur des Agenten haben:

- **Persuasion**: Konfliktlösung zweier Agenten, die gegensätzliche Ergebnisse berechnet haben. Hierbei versucht ein Agent den anderen Agenten zu überzeugen, damit dieser sein Ergebnis nochmals einer Revision unterzieht.
- **Negotiation**: Konfliktlösung zweier Agenten, durch verhandeln. Wobei während der Verhandlung jeder Agent ein Stück von seiner idealen Lösung abrücken muss.

- **Inquiry**: Ein Agent erbittet bei einem anderen Agenten die ihm fehlende Information.
- **Deliberation**: Aufbau eines gemeinsamen Plans der Agenten zur Erlangung einer Lösung für das Gesamtziel.
- **Information seeking**: Besitzt ein Agent Wissen, das ein anderer Agent zur Lösung des Gesamtproblems benötigt, so wird dieses Wissen zwischen den Agenten ausgetauscht.

In den folgenden Beispielen werden die einzelnen Koordinationstechniken, die auf den jeweiligen Anwendungsfall angepasst sind, dargestellt.

- In Maturana [Mat 95] wird ein Verfahren für einen Koordinationsmechanismus auf dem Negotiations-Gedanke aufbauend vorgestellt, dabei wird die Verhandlung durch den Einsatz einer zentralen vermittelnden Instanz gesteuert.
- Das in Park [Par 94] dargestellte Agentensystem zur Lösung des Verkabelungsproblems bei der Entwicklung von Platinenlayouts verteilt nach dem Prinzip *Teile-und-Herrsche* seine Aufgaben auf unterschiedliche Agenten und steuert die Interaktion der Agenten durch eine zentrale Instanz.
- Die Aufgabenverteilung und -zuteilung zu den entsprechenden Agenten erfolgt im Multiagentensystem RETSINA [Syc 01] durch den Einsatz einer zentralen Instanz.
- In Timm [Tim 98], [Tim 99] ist ein Agentensystem beschrieben, das eingesetzt wird, um eine ökologische Transportlogistik zu unterstützen. Dabei stehen neben der Suche der preiswertesten Route in Verbindung mit der schnellsten Route auch ökologische Gesichtspunkte im Vordergrund. Die Agenten repräsentieren dabei jeweils ein Unternehmen und koordinieren sich über den Marktmechanismus nach Angebot und Nachfrage.
- Das COMRIS Agentensystem [Hau 99] hat den Aufbau eines Konferenzzentrums, in dem reale und virtuelle Aktivitäten parallel stattfinden, zum Ziel. Jeder Teilnehmer der Konferenz besitzt einen persönlichen Assistenten, der kabellos mit dem Konferenz-Intranet verbunden ist. Dieser persönliche Assistent, der COMRIS-Parrot, ist das Bindeglied zwischen dem realen und virtuellen Raum. Er beobachtet, was um seinen Besitzer herum geschieht, und informiert ihn über potentiell nützliche Treffen, laufende Demonstrationen, etc. Die Koordination der einzelnen persönlichen Assistenten und die Informationsbeschaffung erfolgt über einen Blackboard-Ansatz.
- In Appelrath [App 99] ist das Konzept eines Multiagentensystems für die verteilte Ablaufplanung beschrieben. Dabei hat das Agentensystem das Zusammenspiel zwischen der Produktion, dem Transport und der Lagerhaltung zu planen. Die lokal in der eigenen Anwendung berechneten Ablaufpläne, beispielsweise einer Produktionsstätte, werden dann über das Contract Net Protocol und über Marktmechanismen verhandelt.

- Ebenso wird im von Funk [Fun 99] vorgestellten Multiagentensystem für die Transportlogistik ein erweitertes Verfahren des Contract Net Protocols verwendet.
- Stiefbold [Sti 98] konzipiert ein Planungssystem für Logistikketten auf Basis von Software-Agenten. Dabei wird der BDI-Ansatz weiterverfolgt mit dem Ziel die Reaktionszeit entlang der Logistikkette zu verkürzen bei gleichzeitiger Synchronisation der Planung von Logistikvorgängen.
- In Schäfer [Sch 98a] wird ein Multiagentensystem entwickelt für den Bereich der Produktion. Es wird die Koordination der einzelnen Produktionskomponenten, wie Ressourcenmanagement, Produktion und Transport mit Hilfe von Agenten verbessert.

2.2.4 Verteilte Systeme

Verteilte Systeme als grundlegende Architektur für Multiagentensysteme wird von Tanenbaum [Tan 02] folgendermaßen definiert:

„Ein verteiltes System ist ein System, das auf einer Menge von Rechnern ausgeführt wird, die nicht über einen gemeinsamen Speicher verfügen, und das sich dem Benutzer wie ein einzelner Rechner darstellt.“

Daraus ergeben sich für den Einsatz von verteilten Systemen [Her 94], [Lan 80], [Tan 94] verschiedene Randbedingungen, wie

- die Unterteilung komplexer Systeme in einfachere Subsysteme mit dem Ziel, eine höhere Stabilität und Wartbarkeit durch Senkung der Komplexität der Teilkomponenten zu erreichen.
- die Auslagerung von immer wieder kehrenden gleichartigen Aufgaben mit dem Ziel, diese Teilsysteme nur einmal entwickeln zu müssen und immer wieder nutzen zu können.
- die Unterteilung von Aufgaben, deren Berechnung komplex und langwierig ist, auf kleinere Aufgabenblöcke, die parallel bearbeitet werden können.

Dabei wird in der parallelen Programmierung in der Informatik zwischen sequentiellen Programmen, also Programme deren Anweisungen nacheinander abgearbeitet werden, und nebenläufigen Programmen, also Programme, die parallel oder quasi-parallel ablaufen, unterschieden. Die nebenläufigen Programme gliedern sich in folgende Klassen:

- **Multiprogramming** (Dijkstra [Dij 68]): Prozesse nutzen einen gemeinsamen Prozessor, d. h. sie werden quasi-parallel abgearbeitet.
- Prozesse nutzen unterschiedliche Prozessoren, d. h. werden echt parallel ausgeführt.
    - **Multiprocessing** (Jones [Jon 80]): Prozessoren haben einen gemeinsamen Speicher.

- **Parallel Processing**: Prozessoren haben keinen gemeinsamen Speicher und sind über ein schnelles, integriertes Interkonnektionsnetzwerk verbunden.
- **Distributed Processing**: Prozessoren haben keinen gemeinsamen Speicher und sind über ein Rechnernetz (LAN, MAN, WAN) miteinander verbunden. Diese Klasse bildet die Grundlage für die Agententechnologie.

Unabhängig von der Wahl des eingesetzten Verfahrens ist immer, unter der Randbedingung, dass zwei oder mehrere Prozesse an ein und derselben Aufgabe bzw. auf denselben Daten arbeiten, eine Synchronisation der Prozesse notwendig [Fei 95], [Shi 95]. Dies kann durch die Benutzung eines gemeinsamen Speichers oder durch den Austausch von Nachrichten geschehen. Die Benutzung eines gemeinsamen Speichers setzt voraus, dass die verteilte Anwendung sich auf einem Rechnersystem befindet.

In den verteilten Systemen werden zwei prinzipielle Architekturen unterschieden:

- **Client/Server**: Unter den Client/Server – Systemen [Gei 95] sind alle die Systeme zu verstehen, bei denen ein Server einen Dienst anbietet und ein Client diesen Dienst nutzt. Dabei kann ein Client die Dienste von verschiedenen Servern in Anspruch nehmen. Das Client/Server – Prinzip kann auch hierarchisch angewandt werden, sodass ein Server als Client die Dienste eines anderen Servers benutzt. Die Client/Server – Systeme verfolgen ein einfaches 1-zu-1 Kommunikationsschema. Der Client stellt eine Anfrage an den Server, der diese bearbeitet und beantwortet. Ein Beispiel hierfür ist ein Webserver (Server), der als Dienstleistung Webseiten anbietet und der Browser (Client), als Dienstnutzer, diese darstellt.
- **Verteilte Systeme**: In den verteilten Systemen [Tan 02] gibt es keine direkte Aufteilung von Dienstanbieter und Dienstnutzer. Die beteiligten Komponenten agieren lösungsorientiert und können dabei sowohl als Dienstanbieter wie auch als Dienstnutzer fungieren. Dabei ist zu berücksichtigen, dass im Vergleich zur Client/Server – Architektur eine beliebige Kommunikation, auch m-zu-n – Kommunikation, gewählt werden kann. So ist es möglich, dass jede Komponente des verteilten Systems zu jedem Zeitpunkt eine Kommunikation initiieren kann, die unter Umständen an mehrere Ziele gerichtet ist oder Antworten von mehreren Zielen erwartet [Agh 99].

In Navarro [Nav 01] wird eine agentenbasierte Service Brokering Architektur für Multiservice Netze beschrieben, die auf dem Client / Server – Prinzip beruht. Einzelne Agenten holen die Informationen von den einzelnen Serviceanbietern ab und andere Agenten bieten die Verwaltung der Services als Middleware den Clients an.

In Cuena [Cue 99] ist eine Agentenarchitektur, dem verteilten Ansatz folgend, entwickelt worden, um den Menschen bei Entscheidungen zu unterstützen, die auf ei-

ne Vielzahl von unterschiedlichen Informationen beruhen, die räumlich verteilt in unterschiedlichen Datenbanken hinterlegt sind. Dabei gibt es Agenten, die Fakten sammeln, andere bewerten diese und die dritte Klasse bestimmt einen oder mehrere Vorschläge.

2.2.5 Repräsentations- und Anfragesprachen

Repräsentationssprachen

Im Datenbankumfeld wird zwischen dem relationalen und dem objektorientierten Paradigma unterschieden. Eine weit verbreitete Anfragesprache für relationale Datenbanken ist die Structured Query Language (SQL) [SQL 93] und für die objektorientierte Welt die Object Query Language (OQL) [Cat 00], [Dou 98]. Diese Anfragesprachen stellen ausdrucksmächtige, mengenorientierte Datenbankschnittstellen zur Verfügung und können sowohl selbständige Sprachen für Ad-hoc-Anfragen sein, als auch in einer bestimmten Wirtssprache eingebettet werden [Här 99]. Beiden Anfragesprachen ist eines gemeinsam, sie dienen ausschließlich zur Informationsbeschaffung unter der Voraussetzung, dass das Datenmodell zum Zeitpunkt der Anfrage bereits bekannt ist. Ein Navigieren oder eine Anfrage mit unvollständigem Wissen ist nicht möglich.

Papadimitriou [Pap 95] stellt die Frage, ob das relationale Modell als das einzig richtige Paradigma angesehen werden kann. Dabei wird die Verschiebung der Datenbank-Theorie zu verschiedenen Modelle als eine wissenschaftliche Aktivität angesehen. Das semistrukturierte Datenmodell wird als das Nachfolge-Modell in diesem Bereich angesehen. In Roddick [Rod 00] wird ebenfalls die Evolution im Datenmanagement behandelt, und zwar in dem Kontext von *Was*, *Warum*, *Wo*, *Wann*, *Wer* und *Wie*. Zusätzlich zu den traditionellen Datenbank-Management-Systemen werden Operationen bei den Anwendungen von XML für Datentransformationen, Anfrage Umsetzung, Datentransport und stream-basiertes Processing angeboten [Suc 01].

Aus dem Grund, Vorwissen über das Datenmodell besitzen zu müssen und zur Abstrahierung der darunter liegenden Datenbanktechnologie, wurden Repräsentationssprachen entwickelt mit deren Hilfe Informationen strukturiert dargestellt werden können.

XML [Bra 04] ist ein Standard, der beim Austausch von Daten nur die Interpretation der Daten definiert, ohne Dateiformate oder Programmierschnittstellen zu definieren und somit eine Lösung für den Datenaustausch zwischen inkompatiblen Systemen bietet. XML wird häufig eingesetzt, wenn es um Datenmanipulation und -übertragung sowie Verwaltung von semistrukturierten Daten und Datenintegration geht, da mit Hilfe der Schema Sprache Data Type Definition (DTD) die Struktur des XML-Dokuments vorgegeben wird und damit auch Wissen über den semantischen Inhalt der Information hinterlegt ist.

Resource Description Framework (RDF) [Las 99], [Bri 00] baut auf XML auf. RDF zeichnet sich dadurch aus, dass neben der reinen Information auch Strukturinformation abgelegt werden kann. So werden mit Hilfe eines Graphen die strukturellen Zusammenhänge der Informationen abgebildet.

Anfragesprachen

Unter den Anfragesprachen werden Sprachen verstanden, mit deren Hilfe Informationen aus einem vorgegebenen Datenbestand extrahiert werden können. Diese Sprachen orientieren sich dementsprechend an der eingesetzten Datenbanktechnologie und an dem verwendeten Datenmodell. Das Datenmodell gibt dabei Auskunft in welchem semantischen Zusammenhang die Daten zueinander stehen.

Für die Weiterverarbeitung von Informationen, die in einer Repräsentationssprache formuliert sind, werden unter anderem Anfragesprachen benötigt. Diese verhalten sich zu den Repräsentationssprachen wie SQL zum relationalen Datenbankparadigma.

Aufbauend auf den XML-Dokumenten existieren eine Vielzahl von Anfragesprachen bzw. Erweiterungen des vorgeschlagenen Standards XML-QL [Deu 98] und XQuery als Erweiterung von XML-QL des W3-Konsortiums [Boa 03]. Dabei wird neben der Suche auf XML-Dokumente auch die Struktur der XML-Dokumente berücksichtigt. So ist mit Hilfe von For-Let-Where-Or-Return (FLWOR)-Ausdrücken [Moe 03] möglich neben der reinen Informationsbeschaffung, durch die Ausdrücke *FLW*, auch die Form des Rückgabewertes, durch den Ausdruck *R*, zu bestimmen [Liu 02].

Der XQuery Ansatz, der eine erweiterte Infrastruktur für Anfragen bereitstellt, wird auch von Fankhauser [Fan 02] befürwortet und durch einen XQuery-Prozessor erweitert, der eine Methode für die Auswertung der XQuery-Anfragen bereithält. Ähnliche Vorhaben der Implementierung eines XQuery-Prozessors nach der Spezifikation des W3-Konsortiums laufen bei Oracle [Ora 03] unter der Bezeichnung XML DB mit den dazugehörigen Schnittstellen OJXQI [Ora 04a], [Ora 04b]. Bei Lucent Technologies [Luc 03a] wird der Einsatz eines XQuery-Prozessors im Galax-Projekt beschrieben und Papakonstantinou [Pap 03] entwirft einen XQuery-Prozessor unter der Namen XMediator.

Fuhr [Fuh 01] entwirft XIRQL als eine Anfragesprache, die auf XML-QL aufbaut und in XPath mündet [Fuh 03]. Für den Entwurf dieser Sprache sind die Punkte Bewertung und Ranking, Relevanz-orientierte Suche, Datentypen mit vagen Prädikaten und semantischer Relativismus aufgegriffen worden. Einige dieser Punkte wurden in der Spezifikation von XQuery in Form des Standards XPath [Ber 03], [Fer 03b], [Dra 03], [Mal 03] berücksichtigt.

Die Anfragesprache LOREL [Abi 97] baut auf dem Object Exchange Model (OEM) auf. Diesem Modell liegt eine objektorientierte Datenbank zugrunde, sodass LOREL ein Vertreter ist, der zu der Klasse der OQL-Sprachen gehört. In Goldman [Gol 99] sind Erweiterungen bzw. Anpassungen von LOREL beschrieben, sodass das darunter liegende OEM an XML angeglichen werden kann.

In Fernandez [Fer 99] wird ein Vergleich unter anderem zwischen XML-QL und LOREL gezogen, während in Bonifati [Bon 01] eine technische Untersuchung zwischen den sechs Schemasprachen XML-DTD [Bra 00], XML-Schema [Bir 00], [Tho 04], RELAX [Mur 00], [ISO 22250], SOX [Dav 99], Schematron [Jel 00], [Mil 00], DSD [Kla 99], [Kla 00] und den sechs Anfragesprachen LOREL [Abi 97], XML-QL [Deu 98], XML-GL [Cer 99], XSLT [Cla 99], XQL [Rob 99], Quilt [Cha 00] vorgenommen wird.

Diverse Erweiterungen wurden für XML-Anfragesprachen vorgenommen, unter anderem wurde in Fuhr [Fuh 00] die Sprache XIRQL eine Erweiterung von XQL entwickelt, um ein XML-Dokument Strukturkenntnisse durchsuchen zu können. Des Weiteren wurde von Liu [Liu 01] eine regelbasierte Anfragesprache für XML entworfen. Basierend auf XSLT wurde in Moerkotte [Moe 00] YAXQL entwickelt, um im XML-Format vorliegende Webseiten von verschiedenen Servern inhaltlich aggregieren zu können.

Mit dem Ziel, die elementaren Zugriffsmechanismen zu erweitern und Lösungen für spezielle Anwendungsfälle zu finden, beschäftigen sich zahlreiche Forschungsaktivitäten, die im Folgenden kurz beschrieben werden:

- In Karvounarakis [Kar 02] wird RQL als RDF Query Language vorgestellt, die sowohl in der Lage ist, Informationen zu finden, dabei aber auch die Möglichkeit bietet Strukturinformationen in die Suche mit einfließen zu lassen bzw. die Struktur über die Suche zu erkunden.
- Baeza-Yates [Bae 02] beschreibt ein Verfahren, wie die Suche im XML-Dokument beschleunigt und verbessert werden kann, sodass auch Anfragen mit navigierendem Charakter möglich sind. So kann z. B. nach dem Über-Dokument-Knoten eines gefunden Dokument-Knoten gesucht werden. Hierzu wird das XML-Dokument in einer Baumstruktur abgebildet.
- Özsu [Özs 95] stellt einen Ansatz für Mechanismen zur Bearbeitung von Anfragen in objektorientierten Datenbanken vor. Im Rahmen des GENIAL-Projekts wird ein intelligenter Zugriff auf ingenieurtechnische Informationen erarbeitet. In einer verteilten Architektur werden unter anderem Wissensakquisition-, Such- und Präsentationskomponenten integriert.
- Einen Ansatz für eine wissensbasierte Dokumentenanfrage in Büroumgebungen wird in Celentano [Cel 95] vorgestellt. Hierbei werden Retrieval-Methoden beschrieben, die Dokumente aufgrund einer semantischen Beschreibung des Inhalts finden. Intelligente Zugriffsmechanismen zu internen und externen Entwicklungsinformationen sind in Radeke [Rad 98] dargestellt.

- In Mylopoulos [Myl 96] wird unter anderem die Schnittstelle des Knowledge Base Management Systems Telos vorgestellt. Neben einer hypertextbasierten Benutzungsschnittstelle existiert auch eine Schnittstelle für Programmiersprachen. Schnittstellen auf dieser Ebene werden auf eine logische Ebene abgebildet, in der einfache Operationen der Wissensbasis zur Definition von Klassen, Regeln und Constraints existieren. Durch einen Anfrageoptimierer werden schließlich optimale Ausführungsstrategien für Anfragen gefunden.
- In Savnik [Sav 99] wurde eine Anfrage-Algebra für komplexe Objekte entwickelt. Die Motivation für diesen Ansatz war eine Untersuchung der Operationen, die notwendig sind zur Abfrage der Struktur von komplexen Objekten in objektorientierten Datenbanken. Durch die vorgestellte Algebra wird zum einen die Möglichkeit zum Stellen von Anfragen für das konzeptionelle Schema eröffnet und zum anderen die Untersuchung der Bestandteile komplexer Objekte erlaubt.

Erweiterte Anfragesprachen

Analog zu probabilistischen Informationsanfragen ist das Ziel bei der Durchführung der unscharfen Suche die richtige Einschätzung, wie relevant ein bestimmtes Dokument oder eine bestimmte Information für einen Benutzer mit einer bestimmten Anfrage ist und wie stark ein vorhandener Wert mit einer formulierten Anfrage übereinstimmen muss. Für Mechanismen zur Suche nach *ähnlichen* Entwicklungsergebnissen können teilweise auch Erkenntnisse von Datenbanksystemen aus anderen Anwendungsbereichen verwendet werden. In Faloutsos [Fal 94] wird beispielsweise ein Ansatz vorgestellt, in dem Datenbankanfragen nach dem Inhalt von Bildern (Farbe, Struktur und Form von Objekten, etc.) umgesetzt werden. Einen Ansatz für probabilistischen Information Retrieval wird in Fuhr [Fuh 94] ausgeführt.

Der in Kamp [Kam 97] angegangene Ansatz erlaubt es unterschiedliche Ingenieurprobleme mit Hilfe von Beschreibungslogiken zu lösen. Die Abhängigkeiten der Wissenselemente werden über Attribute und Strukturen beschrieben und über Constraints verknüpft.

2.2.6 Das Aktive Semantische Netz

Das ASN ist die zentrale Datenhaltung des RPD. Wie in der Einführung und in der Problemstellung bereits erwähnt, soll die zu entwickelnde RPD-Middleware das Bindeglied zwischen ASN und RPD-Anwendungen darstellen. Für die Entwicklung der RPD-Middleware wird daher die genaue Kenntnis des Aufbaus und der Funktionsweise des ASN benötigt.

Die Entwicklung eines aktiven semantischen Netzes für das RPD anstatt eines klassischen Datenbankansatzes wurde aus verschiedenen Gründen im Sfb 374 verfolgt. Zum einen bieten semantische Netze eine höhere Flexibilität in der Datenstrukturierung. Beim klassischen Datenbankansatz wird während der Softwareent-

wicklung ein Datenmodell entworfen, das in ein Datenbankschema überführt wird. Im RPD kann sich aber das Datenmodell durch den Einsatz neuer Verfahren und Werkzeuge ändern. Semantische Netze sind in der Lage diese dynamischen Änderungen des Datenmodells abbilden zu können.

Zum anderen können in semantischen Netzen Zusammenhänge detaillierter abgebildet werden, als in Datenbanken. So ist z. B. eine Differenzierung von verschiedenen Relationstypen möglich.

Durch die Entwicklung des ASN im RPD wurde es möglich mehr Funktionalität als lediglich die Manipulation von Daten bereitzustellen. So wurde eine aktive Komponente konzipiert, die es erlaubt Abhängigkeiten zwischen Informationen zu definieren und diese aufzulösen. So können im ASN Veränderungen eines Attributs bewertet werden und daraufhin andere Attribute direkt beeinflusst werden.

Diese Vorteile und die Aufteilung des ASN auf zwei Abstraktionsebenen stellen eine höhere Flexibilität gegenüber Datenbankansätzen dar und können effektiv im RPD eingesetzt werden. Die in der Problemstellung (siehe 2.1) genannten Nachteile des ASN werden durch die Entwicklung der RPD-Middleware im Rahmen dieser Arbeit angegangen. In der ersten Abstraktionsebene wird die Funktionalität, d. h. die Programmlogik des ASN abgelegt, sowie die Struktur der Datenablage definiert. Die zweite Abstraktionsebene ist die Instanzebene, die die eigentlichen Informationen beinhaltet. Die semantischen Zusammenhänge der Informationen sind über ein objektorientiertes Datenmodell beschrieben.

Strukturebene

Auf der Strukturebene wird definiert, wie die Daten abgelegt werden. Es wurde hierfür ein flexibles Grundgerüst entworfen, das es erlaubt, Daten zu strukturieren und das Datenmodell beliebig zu erweitern. Es handelt sich hierbei nicht um ein klassisches Datenmodell, wie es in der herkömmlichen Softwareentwicklung entworfen wird, sondern um ein Metamodell. Mit Hilfe des Metamodells (siehe Bild 8) kann ein klassisches Datenmodell abgebildet werden. Die Basis des Metamodells umfasst sieben Objekte von denen die ersten vier im weiteren Verlauf dieser Arbeit von Bedeutung sind und die Elemente darstellen, die für die Abbildung eines Datenmodells im herkömmlichen Sinne benötigt werden. Dabei ist das ASN hierarchisch über die Elemente *Net*, *Concept*, *Attribute* aufgebaut [Bul 04a], [Bul 04b], [Rol 02a], [Rol 02b].

- **Net**: Die Klasse *Net* des Metamodells beschreibt ein aktives semantisches Netz. Das Netz beinhaltet über die Aggregation alle benötigten Informationen um ein Netz abzubilden und stellt die oberste Hierarchieebene dar.
- **Concept**: Die Klasse *Concept* steht für einen Begriff oder eine Informationseinheit in einem bestimmten aktiven semantischen Netz. Der Begriff der Klasse aus dem herkömmlichen Datenmodell ist mit dem Begriff Konzept

vergleichbar. Die Konzepte sind typisiert, sodass anhand des Typs entschieden werden kann, welche Attribute das Konzept besitzen soll. Die Klasse *Concept* bildet damit die mittlere Hierarchiestufe.

- **Attribute**: Mit Hilfe der Klasse *Attribute* werden die Attribute der Konzepte realisiert. Durch die Aggregation zwischen *Concept* und *Attribute* ist es möglich, zur Laufzeit beliebig viele Attribute dem Konzept hinzuzufügen bzw. zu entfernen. Dies ist ein Unterschied zum klassischen Datenmodell, wo die Attribute fest in der Klasse verankert sind und nicht zur Laufzeit angepasst werden können. Der Typ des Attributs gibt an, welche Art von Wert im Attribut gespeichert ist, beispielsweise sagt der Typ *Integer* aus, dass es sich bei dem Wert des Attributs um ganze Zahlen handelt. Die Klasse *Attribute* ist die unterste Hierarchieebene.
- **Relation**: Die Klasse *Relation* dient zur Vernetzung. Hier werden die Informationselemente (Konzepte) miteinander verknüpft. Dabei ist zu beachten, dass es sich um gerichtete Verknüpfungen handelt. Die Relationen repräsentieren damit die Verbindungen zwischen Klassen im herkömmlichen Datenmodell. Die Relationen sind ebenfalls wie die Konzepte und Attribute typisiert mit dem Ziel die Art der Relation zu spezifizieren, d. h. hier wird spezifiziert, ob es sich bei einer Relation beispielsweise um eine Komposition oder eine Assoziation handelt.
- **ID**: Diese Klasse ist eine Hilfsklasse und wird lediglich benötigt, um alle Instanzen der Klassen *Interface*, *Net*, *Concept*, *Attribute* und *Relation* mit einer eindeutigen Bezeichnung auszustatten.
- **Interface**: Die Klasse *Interface* ist die Schnittstelle zwischen dem ASN und der RPD-Anwendung. Da es sich beim ASN um eine Client/Server – Applikation handelt, stellt die Klasse *Interface* den serverseitigen Anteil dar.
- **Client**: Hierbei handelt es sich um den clientseitigen Anteil der Schnittstelle zwischen der RPD-Anwendung und dem ASN.

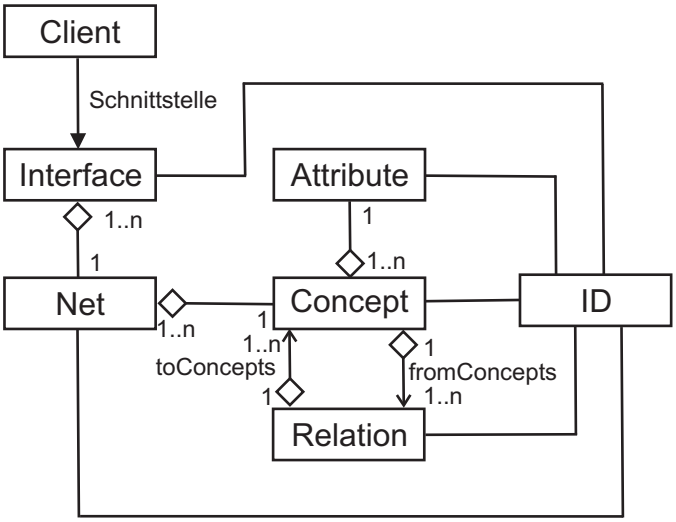


Bild 8: Aufbau des ASN auf Strukturebene

Um auf der Instanzebene die Informationen eindeutig ansprechen zu können, ist ein strenges Namenskonzept notwendig. So besitzen alle Netze einen eindeutigen Namen über den sie angesprochen werden können. Dasselbe gilt für die drei Elemente *Concept*, *Attribute* und *Relation*. Lediglich der Bereich in dem der Name eindeutig sein muss unterscheidet sich. So muss ein Konzept eindeutig sein innerhalb des Netzes zu dem es gehört, während ein Attribut eine eindeutige Auszeichnung im Rahmen des ihm übergeordneten Konzepts besitzen muss. Die Eindeutigkeit der Relation wird durch das Konzept bestimmt von dem die Relation ausgeht. Dieses Konzept wird auch als ausgehendes Konzept bzw. Startkonzept bezeichnet.

In Bild 9 ist ein Ausschnitt aus dem Datenmodell des RPD zur Verdeutlichung der Zusammenhänge zwischen dem ASN-Metamodell und dem klassischen Datenmodell dargestellt.

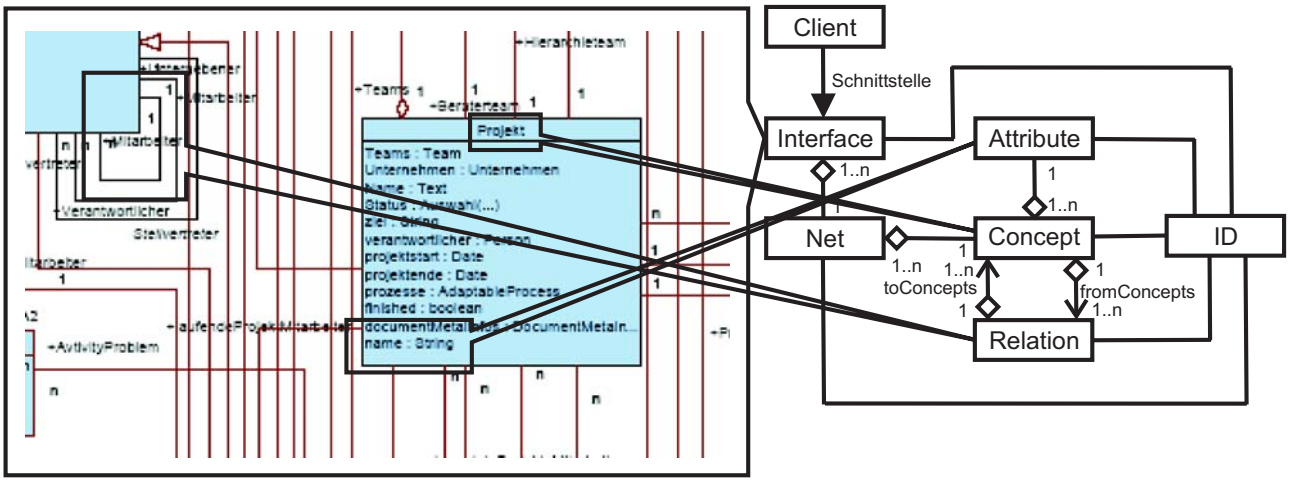


Bild 9: Verknüpfung Datenmodell und Metamodell

Instanzebene

Im Gegensatz zur Strukturebene werden auf der Instanzebene die Informationen in das ASN geschrieben. Es werden dabei Netze instanziiert, Konzepte mit ihren Attributen hinzugefügt und über Relationen untereinander vernetzt. So gehört ein instanziiertes Attribut genau einem Konzept an und dieses genau einem Netz. Die Angaben Netz, Konzept, Attribut sind notwendig um den Wert eines Attributs auszulesen.

Zum Beispiel beinhaltet auf der Struktur-Ebene das Netz für die Entwicklung der Luftdüse ein Konzept vom Typ *Person*, das neben anderen Attributen auch die Attribute *Vor-* und *Nachname* besitzt. Diese Personen gehören dann einem Team an, das ebenfalls durch ein Konzept repräsentiert wird und mindestens das Attribut *Name* mit dem Teamnamen besitzt (siehe Bild 10).

Auf der Instanz-Ebene kann dies z. B. eine Instanz Netz mit dem Namen *Luftdüse*, eine Instanz Konzept mit dem Namen *Person1* und den Attributen Vorname *Karl* und Nachname *Mayer* sowie eine Instanz Konzept vom Typ Team mit dem Namen *Entwicklung* sein. Des Weiteren wird eine Relation *Teamzugehörigkeit* zwischen dem Team und der Person hergestellt.

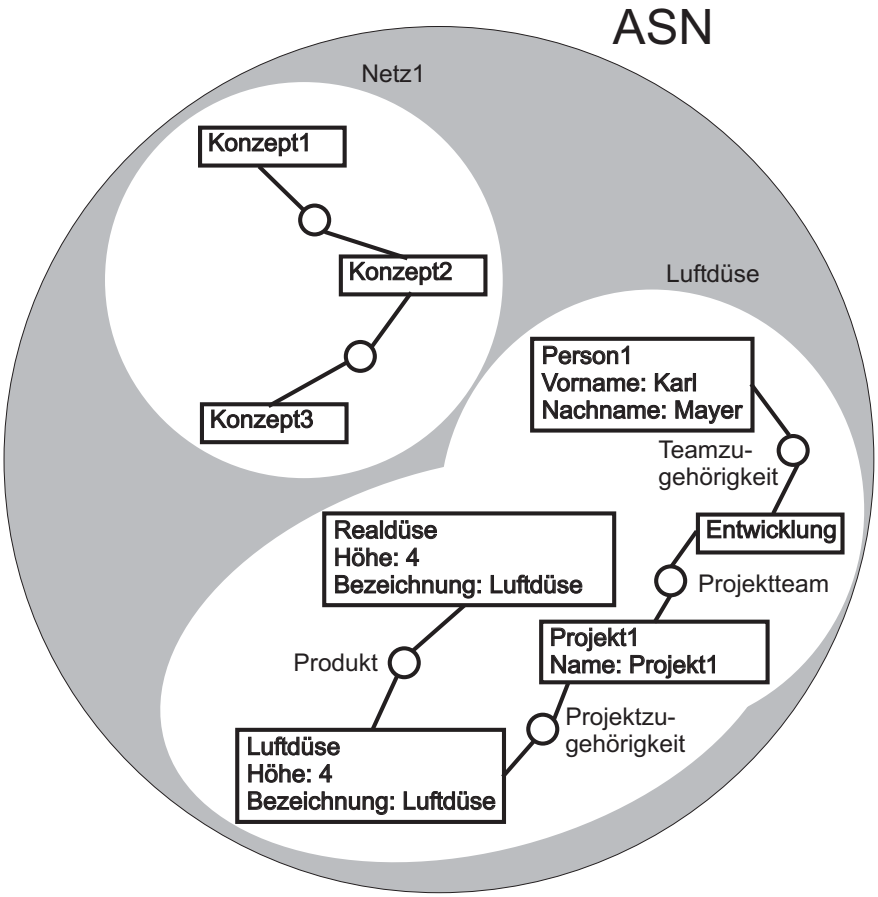


Bild 10: Struktur des ASN auf Instanzebene

3 Zielsetzung und Anforderungen an die RPD-Middleware

3.1 Zielsetzung

Für die Informationsbeschaffung und -aufbereitung von Wissen aus dem ASN wird eine auf das RPD abgestimmte Anfragesprache benötigt. Diese muss einerseits den Anforderungen, die an das darunter liegende Repräsentationsmodell gestellt werden, aber auch speziellen Anforderungen, die aus dem RPD-Prozess entstehen, genügen. Die ausgewählte Repräsentationsform spielt eine entscheidende Rolle für den Entwurf einer entsprechenden Anfragesprache. Daher muss die Anfragesprache zur Informationsbeschaffung und -aufbereitung für die RPD-Middleware das Repräsentationsmodell des Aktiven Semantischen Netzes berücksichtigen. Durch die proprietäre Struktur des ASN (siehe Abschnitt 2.2.6) ist der direkte Einsatz von Anfragesprachen, wie XQuery, XML-QL, LORAL oder ähnliches nicht möglich, da sie auf Standard-XML bzw. Erweiterungen von XML aufbauen. Des Weiteren sind nur in XML-QL Grundfunktionalitäten für die Informationsaufbereitung vorhanden. Daher ist eine Anfragesprache und die dazugehörige Funktionalität als Methode für die Informationsbeschaffung und -aufbereitung und die benötigte Interpretationslogik zu entwickeln.

Die Flexibilität von XML und die bereits geleisteten Arbeiten für die Suche in XML-Dokumenten führen zu einer genaueren Untersuchung, ob eine Transformation der ASN-Struktur in eine XML-Struktur zu weiteren Bearbeitung für die Informationsbeschaffung und -aufbereitung zielgerichtet ist.

Bekannte Anfragesprachen gehen von einer wohl definierten Datenstruktur aus. Im Falle des ASN existieren zwei Abweichungen. Zum einen kann das Datenmodell zur Laufzeit erweitert werden und zum anderen können die Informationen je nach RPD-Domäne unterschiedlich abgelegt sein. Dies impliziert die Forderung nach unscharfen Suchmethoden, d. h. Suchmethoden für die kein Wissen über das Datenmodell bestehen muss, und Navigationsmöglichkeiten auf dem ASN bzw. auf der Ergebnismenge der Suchanfrage. Diese Anforderungen werden von den untersuchten Anfragesprachen nicht unterstützt und müssen neu entwickelt werden.

Im Bereich der verteilten Systeme existiert für die Kommunikation zwischen beteiligten Partnern unter anderem die eventbasierte Kommunikation. Diese stellt eine viel versprechende Möglichkeit dar, Veränderungen des ASN aktiv an die RPD-Anwendung propagieren zu können. Sie ist damit Bestandteil der aktiven Informationsüberwachung, die wie bereits in der Problemstellung erwähnt (siehe Abschnitt 2.1.3) einen wichtigen Beitrag zur Interaktion von RPD-Experten leistet.

Eine eventbasierte Kommunikation ist für die aktive Informationsüberwachung aber nicht ausreichend. Es müssen die Veränderungen zuerst im ASN erkannt werden. Hierzu sind Standardmechanismen wie die Abfrage von Werten mittels einer Anfra-

gesprache aufgrund der proprietären Struktur des ASN nicht möglich. Dies muss von der Methode zur Informationsüberwachung berücksichtigt werden. Dabei sind auch die Spezialitäten des ASN, wie strukturelle Veränderungen zur Laufzeit, zu berücksichtigen. So sind nicht nur die Veränderungen im ASN hinterlegter Informationen für den RPD-Prozess von Bedeutung, sondern auch strukturelle Veränderungen, wie das Löschen, das Verknüpfen von Informationen, sowie statistische Informationen, wie die Anzahl der Lese- und Schreibzugriffe bzw. die Version eines Informationselements, wie auch die Einflechtung von einfachen mathematischen Funktionen.

Bei der Betrachtung existierender aus der Agententechnologie bekannter Abstimmungsansätze, wie Blackboard, Marktmechanismus und Verhandlung oder auch Vertragsmechanismen wird deutlich, dass bei der Implementierung fest vorgegebene Verhandlungsstrategien diesen Ansätzen zugrunde liegen. Der RPD-Prozess wird während seiner Abarbeitung laufend den neuen Gegebenheiten angepasst. Ein starr vorgegebenes Koordinationsprotokoll ist damit unzureichend. Die zu entwickelnde Koordinationskomponente hat dies zu berücksichtigen und muss ein flexibles, frei definierbares Koordinationsprotokoll erlauben. Um die Koordinationspartner in den Koordinationsablauf integrieren zu können, ist Kommunikation ein wichtiges Ziel. Wie bei der aktiven Informationsüberwachung stellt eine eventbasierte Kommunikation eine gute Möglichkeit dar.

Des Weiteren können einzelne Koordinationsvorgänge über die gesamte Laufzeit des RPD-Prozesses stattfinden. Dies bedeutet, dass der Koordinationsmechanismus über einen langen Zeitraum aktiv sein muss und sich fehlertolerant gegenüber beendeten und erneut gestarteten bzw. nicht aktiven RPD-Anwendungen verhalten muss.

Aufgrund der beschriebenen Problemstellung, den besonderen Vorgaben durch das ASN und den integrativen Anforderungen des RPD und deren Anwendungen wird eine auf das RPD zugeschnittene Middleware benötigt. Hierbei hat sich gezeigt, dass sich besonders Agentensysteme durch ihren verteilten und flexiblen Ansatz eignen. Die in der Literatur vorhandenen Agentensysteme sind entweder auf ein Einsatzgebiet, wie beispielsweise Robotik, Routenplanung, Kapazitätsplanung beschränkt, und berücksichtigen dabei keine domänenübergreifende Aspekte. Bei einer weiteren Klasse von Agentensystemen handelt es sich um Framework-Ansätze, die den Fokus auf die Infrastruktur des Agentensystems wie Kommunikation und Koordination von Agenten legen, aber nicht mit der konkreten Ausgestaltung der einzelnen Agenten (siehe Anhang A). Der Einsatz eines Agentensystems in einem komplexen und vielschichtigen Umfeld, wie dem RPD, ist neuartig.

Trotzdem zeichnen sich Multiagentensysteme durch freie und flexible Ausgestaltungsmöglichkeiten und durch eine klare Trennung von Kommunikation und Funktion aus. Sie stellen damit die Grundlage für die RPD-Middleware dar. Besonders durch die nachrichtenbasierte Kommunikation wird es möglich Informationen zwi-

schen den RPD-Anwendungen und der RPD-Middleware auszutauschen, die auf Seiten der RPD-Anwendung die Form von Events annehmen.

Durch die Verwendung eines Multiagentensystems wird die Komplexität der Aufgaben durch Aufteilung auf einzelne spezialisierte Agententypen, dem Prinzip *Teile-und-Herrsche* folgend, gesenkt. Zusätzlich werden die Forderungen nach einer programmiersprachenunabhängigen Schnittstelle, sowie nach netzwerkweiter Verfügbarkeit der Dienstleistungen der RPD-Middleware, realisierbar. Sowohl die Anforderung der netzwerkweiten Verfügbarkeit der RPD-Middleware, als auch die Vielzahl der parallel verlaufenden Kommunikationsbeziehungen stellen eine Herausforderung an die Stabilität und Fehlertoleranz der RPD-Middleware. So kommt bei den untersuchten Kommunikationstechniken nur die Multicast-Technologie in Frage. Unicast scheidet aufgrund ihrer Begrenzung in der Anzahl ihrer Kommunikationsbeziehungen aus. Die Broadcast-Technologie ist wegen ihren räumlichen Restriktionen nicht nutzbar. Den Schwächen der Multicast-Technologie bezüglich Zustellungsgarantie muss die Architektur der RPD-Middleware Rechnung tragen.

Der Ansatz für die Steuerung des der RPD-Middleware zugrunde liegenden Agentensystems ein Blackboard einzusetzen, ist aus Stabilitätsgründen nicht sinnvoll. Auch ein klassischer Client/Server – Ansatz stellt durch den Server einen Single-Point-of-Failure dar und ist damit für die RPD-Middleware nicht geeignet. Es ist nach einem verteilten Ansatz zu suchen, der die Agenten der RPD-Middleware mit einschließt, um Fehler frühzeitig erkennen und melden bzw. beheben zu können. Der Ansatz muss auch die Adressierung der Agenten berücksichtigen. Das Verfahren der Aufteilung auf eine innere und äußere Sprache innerhalb der Agentenkommunikation, wie er in KQML/KIF und von der FIPA verfolgt wird, kann zugeschnitten auf das RPD verwendet werden.

In Tabelle 1 sind Bezug nehmend auf den Stand der Technik (siehe Abschnitt 2.2) die Anwendbarkeit und die Defizite tabellarisch zusammengefasst.

		Informations-beschaffung und -aufbereitung	Informations-überwachung	Koordination des RPD-Prozess
Anfragesprachen	SQL	◔	◔	○
	OQL	◑	◔	○
	XML-QL	◑	◔	○
	XQuery	◑	◔	○
	XIRQL	◔	◔	○
	XSLT	◑	◔	○
	XQL	◑	◔	○
	LOREL	◑	◔	○
	RQL	◑	◔	○
Repräsentations-sprachen	RDF	●	●	●
	XML	●	●	●
Verteilte Systeme	Multipro-gramming	◑	◔	◔
	Client / Server	◑	◑	◑
	Verteilte Systeme	●	●	●
Kommunikations-techniken	Unicast	◑	◑	◑
	Broadcast	◔	◔	◔
	Multicast	●	●	●
Kommunikations-sprachen	KQML / KIF	◑	◑	◑
	FIPA ACL	◑	◑	◑
	ICL	◑	◑	◑

		Informations-beschaffung und -aufbereitung	Informations-überwachung	Koordination des RPD-Prozess
Agenten-architekturen und -modelle	Reaktive Agenten	●	●	●
	Proaktive Agenten	●	●	●
	BDI-Ansatz	◑	◑	◑
	Hybride Modelle	●	●	●
	Modell nach [Bia 97]	◑	◑	◑
	Modell nach [Fat 98]	●	●	●
Koordinations-protokolle	Kooperations-protokoll	●	●	●
	Vertrags-protokoll	◑	◔	○
	Blackboard-Ansatz	○	◔	◑
	Verhandlungs-ansatz	○	○	●
	Marktmechanis-mus	◔	◔	◔
Koordinations-techniken	Persuasion	○	○	◔
	Negotiation	○	○	◔
	Inquiry	●	○	◔
	Deliberation	◑	○	◔
	Information Seeking	◑	○	◔
Legende:	○=nicht relevant	◔=nicht ge-eignet	◑=teilweise geeignet	●=geeignet

Tabelle 1: Bewertung des Stands der Forschung

3.2 Anforderungen an die RPD-Middleware

Die Identifikation der benötigten RPD-Agententypen für die RPD-Middleware erfolgt auf Basis der Problemstellung (siehe Abschnitt 2.1). Hierzu wird überprüft, ob es möglich ist für die geforderten Themenstellungen

- Informationsbeschaffung und -aufbereitung,
- Informationsüberwachung und
- Koordination des RPD-Prozesses

jeweils einen RPD-Agententypen zu entwickeln. Allgemein ist festzustellen, dass bedingt durch die unterschiedlichen Vorgänge der drei Themenfelder verschiedene Anforderungsprofile entstehen, die neben der unterschiedlichen Funktionalität auch verschiedene Spezifikationssprachen für die jeweilige Aufgabe mit sich bringen, sodass eine Aufteilung der RPD-Agententypen auf diese drei Themengebiete als sinnvoll erscheint. Bei der genauen Betrachtung der drei Themenfelder ergibt sich folgendes Bild [Dal 05]:

Bei dem Vorgang der Informationsbeschaffung und -aufbereitung handelt es sich um einen zusammengehörenden Prozess. Der Informationsfluss dieses Prozesses besitzt allerdings zwei Richtungen, zum einen wird die Information aus dem ASN beschafft und zum anderen muss die Information in geeigneter Form der RPD-Anwendung präsentiert werden. Es liegt daher nahe, diesen informationsbeschaffenden Prozess in diese zwei Aspekte aufzuteilen und durch zwei getrennte, aber eng zusammenarbeitenden Agententypen informationstechnisch zu unterstützen. Ein weiterer Grund der Aufteilung ist die Tatsache, dass der Vorgang der Informationsbeschaffung eine andere Spezifikation benötigt als die Informationsaufbereitung. Hieraus resultieren zwei unterschiedliche Anfragesprachen für die jeweiligen RPD-Agententypen. Aus diesen Gründen muss ein Retrieval-Agent für die Informationsbeschaffung und ein Aggregations-Agent für die Informationsaufbereitung entwickelt werden.

Eine vollständige Entkopplung beider Agententypen verbietet sich, da ein Aggregations-Agent nur dann Informationen aufbereiten kann, wenn Informationen beschafft wurden. Dies bedeutet, ein Retrieval-Agent muss die aus dem ASN beschafften Informationen zur Aufbereitung an den Aggregations-Agenten übergeben, wodurch eine direkte Kommunikation zwischen beiden Agententypen stattfindet.

Das Themengebiet der Informationsüberwachung unterscheidet sich vom ersten Themenfeld dadurch, dass direkt keine Informationen zwischen ASN und RPD-Anwendung transportiert werden. Eines der Hauptargumente für die Entwicklung eines Mechanismus zur Informationsüberwachung ist in der Anforderung begründet, die RPD-Anwendung von der Aufgabe dem zyklischen Überprüfen auf Veränderungen im ASN zu entlasten. Der dafür zu entwickelnde Agententyp Monitor-Agent hat daher den Charakter der Präsentation von Informationsveränderungen und

nicht beispielsweise der Informationsaufbereitung, wie ein Aggregations-Agent. Zusätzlich ist die Aufgabenbeschreibung eines Monitor-Agenten geprägt durch die Überwachung genau definierter und bekannter Informationen.

Der Prozess des RPD, also die zeitlich vernetzte Abfolge der einzelnen Arbeitsschritte, die zu der Erstellung eines Prototyps benötigt werden, steht bei der Koordination im Vordergrund. Es findet daher keine Kommunikation mit dem ASN statt, wodurch eine Trennung von den anderen Agententypen sinnvoll ist. Da die Koordination als Verbindungsglied zwischen den RPD-Anwendungen betrachtet werden kann, ist es nicht ratsam diesen Vorgang in mehrere Agententypen aufzuteilen, sodass für dieses Themengebiet ein Koordinations-Agent als informationstechnische Unterstützung entworfen wird.

Die im folgenden Kapitel entwickelten Methoden für die identifizierten RPD-Agenten und ihre Einbettung in eine auf das RPD angepasste Middleware in Form eines Multiagentensystems fördert die Integration der RPD-Experten im vielschichtigen, domänenübergreifenden RPD-Umfeld. Es sind derzeit keine Produkte oder Technologien bekannt, die direkt den Anforderungen des RPD gerecht werden. Die im Stand der Technik (siehe Abschnitt 2.2) vorgestellten Mechanismen müssen auf das Umfeld des RPD angepasst werden.

In Tabelle 2 sind die Anforderungen an die RPD-Agenten tabellarisch zusammengefasst.

RPD-Agenten	Anforderungen
Retrieval-Agent (Informationsbeschaffung)	• Komplexe Anfragen an das ASN • Suchoperationen für die Suche ohne Kenntnis der Datenstrukturierung • Navigierende Mechanismen zur Erkundung des ASN • Suchmechanismen zum Auffinden von Informationen, die nicht direkt miteinander verknüpft sind • Einheitliche, programmiersprachen- und domänenunabhängige Anfragesprache
Aggregations-Agent (Informationsaufbereitung)	• Aufbereitung der abgefragten und aufgefundenen Informationen durch die von der RPD-Anwendung vorgegebenen Richtlinien • Zusammenfassung von beschafften Informationen aus dem ASN • Einheitliche, programmiersprachen- und domänenunabhängige Anfragesprache

RPD-Agenten	Anforderungen
Monitor-Agent (Informations-überwachung)	• Aktive Meldung von inhaltlichen und strukturellen Veränderungen des ASN • Meldung aller Veränderungen unabhängig davon wie kurzlebig diese waren • Einbeziehung von Informationen innerhalb und außerhalb des ASN in die Überwachung • Berücksichtigung von komplexen Vergleichen, sodass Ergebnisse von Vergleichen als Parameter von neuen Vergleichen benutzt werden können • Integration von einfachen, mathematischen Funktionen • Aufbau von einfachen Funktionen zur Überwachung von strukturellen Veränderungen • Einheitliche, programmiersprachen- und domänenunabhängige Anfragesprache
Koordinations-Agent (Koordination)	• Unterstützung von lang andauernden Koordinationsvorgängen • Frei definierbares Koordinationsprotokoll • Automatische Fortschrittsbenachrichtigung der an der Koordination beteiligten RPD-Anwendungen • Entwicklung einer einheitlichen Vorgehensweise, wie die Koordination stattzufinden hat. • Einheitliche, programmiersprachen- und domänenunabhängige Anfragesprache

Tabelle 2: Zuordnung der RPD-Agenten zu ihren Anforderungen

4 Entwicklung der RPD-Middleware-Methoden

4.1 Methode zur Informationsbeschaffung und -aufbereitung

Die Methode der Informationsbeschaffung und -aufbereitung legt die Vorgehensweise für die Wissensexploration fest. Dabei werden die Anforderungen aus Kapitel 3 und die besonderen Gegebenheiten durch das ASN (siehe Abschnitt 2.2.6) berücksichtigt.

Ausgangspunkt für die Entwicklung der Methode ist der Ist-Zustand ohne die RPD-Middleware. Hier ist es möglich, Einzelinformationen in Form von Konzepten und Attributen in das ASN zu schreiben bzw. aus dem ASN auszulesen. Des Weiteren können die Konzepte untereinander über Relationen semantisch verknüpft werden. Dabei ist zu beachten, dass das ASN eine Schnittstelle besitzt, die es nur erlaubt, Einzelinformationen auf einmal zu schreiben und zu lesen, d. h. in einem Schritt kann ein Attribut eines Konzepts geschrieben bzw. gelesen werden. Wird ein weiteres Attribut benötigt, so ist das eine weitere Anfrage an das ASN. Diese Vorgehensweise ist für eine RPD-Anwendung zu unflexibel und arbeitsintensiv.

Die erste Anforderung an die Methode ist die Nachbildung der derzeitigen ASN-Schnittstelle, das Lesen, Schreiben, Löschen und Anlegen von Netzen, Konzepten und Attributen, sowie das Verknüpfen bzw. das Lösen von Verknüpfungen zwischen Konzepten mit Hilfe von Relationen. Diese Nachbildung ist notwendig, da sie die Basis für die weitere Vorgehensweise ist. Sie kann nur eingesetzt werden, wenn der RPD-Anwendung das Datenmodell bekannt ist und damit die Fundorte im ASN genau spezifiziert werden können.

Ist der RPD-Anwendung das Datenmodell nicht bekannt, so kann sie sich die Informationen mühsam über die Verfolgung aller möglichen Relationen, ausgehend von einem beliebigen Konzept, heraussuchen. Dabei müssen alle Konzepte als Startkonzepte getestet werden, da von einem zusammenhängenden Netz nicht zwingend ausgegangen werden darf. Diese Vorgehensweise führt nur dann zum Ziel, wenn die RPD-Anwendung in der Lage ist, die gewonnenen Erkenntnisse geeignet umzusetzen. Dies ist aber nicht Aufgabe einer hoch spezialisierten RPD-Anwendung, wie beispielsweise einem Planungswerkzeug oder einer betriebswirtschaftlichen Software. Die Methode für die Informationsbeschaffung und -aufbereitung muss dafür eine Unterstützung anbieten.

Selbst wenn das Datenmodell der RPD-Anwendung bekannt ist, so ist es aufwendig, Informationen aus dem ASN abzufragen, sobald diese aus mehr als einem Informationselement bestehen, denn für jedes Informationselement muss eine Anfrage an das ASN geschickt werden. Dies bedeutet, wird ein RPD-Mitarbeiter mit seiner Adresse gesucht, so muss zuerst das Konzept *Person* vom ASN erfragt werden, dann einzeln die Attribute *Name* und *Vorname*. Anschließend wird über die Relation *Geschäftsadresse* das Konzept *Adresse* des RPD-Mitarbeiters ausgelesen und

dann wiederum einzeln die Attribute, *Firmenname*, *Strasse*, *PLZ*, *Ort*, *Telefonnummer*. Für dieses Beispiel werden bereits elf Anfragen benötigt. Das ASN bietet für solche Fälle eine Vereinfachung an, indem die Attribute eines Konzepts komplett ausgelesen werden können, dies senkt aber die Anzahl der Anfragen lediglich auf fünf.

Das Paradigma des ASN [Rol 02b] besagt, dass das ASN jederzeit erweiterbar ist, zum einen durch das Hinzufügen von Konzepten und zum anderen durch das Hinzufügen von Attributen zu Konzepten, bzw. durch die Bildung neuer Relationen zwischen Konzepten. Dies führt dazu, dass wenn eine RPD-Anwendung X ein neues Attribut einem Konzept hinzufügt dies der RPD-Anwendung Y völlig verborgen bleibt.

Dies hat Auswirkungen auf die Methode der Informationsbeschaffung und -aufbereitung. Durch Benutzung dieser Funktionalität des ASN wird das Datenmodell dynamisch verändert und ist somit nicht mehr jeder RPD-Anwendung bekannt. Eine dedizierte Anfrage auf Basis des Datenmodells ist damit nicht mehr möglich. Es müssen Anfragemechanismen vorgesehen werden, die eine Datenmodell-unabhängige Anfrage erlauben.

Eine weitere Anforderung an die Methode der Informationsbeschaffung und -aufbereitung resultiert aus der unterschiedlichen Art und Weise wie Daten strukturiert werden können. Je nach Anwendungsfall werden andere Attribute von Konzepten, aber auch andere semantische Zusammenhänge zwischen Konzepten, benötigt. Dies hat zur Folge, dass sich durch den Einsatz unterschiedlicher RPD-Anwendungen unterschiedliche Datenmodelle im ASN befinden, die unter Umständen teilweise ein- und dasselbe aussagen.

Eine funktionelle Erweiterung bei der Informationsaufbereitung ist die Möglichkeit, quantitative und qualitative Anfragen an das ASN stellen zu können. Derzeit ist es beispielsweise nicht möglich, das ASN direkt nach der Anzahl der Mitarbeiter eines Teams zu fragen. Die RPD-Anwendung muss dazu über das Team und der Relation zwischen den Konzepten *Team* und *Mitarbeiter* alle Mitarbeiter auffinden und diese selbständig aufsummieren.

Scharfe Suche

Die scharfe Suche stellt die Basisstrategie für die Informationsbeschaffung dar. Bei der scharfen Suche wird die Kenntnis über das Datenmodell und damit die genaue Struktur der Informationen vorausgesetzt.

Sie bildet die derzeitig existierende Schnittstelle des ASN nach. Die scharfe Suche beinhaltet damit Sprachelemente, die die Definition der einzelnen ASN-Elemente, wie Konzept, Attribut und Relation erlaubt, um deren Inhalt auslesen zu können.

Navigation

Mit Hilfe der Navigation wird das Umfeld des Ergebnisses der scharfen Suche erkundet. Dadurch ist es möglich, ausgehend von einem Konzept aus der Ergebnismenge, alle umliegenden Konzepte und deren Attribute auf einmal abzufragen. Dieses Umfeld kann dann durch die RPD-Anwendung ausgewertet werden und als Basis für weitere Suchvorgänge benutzt werden. Dabei muss die Datenmenge begrenzbar sein. Dies geschieht durch die Definition eines Umkreises um das Basiskonzept. Dieser Umkreis wird definiert durch die maximale Anzahl von Relationen, die ein Konzept vom Basiskonzept entfernt sein darf.

Aggregation mehrerer Anfragen

Der Forderung nach Mengenanfragen wird Genüge getan, indem das Ergebnis neben dem Konzept immer auch alle Attribute enthält, sowie alle Relationen, die von diesem Konzept abgehen. Dadurch wird die RPD-Anwendung in die Lage versetzt, die relevanten Informationen heraus zu filtern bzw. bei Bedarf Relationen weiterzuverfolgen. Die RPD-Anwendung erlangt mit einer Abfrage damit sowohl die benötigten Information als auch die Kenntnis über deren Strukturierung. Es genügt zum Beispiel nicht, eine Menge von Attributwerten an die RPD-Anwendung zurückzuliefern wie im ASN realisiert, es müssen auch die dazugehörigen Attributnamen passend zu den Werten der RPD-Anwendung präsentiert werden.

Für die Flexibilisierung der Anfrage werden Suchmechanismen entwickelt, die es erlauben, auch Konzepte aufzufinden, die nicht direkt durch eine Relation untereinander verbunden sind, sondern transitiv über mehrere Relationen hinweg, d. h. zwischen den beiden Konzepten existiert ein Pfad, der aus mehreren direkt miteinander verknüpften Konzepten besteht.

Des Weiteren sieht die Methode der Informationsaufbereitung ein Verfahren vor, das es erlaubt mehrere voneinander unabhängige Anfragen als eine Anfrage zusammenzufassen. Dieses Verfahren wird Aggregation genannt. Die daraus resultierenden Ergebnisse müssen ebenfalls zusammengefasst und gruppiert nach der Anfrage der RPD-Anwendung präsentiert werden (siehe Abschnitt 6.2). Dieses Verfahren muss sich an der Repräsentationsform der Ergebnisse der Suchstrategien orientieren.

Datenmodellunabhängige Anfragen

Die Methode der datenmodellunabhängigen Anfrage bietet die Möglichkeit, ohne Kenntnis der Relationen Ergebnisse zu finden. Hierzu wird die Suchanfragesprache um Funktionen erweitert, die es erlauben unabhängig vom Typ der Relation Verknüpfungen im Rahmen der Suche zwischen Konzepten zu verfolgen. Hierzu werden zum einen alle möglichen Relationstypen in die Suchanfragen integriert und zum anderen werden Gruppen von Relationstypen bei der Suche berücksichtigt.

Die Relationstypen orientieren sich dabei an den in der objektorientierten Welt bekannten Relationstypen. Des Weiteren wird die Möglichkeit angeboten, den Suchraum zu beschränken. Dabei kann der Mechanismus des Umkreises, wie er bei der Navigation beschrieben ist, genutzt werden.

Die Datenmodellunabhängige Anfrage stellt die erste Stufe für die unscharfe Suche dar und wird dort weiter verfeinert.

Unscharfe Suche

Die unscharfe Suche abstrahiert von dem Metamodell des ASN (siehe Abschnitt 2.2.6). Innerhalb der Suchanfrage werden die Strukturelemente des ASN wie Konzept, Attribut, Relation benötigt. Es werden zwei Mechanismen unterschieden.

Der erste Mechanismus der unscharfen Suche vergleicht alle Konzepte und Attribute mit einem beliebigen Suchbegriff. Dadurch wird das Wissen, ob es sich bei dem Suchbegriff um ein Konzept oder ein Attribut handelt nicht mehr benötigt. Dieses Verfahren wird um reguläre Ausdrücke erweitert, um Teilbegriffe und falsch geschriebene Informationen im ASN auffinden zu können. Um den Suchraum hier zu begrenzen, ist ebenfalls ein Umkreis um ein vorgegebenes Konzept zu definieren.

Der zweite Mechanismus dient zum Auffinden von Konzepten und deren Attribute auch dann, wenn die Suchkriterien nicht vollständig erfüllt sind. Dies bedeutet, dass alle Konzepte und deren Attribute als Ergebnisse der RPD-Anwendung zu präsentieren sind, auch wenn beispielsweise nur die Werte von zwei Attributen anstatt von drei Attributen, die in der Abfrage spezifiziert wurden, die Bedingung der Anfrage erfüllen. Dabei ist eine Gewichtung zu realisieren, die es erlaubt, die Ergebnisse nach ihrer bestmöglichen Übereinstimmung zu sortieren. Die Gewichtung ist dabei ein Maß für die semantische Nähe zweier Konzepte.

Quantitative und qualitative Anfrageelemente

Sowohl die scharfe als auch die unscharfe Suche wird um quantitative und qualitative Anfrageelemente erweitert. Die hinterlegten Suchmechanismen müssen dabei im Falle der quantitativen Anfragen, die Elemente *Wie viele?* und *Wie oft?* beinhalten. Diese Suchelemente haben dann die Attribute aufzusummieren. Die qualitativen Anfragen müssen die Fragestellungen *Welche?* oder *Womit?*, realisieren (siehe Abschnitt 6.1).

Für die Auswertung der quantitativen und qualitativen Anfragen sollen Attributwerte bzw. spezielle Relationen herangezogen werden. Diese Attribute und Relationen sind bei der Anfrage mit anzugeben.

Zusammenfassung

In Tabelle 3 sind die Anforderungen und die methodischen Ansätze zusammengefasst:

Nummer	Anforderung	Methode
1	Informationsbeschaffung bei gegebener Datenstruktur	Scharfe Suche
2	Datenmodell unbekannt	Navigation
3	Keine komplexe und Mengenanfragen möglich	Aggregation mehrerer Anfragen
4	Datenmodell nur teilweise bekannt bzw. dynamisch verändert	Vom Datenmodell unabhängige Anfragen / Erkennen durch Navigieren
5	Semantisch äquivalente Datenmodelle	Unscharfe Suche
6	Quantitative und qualitative Anfragen	Quantitative und qualitative Anfrageelemente

Tabelle 3: Anforderungen und ihre Lösungsmethoden

4.2 Methode zur Informationsüberwachung

Die Methode zur Informationsüberwachung verfolgt das Ziel, im ASN aufgetretene Veränderungen der RPD-Anwendung aktiv zu melden, d. h. die RPD-Anwendung muss nicht zyklisch das ASN auf Veränderungen prüfen, sondern wird von der Middleware benachrichtigt.

In der derzeitigen RPD-Umgebung ist die Überwachung von ASN-Elementen auf zwei Wegen möglich. Die erste Methode nutzt dabei den Mechanismus Event-Condition-Action-Regel (ECA-Regel) des ASN [Rol 02a], [Rol 02b]. Dieser Mechanismus erlaubt es, einen Attributwert im ASN zu verändern, wenn ein bestimmtes Ereignis, beispielsweise die Veränderung eines Attributwerts, eingetreten ist und eine Randbedingung, in Form einer Bedingungsanweisung, erfüllt ist. Diese Technik ermöglicht eine einfache Überwachung einzelner Werte im ASN, besitzt aber zwei Schwächen. Zum einen ist eine aktive Benachrichtigung einer RPD-Anwendung über die Ausführung der ECA-Regel nicht möglich. Und zum anderen werden nur Veränderungen von Attributwerten berücksichtigt und keine Strukturänderungen, wie beispielsweise das Löschen von Attributen oder die Überwachung der Versionsnummer eines Informationselements. Diese zwei Nachteile sind durch die Methode zur Informationsüberwachung auszugleichen.

Der zweite Mechanismus zur Informationsüberwachung ohne Einsatz der RPD-Middleware wird von der RPD-Anwendung aus initiiert. Dabei überprüft die RPD-

Anwendung selbständig in regelmäßigen Abständen das ASN, ob eine Veränderung eines bestimmten Wertes oder der Struktur stattgefunden hat. Dabei kann auch die ECA-Regel mit genutzt werden um die Überprüfung zu vereinfachen. Diese Technik löst zwar die Probleme der ersten Technik, verlagert aber die gesamte Programmlogik in die RPD-Anwendung. Dies bedeutet jede RPD-Anwendung, die das ASN überwachen möchte, muss die dazugehörende Technik immer wieder neu entwickeln.

Ein Problem des zweiten Ansatzes liegt in der zyklischen Überprüfung. Zum einen wird die Performanz der RPD-Anwendung durch die zyklische Überprüfung des ASN vermindert. Zum anderen kann es je nach Wahl des Überprüfungsintervalls passieren, dass Veränderungen, die nur kurzfristig auftraten, nicht registriert werden.

Die Methode der Informationsüberwachung nimmt sich diesen Problemstellungen an und gibt einen Lösungsansatz vor, der dann durch die RPD-Middleware umgesetzt wird. Die Methode verfolgt den Lösungsansatz der zweiten Technik weiter.

Die Funktion des Überwachens durch die RPD-Middleware muss so umgesetzt werden, dass das zyklische Warten durch die RPD-Middleware entfällt. Hierzu ist es notwendig, das ASN so anzupassen, dass es Veränderungen der betroffenen Elemente an die RPD-Middleware aktiv meldet. Unter *betroffenen Elementen* sind die Informationselemente zu verstehen, die für die Auswertung der Bedingung notwendig sind.

Die Auswertung der Bedingung hat sofort bei Veränderung eines Informationselements zu erfolgen. Ändert sich dabei der Wahrheitsgehalt, so ist dies der RPD-Anwendung mitzuteilen. Durch dieses Verfahren wird der Aufwand der RPD-Anwendung auf ein Minimum bei gleichzeitiger Maximierung der Überwachungsmöglichkeiten begrenzt.

Bedingungsanweisung

Bei der Informationsüberwachung werden Bedingungen formuliert. Es ist zu prüfen, ob eine Veränderung beliebiger Form im ASN stattgefunden hat, d. h. das Ergebnis der Informationsüberwachung sind die Wahrheitswerte *wahr* oder *falsch*.

Für die Berechnung des Wahrheitswerts erhält die Informationsüberwachung die Bedingung in Form einer zu spezifizierenden Sprache von der RPD-Anwendung mitgeteilt. Die Sprachdefinition muss dabei alle möglichen Vergleichsoperationen erlauben.

Überwachung von konkreten Inhalten

Die Überwachung von konkreten Inhalten im ASN stellt die erste Stufe der Überwachung dar. Bei dieser Variante ist ein einzelnes Attribut mit einem anderen Attribut des ASN oder einem in der Bedingungsanweisung fest vorgegebenen Fixwert zu vergleichen. Als Vergleichsoperatoren kommen die Operatoren *gleich*, *kleiner*, *größer*, etc. in Betracht. Beim Vergleich ist auf die Typisierung der Attribute des ASN zu achten, d. h. Zahlenwerte werden auf ihre Größenverhältnisse und Textstrings auf ihre alphabetische Reihenfolge verglichen.

Überwachung durch Nutzung von Zusatzfunktionen

Für den Vergleich sind einzelne mathematische Funktionen wie die vier Grundrechenarten einzubeziehen. Als Parameter, sowie als Vergleichswert kommen sowohl Fixwerte als auch Attributwerte aus dem ASN in Frage.

Neben den Attributen, Fixwerten und den Ergebnissen mathematischer Operationen, sind auch statistische Informationen zu berücksichtigen. Diese sind zum einen die Versionsnummer eines ASN-Informationselements (Netz, Konzept, Attribut, Relation) und zum anderen die Anzahl der Schreib- und Lesezugriffe auf ein ASN-Informationselement.

Überwachung von strukturellen Veränderungen

Unter strukturellen Veränderungen werden alle Veränderungen im ASN verstanden, die sich nicht auf die Inhalte (Attributwerte) beziehen. Diese sind:

- Das Löschen von ASN-Informationselementen.
- Das Erkennen, ob sich ein Attributwert geändert hat.
- Das Erkennen, ob sich die Version eines ASN-Informationselements geändert hat.
- Die Überprüfung, ob ein bestimmtes Konzept Bestandteil einer Relation ist.

Es ist zu beachten, dass das Ergebnis einer strukturellen Veränderung ein aussagenlogischer Wahrheitswert und nicht ein Zahlenwert oder Zeichenkette ist.

Komplexe Vergleiche

Mit Hilfe komplexer Vergleiche wird es möglich, die Bedingungen und aussagenlogischen Ergebnisse der vorangegangenen Abschnitte zu einer Gesamtbedingung zusammenfassen zu können. Hierzu werden die Teilbedingungen durch die aussagenlogischen Operatoren *AND*, *OR*, *XOR*, etc. verknüpft bzw. durch den Operator NOT einzelne Wahrheitswerte negiert.

Zusammenfassung

Die Methode der Informationsüberwachung kann damit zur Aufgabenübergabe bzw. Ergebnisübergabe zweier oder mehrerer RPD-Anwendungen entlang des RPD-Prozesses genutzt werden. Dabei wird die Aufgabe bzw. das Ergebnis in das ASN eingestellt. Die jeweils andere RPD-Anwendung überwacht die entsprechenden Informationselemente. Tritt eine Veränderung ein, kann die überwachende RPD-Anwendung sofort die Aufgabe bzw. das Ergebnis weiterbearbeiten. Durch dieses Verfahren müssen die beteiligten RPD-Anwendungen lediglich über die Struktur und den Ort der Information im ASN Kenntnis besitzen. Eine direkte Schnittstelle zwischen den RPD-Anwendungen zum Informationsaustausch ist nicht nötig.

4.3 Methode zur Koordination von Experten und deren RPD-Anwendungen

Die Koordination von Experten und deren RPD-Anwendungen bedeutet die informationstechnische Unterstützung der Durchführung des RPD-Prozesses. Im Vordergrund der Methode zur Koordination steht die Informationsschnittstelle zwischen RPD-Domänen, da hier unterschiedliche RPD-Anwendungen im Rahmen des RPD-Prozesses koordiniert werden müssen.

An den Informationsschnittstellen zwischen den RPD-Anwendungen treten nicht nur einfache Aufgaben- und Ergebnisübergabeprozesse auf, sondern auch Konfliktsituationen. Beispielsweise können Ergebnisse, bedingt durch Verzögerungen im RPD-Prozess, unvollständig sein, sodass eine nachfolgende gelagerte RPD-Anwendung ihre Aufgabe nicht beginnen kann. Um dieses Problem zu lösen, müssen Koordinationsprotokolle eingesetzt werden, die einen verhandelten Charakter besitzen. Es kann beispielsweise ein Koordinationsprotokoll erstellt werden, mit dem Ziel einen Termin auszuhandeln zu dem die Ergebnisse vollständig sein müssen.

Eine weitere Klasse von Koordinationsprotokollen ist für die Synchronisation von Abläufen zuständig. Benötigt z. B. eine RPD-Anwendung zwei Ergebnisse von unterschiedlichen RPD-Anwendungen, so wird ein Koordinationsprotokoll benötigt, das der RPD-Anwendung mitteilt, wenn beide Ergebnisse vorliegen.

Das Koordinationsprotokoll

Nach der Identifikation der zu koordinierenden Informationsschnittstellen zwischen zwei oder mehreren RPD-Anwendungen wird das Koordinationsprotokoll definiert. Als Repräsentationsform für das Koordinationsprotokoll ist das Zustandübergangsdiagramm zu verwenden. Die Zustände stellen dabei Koordinationspunkte (Teilergebnisse der Koordination) während der Koordination dar, zwischen denen durch Zustandsübergänge, die von den Eingaben durch die RPD-Anwendung abhängig sind, gewechselt wird. Durch entsprechende Ausgaben wird den RPD-

Anwendungen der Zustandswechsel angezeigt. Die Zustandsübergänge definieren die Wege, auf denen von einem Teilergebnis zum nächsten gelangt werden kann. Sie sind die Vorschriften, wie im Rahmen der Koordination zu agieren ist.

Da der RPD-Prozess und damit die Informationsschnittstellen zwischen den RPD-Anwendungen einer dynamischen Veränderung unterliegen, müssen die verwendeten Koordinationsprotokolle der jeweils neuen Aufgabe angepasst werden können. Dies bedeutet, dass die Koordinationsprotokolle nicht vorgegeben sein dürfen, sondern durch die RPD-Anwendung frei definierbar sein müssen. Die Spezifikation des Zustandübergangsdiagramms für das dazugehörige Koordinationsprotokoll erfolgt über eine Sprachschnittstelle zwischen der RPD-Anwendung und der RPD-Middleware.

Abarbeitung des Koordinationsprotokolls

Um die Koordination durchführen zu können, muss die Möglichkeit für die Koordinationspartner, d. h. den RPD-Anwendungen, geschaffen werden anhand des Koordinationsprotokolls, definiert durch das Zustandübergangsdiagramm, miteinander zu kommunizieren. Diese Kommunikation hat mit Hilfe von Nachrichten stattzufinden, die zwischen den Koordinationspartner über die Koordinationskomponente der RPD-Middleware ausgetauscht werden. Anhand der Nachrichten, die an die Koordinationskomponenten geschickt werden, wird entschieden, welche Zustandsübergänge durchzuführen sind. Die Koordinationskomponente hat dann, entsprechend dem durch die RPD-Anwendungen definierten Koordinationsprotokolls, durch das Verschicken von Ausgabenachrichten den Zustandsübergang den Koordinationspartnern mitzuteilen.

Wird die Koordination als Synchronisation eingesetzt, so verläuft die Koordination in der Regel dadurch, dass zuerst, im Falle von zwei RPD-Anwendungen, die eine RPD-Anwendung einen Lösungsvorschlag annimmt und dann die andere. Hierbei muss vom Koordinationsprotokoll berücksichtigt werden, dass es keine Rolle spielen darf, welcher Koordinationspartner den ersten Schritt macht. Die Umsetzung im Zustandübergangsdiagramm würde dieses Vorgehen einem Ausgangszustand entsprechen, von dem aus in zwei getrennte Übergangszustände gewechselt wird, je nachdem welcher Koordinationspartner als erster den Vorschlag akzeptiert. Mit der Akzeptanz durch den jeweils anderen Koordinationspartner wird dann der Endzustand erreicht (siehe Bild 11). Die Übergangszustände dienen dabei der Festlegung der Reihenfolge, wie synchronisiert wird und garantieren, dass beide Koordinationspartner dem Vorschlag zustimmen. Diese Vorgehensweise ist auf die Koordination von mehr als zwei Koordinationspartner entsprechend übertragbar.

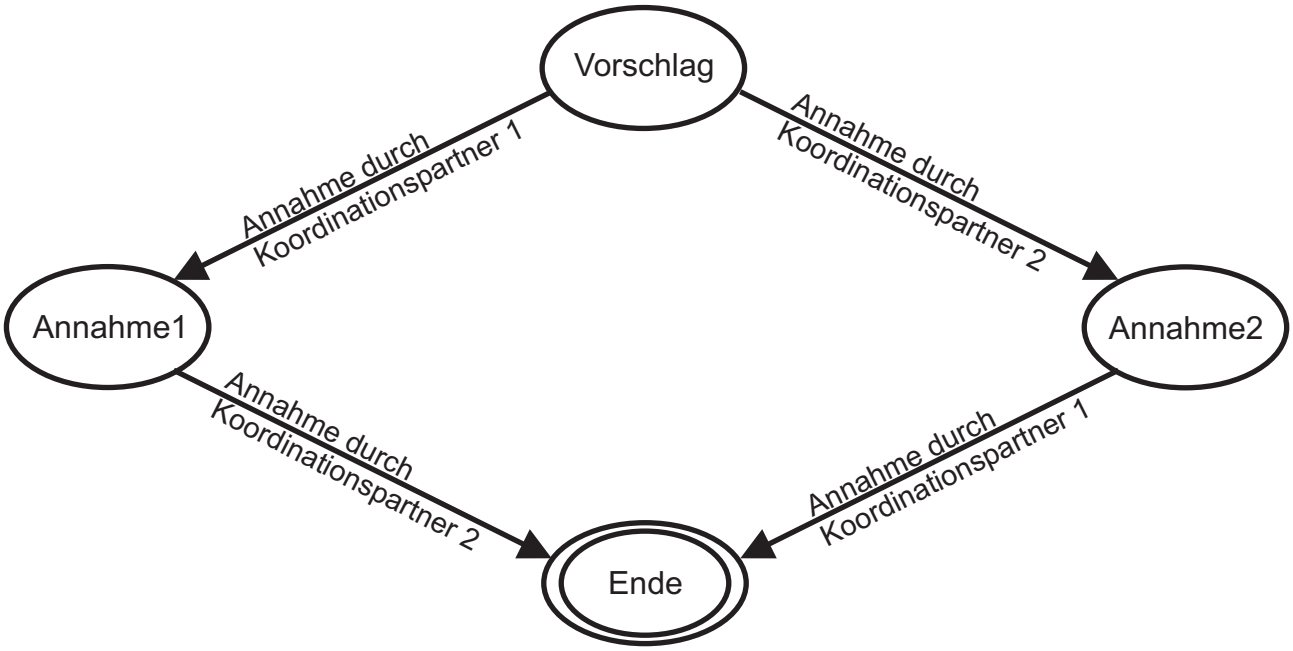


Bild 11: Zustandübergangsdiagramm einer einfachen Synchronisation

Bei der Verwendung dieser Art von Koordinationsprotokollen wird deutlich, dass sowohl die Eingabe- als auch die Ausgabenachrichten durch den entsprechenden sendenden bzw. empfangenden Koordinationspartner ausgezeichnet werden müssen. Denn nur dadurch kann entschieden werden, welche Verzweigung verfolgt werden soll. Im klassischen Zustandsübergangsdiagramm ist eine Auszeichnung der Ein- und Ausgaben nicht vorgesehen. Dieser Umstand muss durch die Realisierung der Koordinationskomponente in der RPD-Middleware berücksichtigt und die Funktionalität des Zustandübergangsdiagramms entsprechend erweitert werden.

Zusammenfassung

Zusammenfassend treten durch den Einsatz der informationstechnischen Unterstützung der Koordination durch die RPD-Middleware Randbedingungen auf, die nicht durch die RPD-Middleware direkt aufgelöst werden können.

- **Wer definiert das Koordinationsprotokoll?** Diese Fragestellung beschäftigt sich mit dem Problem, wie ein Koordinationsprotokoll zustande kommt. Das Koordinationsprotokoll muss außerhalb der RPD-Middleware definiert werden. Dies geschieht typischerweise bei der Anwendungsentwicklung bzw. der Konfiguration durch die Entwickler. Es ist zu bedenken, dass die Entwicklung des Koordinationsprotokolls eine Gemeinschaftsarbeit aller Beteiligten ist, d. h. das Koordinationsprotokoll wird nicht von einem Entwickler einer RPD-Anwendung vorgegeben, sondern muss in Abstimmung aller an der Koordination Beteiligten definiert werden.

- **Wer ist an der Koordination beteiligt?** An der Koordination sind alle RPD-Anwendungen beteiligt, die über dieses Koordinationsprotokoll abgestimmt werden.
- **Wie wird der Koordinationspartner adressiert?** Um Eingabenachrichten bewerten und Ausgabenachrichten auszeichnen zu können, müssen die RPD-Anwendungen mit einem für das definierte Koordinationsprotokoll eindeutigen Namen versehen werden. Dieser eindeutige Name wird bei der Definition des Koordinationsprotokolls außerhalb der RPD-Middleware festgelegt. Der Name wird dann bei der Definition der Zustandsübergänge durchgängig verwendet.
- **Wie finden die RPD-Anwendungen das Koordinationsprotokoll?** Das vorher definierte Koordinationsprotokoll wird durch eine RPD-Anwendung in die RPD-Middleware eingestellt. Die beteiligten RPD-Anwendungen kommunizieren mit der Koordinationskomponente der RPD-Middleware, indem sie das Koordinationsprotokoll über den eindeutigen Namen ansprechen.
- **Was geschieht mit Nachrichten an Koordinationspartner, die zum Zeitpunkt des möglichen Empfangs nicht aktiv sind?** Diese Nachrichten werden verworfen, da die Nachrichten durch erneute Zustandsübergänge ihre Aktualität verlieren. Die RPD-Anwendung muss dafür aber in der Lage sein, jederzeit den aktuellen Zustand abzufragen, um sich beim Eintritt in die Koordination den aktuellen Zustand zu beschaffen.

4.4 Methode zur Kommunikation

Die Methode zur Kommunikation nimmt sich der Fragestellung an, welche Funktionalitäten benötigt werden, damit die RPD-Anwendungen im Rahmen der Nutzung der RPD-Middleware mit dieser kommunizieren können.

Kommunikationsbeziehungen

Die in Tabelle 4 dargestellte Matrix zeigt sowohl die gerichteten Kommunikationsbeziehungen für die Informationsbeschaffung und -aufbereitung, als auch für die Informationsüberwachung und die Koordination zwischen den RPD-Anwendungen und der RPD-Middleware die einzelnen Vorgänge.

Die folgenden Begriffe innerhalb der Matrix bedeuten:

- **Anfrage**: Einmalige Anfrage an den entsprechenden Kommunikationspartner. Dies ist hier immer die RPD-Middleware oder das ASN. Die Anfrage dient zur Beauftragung der entsprechenden Komponente der RPD-Middleware.
- **Ergebnis**: Einmalige Ergebnisübermittlung als Antwort auf einen Auftrag. Die Kommunikation findet zwischen der RPD-Middleware, dem ASN und den RPD-Anwendungen statt.

Kommuniziert von / nach	Einzelne RPD-Anwendung	Weitere RPD-Anwendungen	Informationsbeschaffung	Informationsaufbereitung	Informationsüberwachung	Koordination	ASN
Einzelne RPD-Anwendung		Direkte Kommunikation nicht berücksichtigt	Anfrage (A)	Anfrage (A)	Anfrage (A)	Anfrage und laufende Kommunikation (B)	Anfrage (A)
Weitere RPD-Anwendungen	Direkte Kommunikation nicht berücksichtigt		Siehe einzelne RPD-Anwendung	Siehe einzelne RPD-Anwendung	Siehe einzelne RPD-Anwendung	Laufende Kommunikation (B)	Siehe einzelne RPD-Anwendung
Informationsbeschaffung	Ergebnis (A)	Siehe einzelne RPD-Anwendung		Ergebnis (A)			Anfrage (A)
Informationsaufbereitung	Ergebnis (A)	Siehe einzelne RPD-Anwendung	Anfrage (A)				
Informationsüberwachung	Mehrere Ergebnisse (A)	Siehe einzelne RPD-Anwendung					Anfrage (A)
Koordination	Laufende Kommunikation (B)	Laufende Kommunikation (B)					
ASN	Ergebnis (A)		Ergebnis (A)		Mehrere Ergebnisse (A)		

Tabelle 4: Kommunikationsbeziehungen der RPD-Anwendungen, des ASN und der RPD-Middleware

- **Mehrere Ergebnisse**: Auslieferung von Ergebnissen in loser Zeitfolge. Diese Kommunikation findet auf dem Pfad ASN, RPD-Middlewarekomponente Informationsüberwachung, RPD-Anwendung statt.
- **Laufende Kommunikation**: Kommunikationsbeziehung in beide Richtungen in loser Zeitfolge. Diese Kommunikation findet zwischen den RPD-Anwendungen und der RPD-Middlewarekomponente Koordination statt.

Die in Klammern dargestellten Kategorien A und B stehen für die entsprechenden Kardinalitäten der Kommunikationsbeziehungen. Dabei repräsentiert die Kategorie A: 1-zu-1 – Beziehungen und B: 1-zu-n – Beziehungen. Es ist zu beachten, dass durch die Abwicklung der Koordination durch die RPD-Middleware, die m-zu-n – Beziehung zwischen den zu koordinierenden RPD-Anwendungen auf 1-zu-n – Beziehungen vereinfacht wurde.

Aus der Matrix der Kommunikationsbeziehungen lässt sich ableiten, dass die Kommunikations-Komponente der RPD-Middleware 1-zu-1 – und 1-zu-n – Beziehungen unterstützen muss.

Kommunikationspartnerwahl und Adressierung

Eine in Tabelle 4 nicht dargestellte Kommunikationsbeziehung ist das Auffinden der RPD-Middlewarekomponente, die zur Erbringung einer Dienstleistung von der RPD-Anwendung beauftragt werden soll. Dabei tritt die Frage auf, wie die Kommunikationspartner zu adressieren sind. Die Nutzung einer physischen Adresse, wie z. B. die IP-Adresse ist nicht sinnvoll, da einzelne Komponenten der RPD-Middleware nicht ständig bzw. mehrfach benötigt werden. Es ist daher ein logisches Adressierungsschema zu wählen, das den Typ der benötigten RPD-Middlewarekomponenten beinhaltet.

Diese logische Adresse muss dann in eine Physische umgesetzt werden. Dies kann beispielsweise durch den Einsatz einer Multicast-Umgebung erreicht werden. Nachrichten, die innerhalb dieser Umgebung verschickt werden, beinhalten die logischen Sender- und Empfängeradressen. Verschickt ein Kommunikationspartner eine Nachricht wird diese von allen Kommunikationspartnern empfangen. Ist die Empfängeradresse identisch mit der Adresse eines Kommunikationspartners, so ist die Nachricht für diesen bestimmt und wird von diesem weiterverarbeitet. Durch dieses Verfahren muss jedem Kommunikationspartner nur noch die physische Adresse der Multicast-Umgebung (Multicast-Adresse) bekannt sein.

Bei der Beauftragung einer RPD-Middlewarekomponente durch eine RPD-Anwendung ist dieser die logische Adresse der Middlewarekomponenten nicht bekannt. Der Grund hierfür ist, dass die entsprechende Komponente der RPD-Middleware entweder zu diesem Zeitpunkt benutzt wird oder noch nicht instanziiert ist. Es wird daher eine Komponente innerhalb der RPD-Middleware benötigt, die Anforderungsnachrichten der dienstanfordernden RPD-Anwendungen zur weiteren

Bearbeitung entgegen nimmt. Diese instanziiert dann die entsprechende Komponente und teilt die logische Komponentenadresse der RPD-Anwendung mit. Dieser Ansatz hat den Nachteil, dass eine dedizierte Komponente der RPD-Middleware für diesen Vorgang benötigt wird und im Fehlerfalle die gesamte Middleware blockiert. Daher ist diese Funktionalität mit Hilfe eines verteilten Ansatzes auf alle Middlewarekomponenten zu verteilen.

Der in diesem Abschnitt entwickelte Adressierungsmechanismus entspricht der in Abschnitt 2.2.2 beschriebenen äußeren Sprache.

Kommunikationsinhalt und -protokoll

Ist die Kommunikationsbeziehung bestimmt und der Kommunikationspartner ausgewählt, findet die Kommunikation zwischen den Kommunikationspartnern statt. Innerhalb dieser Kommunikation werden die für die Aufgabenabwicklung benötigten Informationen ausgetauscht. Hierfür sind die Nachrichten derart zu strukturieren, dass sie von beiden Seiten verstanden werden können. XML eignet sich durch seine Flexibilität in diesem Bereich (siehe Abschnitt 2.2.5).

Die XML-Struktur ist dabei von Anwendungsfall zu Anwendungsfall verschieden. Die Informationsbeschaffung verwendet andere Informationen als die Informationsüberwachung. So ist für jeden Aufgabentyp der RPD-Middleware eine Sprache auf seine Aufgabe abgestimmt zu definieren. Diese Sprache ist in eine XML-Struktur zu überführen und stellt die innere Sprache der Kommunikation dar (siehe Abschnitt 2.2.2).

Welche Nachrichten wann, warum und zwischen wem ausgetauscht werden, wird durch ein Kommunikationsprotokoll bestimmt. Das Kommunikationsprotokoll ist ebenfalls von Anwendungsfall zu Anwendungsfall verschieden und muss bei der Entwicklung der einzelnen Middlewarekomponenten festgelegt werden.

Zusammenfassung

Die Methode zur Kommunikation legt durch die Aufteilung in eine physische Multicast-Adresse und logische Adresse für die Kommunikationspartner den Grundstein zur Kommunikation. Durch den Einsatz von XML als Nachrichtenformat sind die Nachrichten für jeden Kommunikationspartner unabhängig von seiner Implementierung verständlich. Die Aufteilung der Adressierung in eine äußere Sprache und des eigentlichen Inhalts in eine innere Sprache führt zur Trennung zwischen allgemeingültigen und anwendungsfallabhängigen Informationen. Die Strukturierung der Informationen in der inneren Sprache und die Protokolle, die für den Austausch der Nachrichten verwendet werden, sind von jedem Typ von Middlewarekomponenten getrennt zu entwickeln (siehe Kapitel 6).

5 Entwicklung der RPD-Middleware-Architektur

5.1 Schnittstelle zur RPD-Middleware

In Bild 12 ist das Zusammenwirken und die Schnittstellen zwischen den RPD-Anwendungen, der RPD-Middleware und dem ASN dargestellt. Dabei wird deutlich, dass die Schnittstelle zwischen RPD-Anwendung und RPD-Middleware offen und programmiersprachenunabhängig sein muss, während die Schnittstelle zwischen RPD-Middleware und ASN eng und auf Effizienz ausgelegt sein muss. Die offene Schnittstelle zu den RPD-Anwendungen ist deshalb zwingend notwendig, da die RPD-Anwendungen unterschiedlich sowohl in ihrer Struktur als auch in ihren Implementierungssprachen sind.

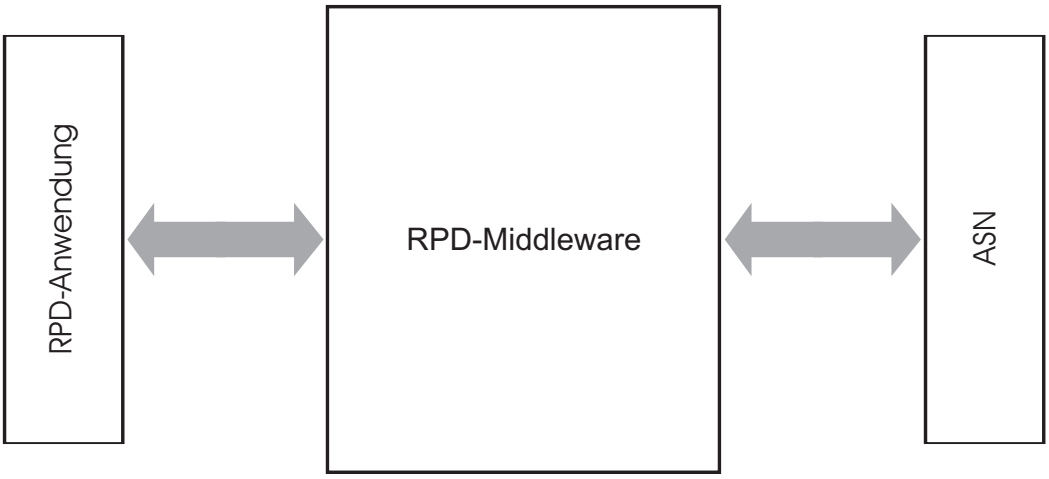


Bild 12: Schnittstellenbeschreibung

Schnittstelle zwischen RPD-Anwendung und RPD-Middleware

Um dem Ziel einer möglichst einfachen und programmiersprachenunabhängigen Schnittstelle gerecht zu werden, wird auf eine nachrichtenbasierte Schnittstelle zurückgegriffen (siehe 4.4). Nachrichten haben den Vorteil, dass sie in ihrer Implementierung unabhängig von einer Programmiersprache sind. Das Kommunikationsprotokoll legt fest, wann welche Nachrichten zwischen welchen Kommunikationspartner ausgetauscht werden.

Als Nachrichtenformat wird XML [Bra 04] benutzt (siehe Abschnitt 2.2.5). Die RPD-Anwendungen müssen unabhängig von ihrer internen Struktur und ihrer Implementierungssprache zur Kommunikation mit der RPD-Middleware lediglich XML-Nachrichten erzeugen und empfangene Nachrichten auswerten. Für die RPD-Middleware hat dies den Vorteil, dass ein einheitliches Empfangs- und Aufbereitungsmodul für die Agenten entwickelt werden kann (siehe Abschnitt 5.5). Die möglichen Nachrichten und ihr exaktes Format hängen stark von der Funktionalität der

einzelnen Agententypen ab. Die genaue Definition der Nachrichten finden sich im Anhang E.

Schnittstelle zwischen RPD-Middleware und ASN

Die Schnittstelle zwischen der RPD-Middleware und dem ASN ist durch die Implementierung des ASN vorgegeben. Das ASN bietet als Zugang eine Java-basierte Schnittstelle in Form einer Klasse an, die als Bibliothek betrachtet, alle Befehle zum Informationsaustausch mit dem ASN beinhaltet. Diese Schnittstelle wird sowohl von der RPD-Middleware, als auch von den RPD-Anwendungen bei der direkten Kommunikation mit dem ASN genutzt.

Die Informationsüberwachung stellt gegenüber der Informationsbeschaffung und -aufbereitung, sowie der Koordination, eine Besonderheit dar, da zur aktiven Meldung von Veränderungen im ASN Anpassungen des ASN unabdingbar sind (siehe Abschnitt 6.3). Die Erweiterung des ASN erzwingt auch eine Erweiterung der Schnittstelle zwischen RPD-Middleware und ASN. Diese Schnittstelle wurde auf dieselbe Art und Weise aufgebaut, wie die bereits existierende Schnittstelle.

5.2 Entwicklung einer einheitlichen Kommunikationssprache

Als Kommunikationssprache wird hier eine Sprache verstanden, in der die RPD-Anwendungen mit der RPD-Middleware und die Agenten untereinander kommunizieren. Die Kommunikation unterliegt zwei Anforderungen, zum einen der Forderung nach einer eindeutigen Adresse des Kommunikationspartners und zum anderen nach einem einheitlichen Nachrichtenformat der ausgetauschten Nachrichten (siehe Abschnitt 4.4 und Bild 13).

Bild 13: Nachrichtenaustausch in der Kommunikationssprache

Als eindeutige, logische Adresse wurde ein *Unique Agent Identifier* (UAI) definiert. Anhand des UAI wird innerhalb der Multicast-Umgebung festgestellt wer mit wem kommuniziert, d. h. Nachrichten innerhalb der Multicast-Umgebung werden nur von

den Teilnehmern empfangen und weiter bearbeitet, deren Empfänger-UAI mit ihrer eigenen UAI übereinstimmen. Der UAI besteht aus der Sequenz <AgentName> / <AgentType> / <ID> und muss innerhalb des Agentensystems (der RPD-Middleware) eindeutig sein. Der *AgentType* stellt den Agententyp dar, wie z. B. ein Monitor-Agent und wird für die Instanziierung des entsprechenden Agenten benötigt. Der *AgentName* ist eine frei definierbare Zeichenkette, die bei der Instanziierung für das Verständnis der Kommunikation mit einem logischen Inhalt belegt wird. Durch die Middleware kann eine Eindeutigkeit des *AgentName* nicht garantiert werden, da die RPD-Anwendungen diesen frei vergeben können und keine Absprache zwischen den RPD-Anwendungen stattfindet. Die *ID* dient zur Herstellung der Eindeutigkeit, weshalb in der Regel die Systemzeit benutzt wird.

Als Nachrichtenformat wurde aufgrund der Strukturierungsmöglichkeiten und der guten Lesbarkeit ein Format auf Basis des XML-Standards [Bra 04] entwickelt (siehe Abschnitt 4.4). Das Nachrichtenformat (siehe Bild 14) beinhaltet vier Parameter:

- **sender**: UAI des Senders zur Information, an wen die Nachricht zu richten ist.
- **destination**: UAI des Empfängers der Nachricht.
- **reqControl**: Informationsparameter, wie die Nachricht durch den adressierten Agenten zu behandeln ist. Diese Information wird z. B. für das Protokoll zur Bestimmung des Master-Agenten benutzt.
- **dataType**: Typ der Nachricht, die im Data-Block eingebettet ist. Anhand des Typs entscheidet der Empfänger, wie die Nachricht innerhalb des Kommunikationsprotokolls einzuordnen ist.

Zwischen den Auszeichnungen <*data*> und <*/data*> ist die eigentliche Nachricht verpackt. Hier befinden sich die Sprachprimitiven der inneren Sprache.

```
<request sender=" " destination=" " reqControl=" " dataType=" ">
        <data></data>
</request>
```

Bild 14: Syntax des Befehlsformats für die Kommunikation zwischen den Agenten

In Bild 15 ist die Bearbeitung einer ankommenden Nachricht durch den Agenten dargestellt. Der *requestProcessor* organisiert die Kommunikation zwischen den Agenten ohne deren gegenseitige Kenntnis voneinander. Er bestimmt anhand der *destination* und des *dataType,* ob diese Nachricht für den Agenten bestimmt ist. Durch die zusätzliche Betrachtung des *dataType*, werden Nachrichten, die fälschlicherweise an den Agenten gerichtet sind, aber nicht zum Protokoll des Agenten gehören, ausgefiltert. Z. B. passiert dies dann, wenn eine Nachricht aus dem Retrieval-Protokoll an einen Monitor-Agenten geschickt wird. Eine weitere Aufgabe des *requestProcessor* ist die Umwandlung der Nachricht in ein für den Agent besser

lesbares Event. Dieses Event wird über den *agentObserver* an den registrierten *agentListener* des Agenten, der Bestandteil des Verwaltungsmoduls (siehe Abschnitt 5.5) ist, zur Bearbeitung geschickt.

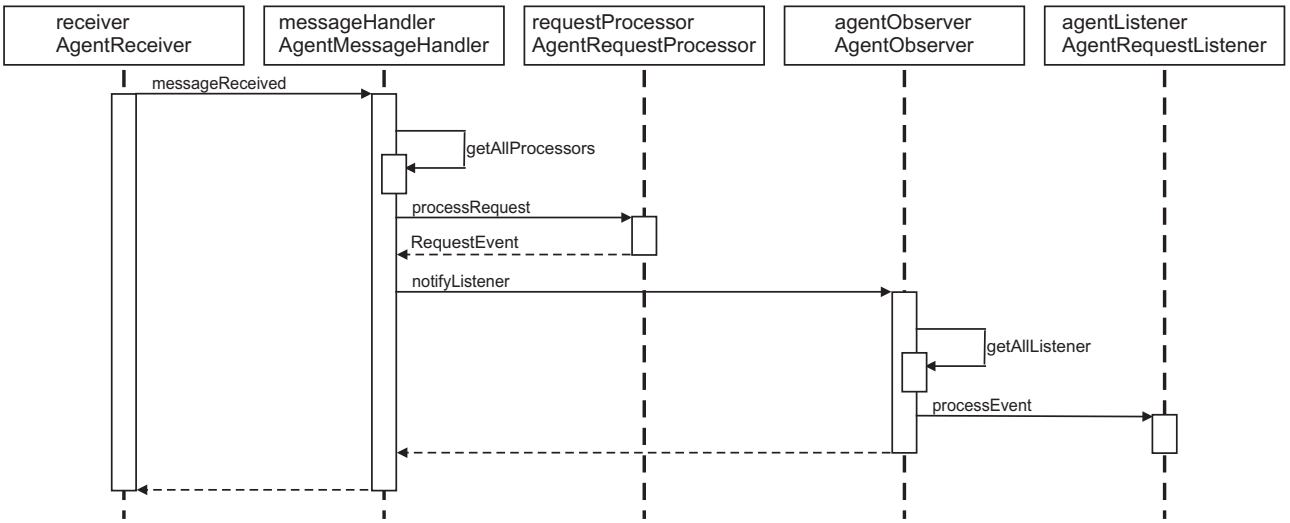


Bild 15: Sequenzdiagramm der internen Nachrichtenbearbeitung eines Agenten

5.3 Multicast-Umgebung

Für die Kommunikation zwischen RPD-Anwendung und den Agenten der RPD-Middleware sowie der Agenten untereinander ist eine Kommunikationsform zu wählen, die der Methode aus Abschnitt 4.4 gerecht wird. Auf eine sichere und fehlertolerante Kommunikation und einem einfach aufgebauten Zugangspunkt zum Agentensystem ohne aufwendige Konfigurationsaufgaben auf Seiten der RPD-Anwendung ist dabei zu achten.

Dabei ist zu berücksichtigen, dass die RPD-Anwendung ein Problem zu lösen hat und dafür die Unterstützung der RPD-Middleware benötigt. Das Problem kann vielschichtig sein, es kann sich um das Suchen von Wissen handeln oder aber auch um das Überwachen des ASN, um auf Veränderungen reagieren zu können, die von anderen RPD-Anwendungen eingebracht wurden. Hieraus folgt als weitere Anforderung an das Agentensystem das Anbieten einer Dienstleistung und das Auffinden der angebotenen Dienstleistung durch eine einheitliche Schnittstelle.

Eine mögliche Vorgehensweise entsprechende Agenten aufzufinden ist die einzelne Überprüfung der Agenten durch die anfragende RPD-Anwendung auf deren Fähigkeit, die entsprechende Aufgabe zu übernehmen. Das hat aber den Nachteil, dass allen RPD-Anwendungen die Adressdaten aller Agenten bekannt sein müssen und zum anderen ist das sequentielle Abfragen der Agenten, ob sie in der Lage sind, die Aufgabe zu übernehmen, langwierig. Dieses Verfahren hätte den Vorteil eines einfachen Agentensystems mit dem Nachteil eines komplexen Kommunikationsmoduls innerhalb der RPD-Anwendungen.

Ein Ansatz ist das Implementieren einer zentralen Serverinstanz, z. B. durch einen zentralen Verzeichnisdienst, die alle Anfragen entgegennimmt und dem entsprechenden Agenten weiterreicht. Ist die Serverinstanz im Fehlerfall nicht verfügbar, bedeutet dies den Zusammenbruch des Systems. Des Weiteren muss die Serverinstanz zyklisch alle Agenten auf Verfügbarkeit und Fehlerfreiheit sequentiell überprüfen. Durch diese Vorgehensweise wird die Komplexität der Kommunikation in das Agentensystem verlagert.

Durch die Realisierung einer Multicast-Umgebung wird ein Zugriffspunkt für alle Kommunikationspartner definiert. Dies bedeutet, alle Informationen sind allen Kommunikationspartnern zugänglich (siehe Bild 16), sodass jedem nur noch eine physische Adresse bekannt sein muss. Die Adressierung der einzelnen Komponenten erfolgt über das logische Adressierungsschema (dem UAI) aus Abschnitt 5.2.

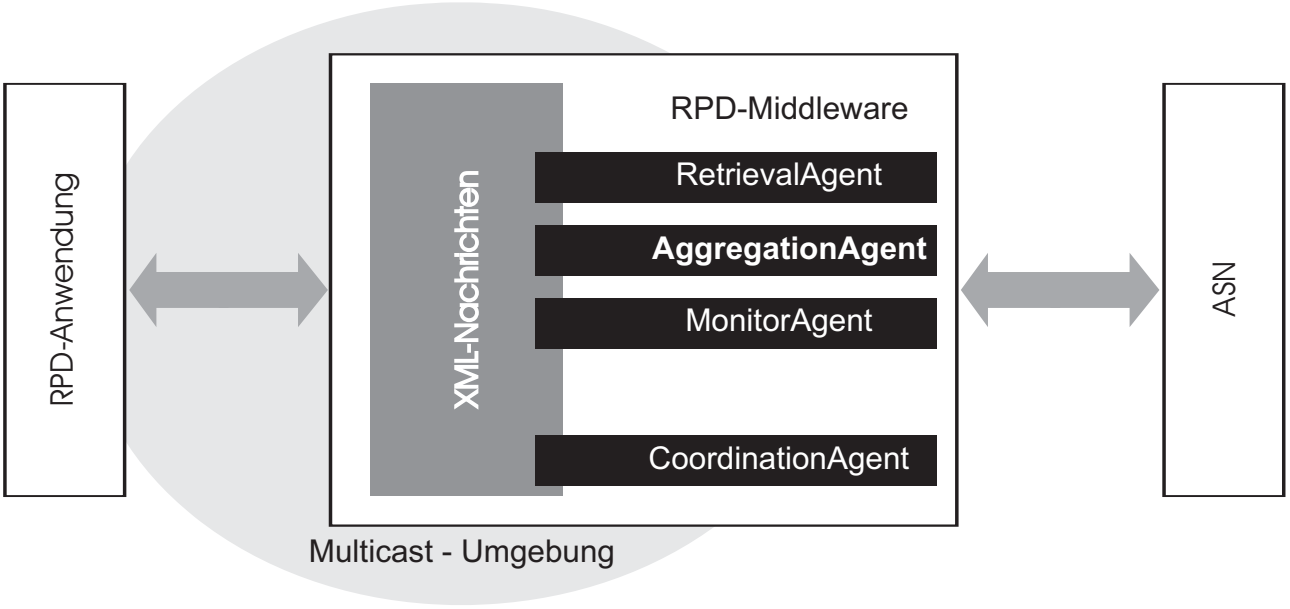


Bild 16: Multicast-Umgebung der RPD-Middleware

Das Problem des Auffindens eines verfügbaren Agenten wird auf das Agentensystem verlagert. Innerhalb des Agentensystems wird eine zentrale Instanz implementiert, der so genannte Master-Agent [Die 01], [War 02], in Bild 16 fett dargestellt. Dieser Agent übernimmt die Aufgabe, die anderen Agenten auf ihre Verfügbarkeit hin zu überwachen, zu überprüfen und bei Bedarf neue zu instanziieren. Zu diesem Zweck führt der Master-Agent eine Liste, die alle verfügbaren Agenten beinhaltet. Zusätzlich beendet der Master-Agent alle Agenten, die durch nicht korrekt abgeschlossene Aufträge, verursacht durch die RPD-Anwendungen, verwaist sind. Damit ist eine Fehlertoleranz innerhalb der Kommunikation zwischen RPD-Anwendung und den RPD-Agenten erreicht. Allerdings stellt der Master-Agent durch sein Alleinstellungsmerkmal bei der Annahme von Aufträgen einen *Single-Point-of-Failure* dar, d. h. fällt er aus, bricht das System zusammen.

Dem *Single-Point-of-Failure*-Problem, bedingt durch die Verfügbarkeit des Master-Agenten, wird dadurch begegnet, dass der Master-Agent nicht als einzelner, besonderer Agent realisiert wird, sondern als Bestandteil aller Agententypen. Dadurch ist es möglich, dass alle Agenten Master-Agent werden können. Um den Master-Agent zu bestimmen, wird das Netzwerkprotokoll Tokenring [IEEE 802.5] benutzt, wodurch der Master-Agent selbst auf Fehlfunktionen überwacht und damit gewährleistet wird, dass immer ein Master-Agent innerhalb des Agentensystems zur Verfügung steht (siehe Abschnitt 5.4).

Durch die Wahl einer Multicast-Umgebung, ist es für den Master-Agent einfach, die Verfügbarkeit der Agenten zu testen. Er muss lediglich regelmäßig eine Nachricht an die Multicast-Adresse schicken, auf die die Agenten antworten müssen. Die erhaltenen Antworten werden dann mit der geführten Liste abgeglichen.

Allerdings weist die Basistechnologie Multicast gegenüber der Broadcast- und Unicast-Technologie Schwächen auf. Es wird die Übertragung der Pakete zwischen Sender und potentiellen Empfängern genauso wenig garantiert, wie die chronologische Reihenfolge des Eintreffens der Pakete beim Empfänger. Diesem Problem wird durch die Erweiterung des Multicast-Protokolls begegnet, wodurch ein Reliable Multicast Protokoll (RMP) entsteht [Lia 00]. Dieses RMP wird in der RPD-Middleware eingesetzt (siehe Abschnitt 7.2).

5.4 Protokoll der Masterfunktionalität

Auf der Basis der im Abschnitt 5.3 beschriebenen Masterfunktionalität wird nun das Protokoll der Masterfunktionalität detailliert beschrieben. Dieses Protokoll ist gleichzeitig ein Beispiel für die Funktionsweise der Multicast-Umgebung und des Nachrichtenaustauschs.

In Bild 17 ist das Protokoll zur Bestimmung des Master-Agenten bzw. zur Kontrolle ob ein Agent mit Masterfunktionalität vorhanden ist dargestellt. Dieses Protokoll wird von allen Agenten realisiert und enthält die folgenden Nachrichten:

- **DAT** (Duplicate Address Test): Mit dieser Nachricht wird überprüft, ob der UAI des Agenten innerhalb der RPD-Middleware eindeutig ist.
- **AMP** (Active Master Present): Diese Nachricht wird vom Master-Agenten zyklisch verschickt und zeigt allen Agenten die Präsenz des Master-Agenten und dessen eindeutige Adresse (UAI) an.
- **SMP** (Standby Master Present): Diese Nachricht wird von allen Agenten, außer dem Master-Agenten, an den Master-Agenten als Antwort auf die AMP-Nachricht verschickt. Dadurch erhält der Master-Agent einen Überblick über die sich in der RPD-Middleware befindlichen Agenten, deren UAI er in einer Liste ablegt.

- **RP** (Ring Purge): Diese Nachricht wird nach der Wahl eines Master-Agenten, vom Master-Agenten als Zeichen dafür, dass die RPD-Middleware einen stabilen Zustand erreicht hat, verschickt.

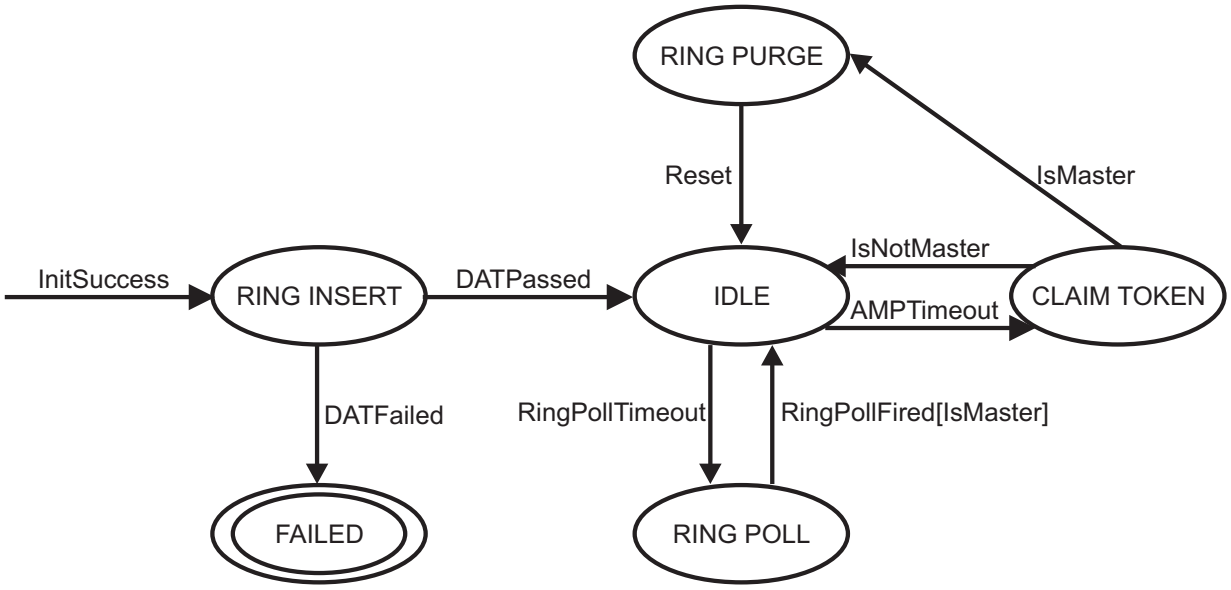


Bild 17: Zustandsdiagramm eines Agenten

Anhand dieser Nachrichten durchläuft der Agent im Laufe seines Lebens die folgenden Zustände:

- **Ring Insert**: Diesen Zustand erreicht der Agent bei seiner Instanziierung. In diesem Zustand wird der Duplicate Address Test durchgeführt. Besitzt ein Agent dieselbe UAI wie der anfragende Agent, dann wird dies dem anfragenden Agenten mitgeteilt. Ist die Adresse noch nicht vergeben so wird der Agent in das Agentensystem aufgenommen und gelangt in den Zustand *Idle*.
- **Idle**: In diesem Zustand nimmt der Agent seine eigentlichen Aufgaben wahr und kommuniziert mit der oder den RPD-Anwendungen, sowie im Falle des Retrieval- und Aggregations-Agenten auch mit anderen Agenten. Mit Eintritt in den Zustand *Idle* wird ein AMP-Timer gestartet. Läuft dieser ab, ist zu dieser Zeit kein Master-Agent vorhanden und es wird in den Zustand *Claim Token* gewechselt. Dies passiert immer dann, wenn der Master-Agent beendet wurde oder der neu instanziierte Agent der erste Agent in der RPD-Middleware ist. Wird innerhalb des Zeitintervalls eine AMP-Nachricht empfangen, so wird diese durch eine SMP-Nachricht quittiert und der Timer neu gestartet. Befindet sich der Master-Agent im Zustand *Idle*, so startet er zusätzlich einen Ring Poll-Timer, der schneller abläuft als der AMP-Timer und wechselt nach Ablauf des Ring Poll-Timers in den Zustand *Ring Poll*.
- **Claim Token**: In diesem Zustand wird der neue Master-Agent ausgehandelt. Der Prozess basiert auf den UAIs der Agenten. Jedem Agent sind bedingt durch die AMP- und SMP-Nachrichten die UAI der anderen Agenten in der

RPD-Middleware bekannt. Der Agent mit der höchsten UAI wird Master-Agent und wechselt in den Zustand *Ring-Purge*. Die anderen Agenten wechseln zurück in den Zustand *Idle*.

- **Ring Purge**: In diesem Zustand sendet der Master-Agent eine RP-Nachricht und zeigt damit allen Agenten an, dass ein neuer Master-Agent gewählt wurde. Anschließend wechselt er in den Zustand *Idle*.
- **Ring Poll**: In diesem Zustand verschickt der Master-Agent die AMP-Nachricht und wartet auf die Quittierung aller Standby-Agenten durch eine SMP-Nachricht und wechselt zurück in den Zustand *Idle*.

5.5 Agentenaufbau und Funktionsweise

In diesem Abschnitt wird die interne Struktur und Funktionsweise der Agenten betrachtet. Hierzu wurde der Aufbau von Agenten in einem Literaturstudium untersucht (siehe Abschnitt 2.2.1) und das Modell nach Fatima [Fat 98] weiterverfolgt. Dabei werden die Aufgaben der einzelnen Module derart aufgeteilt, dass zwischen den einzelnen Agententypen lediglich die Problemlöse-Komponente und das Design der Datenbasis angepasst werden müssen.

In Bild 18 sind die Module der Agenten mit ihren Wechselwirkungen im Rahmen der RPD-Middleware-Architektur dargestellt. Diese Module finden sich in jedem Agententyp der RPD-Middleware wieder.

- **Kommunikation**: Das Kommunikationsmodul stellt die Schnittstelle des Agenten nach außen dar. Alle Nachrichten werden dort verschickt, empfangen und aufbereitet. Des Weiteren reagiert das Kommunikationsmodul selbständig auf Verfügbarkeitsanfragen des Master-Agenten. Zusätzlich überprüft das Kommunikationsmodul zyklisch, ob ein Master-Agent im System vorhanden ist, d. h. in diesem Modul werden die Mechanismen aus den vorangegangenen Abschnitten realisiert.
- **Verwaltung**: Das Verwaltungsmodul steuert den gesamten Agenten; es stellt damit das Kernstück des Agenten dar. Die vom Kommunikationsmodul gelieferten Informationen werden vom Verwaltungsmodul in die Datenbasis geschrieben und die Problemlösungskomponente angestoßen. Nach erfolgter Abarbeitung der Aufgabe holt das Verwaltungsmodul die Ergebnisse aus der Datenbasis und stellt sie über das Kommunikationsmodul der anfragenden RPD-Anwendung oder des anfragenden Agenten zur Verfügung.
- **Problemlösung**: Das Problemlösungsmodul bearbeitet die eigentliche Aufgabe. Das Problemlösungsmodul ist daher in seinem inneren Aufbau von Agententyp zu Agententyp unterschiedlich. Die Problemlösung findet unter Zuhilfenahme der in der Datenbasis abgelegten Informationen statt.
- **Datenbasis**: Die Datenbasis enthält alle für die Bearbeitung der Aufgabe wichtigen Daten. Ebenfalls dient sie zur Zwischenspeicherung von Teilergebnissen der Problemlösungskomponente und zur Ablage des Endergebnisses.

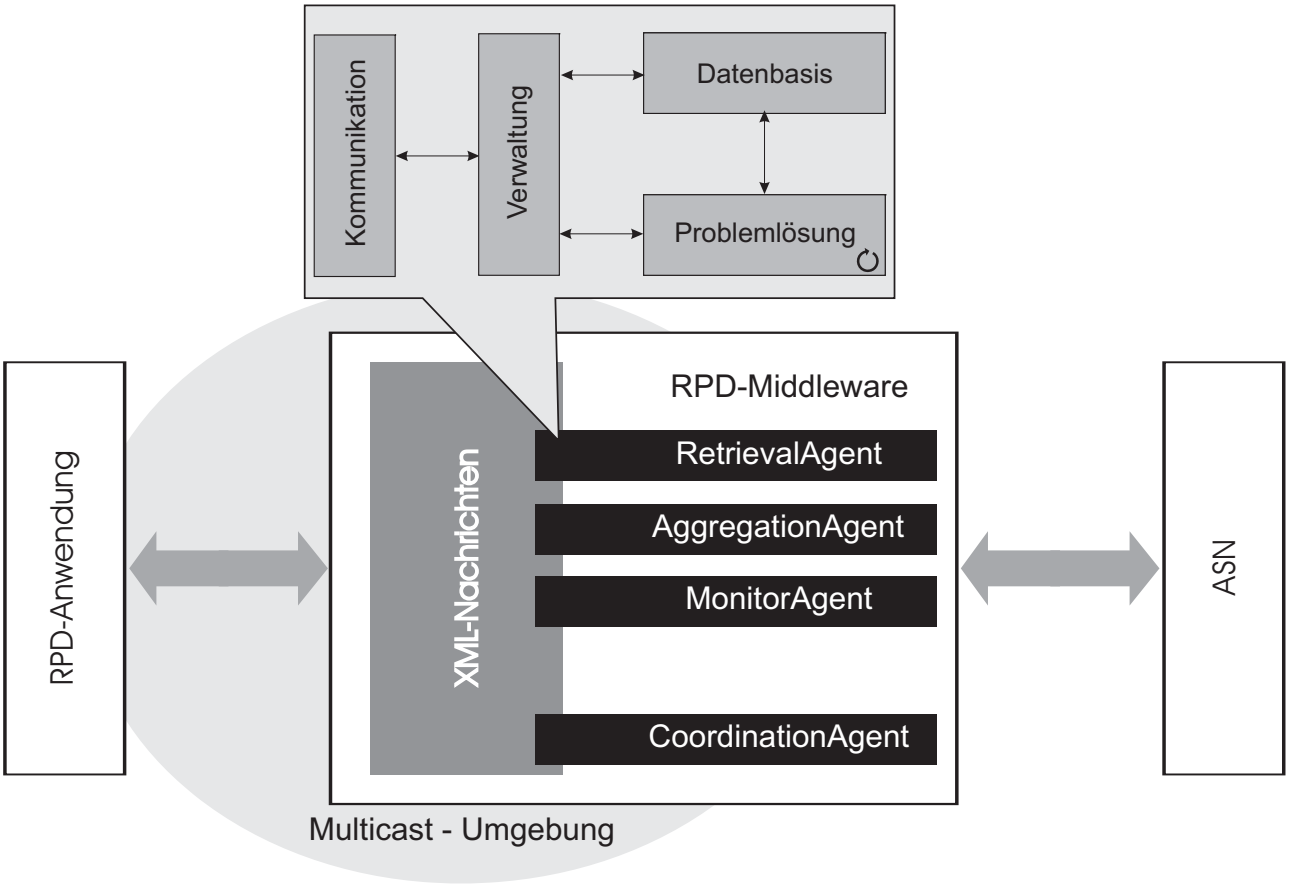


Bild 18: Interner Aufbau des Agenten

6 Entwicklung der spezifischen Agententypen

6.1 Retrieval-Agent

Der Retrieval-Agent hat die Aufgabe, eine effektive Wissensbeschaffung zu realisieren (siehe Abschnitt 4.1) [Dal 05]. Dafür werden Methoden benötigt, die den semantischen Informationsgehalt der Relationen im ASN so nutzen, damit das bereits konzeptualisierte Wissen um das relationsbasierte Wissen bereichert wird, ohne dabei die interne Struktur des ASN im Detail kennen zu müssen. Unter der internen Struktur wird das Datenmodell und nicht das Modell des ASN verstanden (siehe Abschnitt 2.2.6). Ist die interne Struktur bekannt, so kann eine scharfe Suche auf dem Datenbestand des ASN durchgeführt werden. Dabei ist für den RPD-Anwender nicht relevant, ob die gestellte Anfrage durch eine scharfe oder eine unscharfe Suche realisiert wird. Interessant ist nur die Findung des relevanten Wissens aus dem ASN. Nach der Abfrageformulierung wird erst mit scharfen Methoden gesucht und wenn keine zufrieden stellenden Ergebnisse geliefert werden, müssen unscharfe Methoden angewendet werden.

Die Qualität der Ergebnisse hängt zum einen davon ab, wie detailliert in einer Anfrage auf die Struktur des ASN Rücksicht genommen wird. Zum anderen kann die Qualität durch weitere Anfragen unter der Berücksichtigung der bereits gefundenen Ergebnisse gesteigert werden. Diese Vorgehensweise wird als navigierende Anfrage bezeichnet.

Der Ablauf einer Anfrage ist in Bild 19 als Aktivitätsdiagramm zu sehen. Der Retrieval-Agent wartet auf eine Anfrage, trifft diese ein wird sie in einem ersten Schritt syntaktisch geprüft. Nach positiver syntaktischer Prüfung, wird das ASN mit Hilfe der scharfen Suche durchsucht. Wurde dabei ein Ergebnis gefunden, wird dieses an die RPD-Anwendung zurückgegeben bzw. wird der Ergebnisraum im Rahmen der eingestellten Navigation erweitert. Liefert die scharfe Suche kein Ergebnis, so wird mit der unscharfen Suche fortgefahren. Die Ergebnisse der unscharfen Suche werden bewertet und ebenfalls der RPD-Anwendung präsentiert. Der Ergebnisraum kann auch hier durch die Navigation erweitert werden. Die Ergebnisse der unscharfen Suche werden nach der semantischen Nähe zwischen dem Ausgangskonzept und dem jeweiligen Ergebniskonzept sortiert.

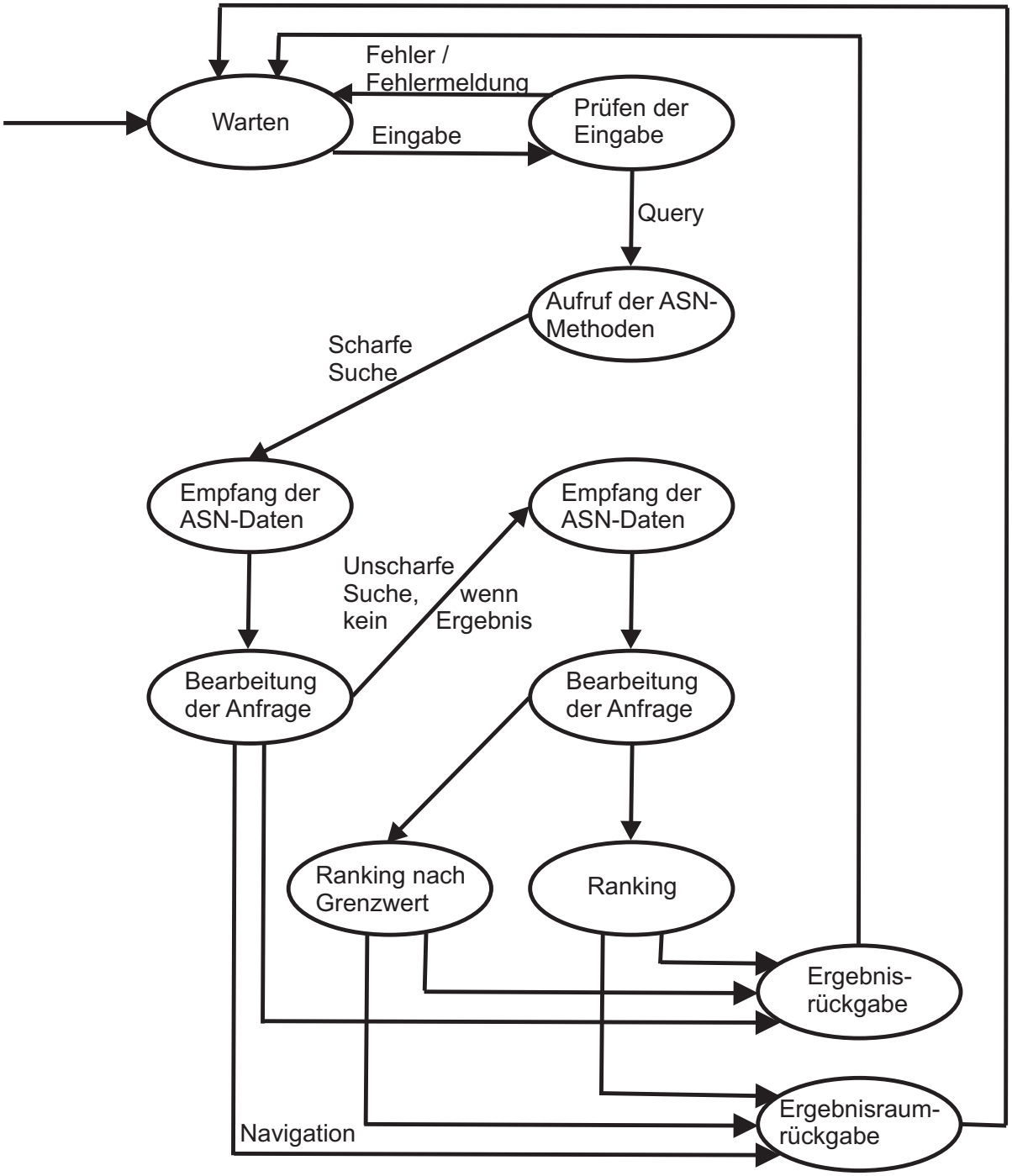


Bild 19: Zustände der Anfrage

Die Berechnung der semantischen Nähe [Opl 03] hängt zum einen davon ab, ob es sich bei der Art der Anfrage, um eine Assoziations-, Aggregations- oder Generalisierungssuche handelt (siehe Tabelle 5), und zum anderen von der Entfernung der Konzepte, d. h. sind die Konzepte über mehrere transitive Relationen verbunden, so fließt die Kombination der Relationen entlang des Pfades in die Berechnung der semantischen Nähe mit ein (siehe Tabelle 6). Der Ergebnisraum kann durch ein

RatingLimit begrenzt werden, wobei das *RatingLimit* die untere Grenze darstellt, die ein Ergebnis erfüllen muss.

Art der Suche	Assoziation	Aggregation	Komposition	Generalisierung
Assoziationssuche	hoch	mittel	mittel	niedrig
Aggregations- / Kompositionssuche	niedrig	hoch	hoch	mittel
Generalisierungssuche	niedrig	mittel	mittel	hoch

Tabelle 5: Bewertungstabelle für die Ergebnisse der unscharfen Suche anhand der Relation

Nachfolger / Vorgänger	Assoziation	Aggregation	Komposition	Generalisierung
Assoziation	hoch	mittel	mittel	niedrig
Aggregation	mittel	mittel	mittel	niedrig
Komposition	mittel	mittel	mittel	niedrig
Generalisierung	niedrig	niedrig	niedrig	niedrig

Tabelle 6: Bewertungstabelle für die Ergebnisse der unscharfen Suche bei transitiven Relationsfolgen

Anfragesprache

Für die Suche wurde aufgrund der Vorgaben durch das ASN (siehe Abschnitt 2.2.6) und den Anforderungen (siehe Abschnitt 4.1) eine eigene Anfragesprache die ASN-Query Language (ASN-QL) entwickelt, die ausschließlich für den Zugriff auf das Aktive Semantische Netz entworfen worden ist und sich an der XQuery orientiert. Die genaue Sprachdefinition findet sich im Anhang C.1, Tabelle 16.

Die entwickelte Anfragesprache legt eine Abstraktionsschicht über die Sprache XQuery [Boa 03]. XQuery stützt sich auf das XML-Query-Data-Model, das für semistrukturierte Daten am besten geeignet ist. Dabei werden arithmetische und logische Ausdrücke sowie Schleifen und Funktionen, Pfadausdrücke und ein mächtiger XQuery-Prozessor für Anfragen in ASN-QL zur Verfügung gestellt.

Eine der sieben Ausdrucksformen, die die XQuery zur Verfügung stellt, ist der For-Let-Where-Or-Return-Ausdruck (FLWOR) (siehe Tabelle 7). Der FLWOR-Ausdruck bei XML-Dokumenten entspricht den SQL-Anfragen [SQL 93] bei relationalen Datenbanken. Ein FLWOR-Ausdruck wird mit Hilfe von Templates aufgebaut und erlaubt die Anwendung von scharfen und unscharfen Methoden für die Informationsbeschaffung. Dabei wird der FLWOR-Ausdruck mit Hilfe eines Mappings (siehe Tabelle 7) durch die Semantik der Non-Terminal-Semantic (SelectNodes) generiert,

d. h. über das Nicht-Terminal *SelectNodes* wird die Verbindung zu den scharfen und unscharfen Suchmethoden der Informationsbeschaffung hergestellt. Zusätzlich dazu wird durch die *QFunction* des Ausdrucks die Return-Struktur bestimmt.

FLWORExpr	=	{ForClause | LetClause}- [WhereClause] [OrderByClause] ”return” ExprSingle;
ForClause	=	“for” “$” VarName “in” InputDocument/PathExpr {SelectNodes};
LetClause	=	“let” “$” VarName TypeDeclaration? “ :=” InputDocu-ment/PathExpr;
SelectNodes	=	(SharpMethods | UnsharpMethods);

Tabelle 7: EBNF des FLWOR-Ausdrucks

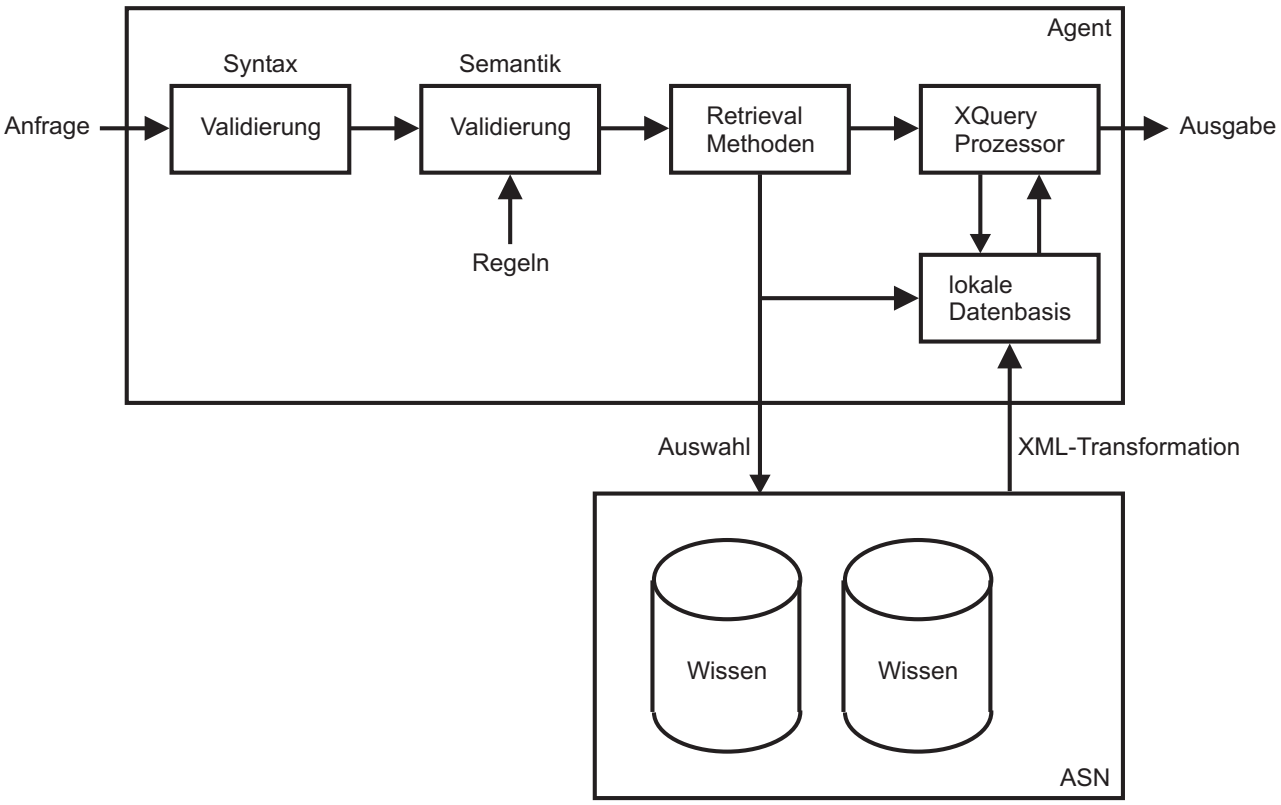


Bild 20: Abarbeitung einer Anfrage durch den Retrieval-Agent

In Bild 20 ist die Abarbeitung einer Anfrage durch den Retrieval-Agenten zu sehen. Der Agent erhält die Anfrage über die definierte Syntax zwischen den Data-Tags (siehe Abschnitt 5.2). Dabei wird der Datatype als ASN-QL-Request für die Bearbeitung als Anfrage festgelegt. Nachdem der Agent die Anfrage auf Syntax und Semantik überprüft hat, wird die Anfrage in ihre Bestandteile zerlegt. Für jeden Bestandteil wird der Suchraum in Abhängigkeit der verwendeten Retrieval-Methode (scharfe oder unscharfe Suche) aus dem ASN extrahiert und in einer XML-Repräsentation in der lokalen Datenbasis abgelegt. Anschließend wird die Anfrage

auf der XML-Repräsentation abgearbeitet und die Ergebnisse an die RPD-Anwendung bzw. den Aggregations-Agenten ausgeliefert.

Quantitative und qualitative Anfragen

Quantitative und qualitative Anfragen, wie sie im RPD vorkommen, wie z. B. *Wie viel?*, *Wie oft?* oder *Wer?* werden in der Spezifikation der Anfragesprache an den Retrieval-Agenten als *QFunction* bezeichnet. Dabei beschreibt die *QFunction*, obwohl sie ein Teil der Anfrageformulierung ist, ausschließlich die Form der Resultate, wodurch bereits bei der Anfrageformulierung die gewünschte Form des Ergebnisses festgelegt wird. Zusätzlich ist mit Hilfe der rekursiven Definition der *Semantic-Function*, die die eigentliche Suche beinhaltet, eine rekursive Suche möglich. Die rekursive Suche ermöglicht es, gefundene Teilmengen weiter zu durchsuchen, um so Ergebnisse weiter einschränken bzw. Teilaspekte miteinander verknüpfen zu können. Die *Range* gibt dabei an welche Information im Umfeld um das gefundene Ergebnis mit ausgegeben werden soll. Der Wert des *Ranges* entspricht der Anzahl der Relationen über die ein entferntes Konzept am gefundenen Ergebnis-Konzept verbunden ist.

Navigation

Weitere Verfeinerungen der Anfrage bestehen über die optionalen Mechanismen *Restriction* und *Navigation*. Unter *Restriction* wird die Möglichkeit verstanden, Abfragen mit einfachen Bedingungen zu verknüpfen. Diese Bedingungen beinhalten Vergleiche von Attributwerten mit festen Größen. Im Gegensatz dazu wird mit Hilfe der Navigation der Suchraum um die angegebene Größe (Range) erweitert. Auch hier ist die *Range* ein Maß, wie weit um das Ergebnis-Konzept herum navigiert werden darf. Um die zusätzlichen Informationen der Navigation zu erhalten wird die Anfrage mit einem Plus-Zeichen abgeschlossen bzw. mit einem Minus-Zeichen, wenn die Navigation nicht benötigt wird (siehe Tabelle 16 in Anhang C.1).

Scharfe Suche

Bei der scharfen Suche werden die Konzepte so detailliert in der Anfrage beschrieben, z. B. durch die eigene ID oder den Namen, dass genügend Information für eine erfolgreiche Suche vorliegt. Die Erweiterung der Sprachdefinition für die scharfen Methoden sind im Anhang C.1 in Tabelle 17 dargestellt.

Die intern im Retrieval-Agenten verwendeten Methoden für die scharfe Suche sind:

- **get-concept**: Diese Funktion sucht alle im ASN existierenden Konzepte, deren Attribute und Relationen, die mit der Beschreibung (*ConceptDescr*), wie sie in der Anfrage formuliert ist, übereinstimmen.
- **is-associated-with**: Mit dieser Funktion werden alle assoziierten Konzepte gesucht. Für die Suche werden als Parameter entweder ein oder mehrere

Startkonzepte und die Größe der zu durchsuchenden Umgebung herangezogen oder ein oder mehrere Start- und Zielkonzepte, ebenfalls mit einer beschränkten Umgebung. Im zweiten Fall ist dann das Ergebnis die Menge von Zielkonzepten für die ein assoziierter Pfad zwischen dem betrachteten Start- und Zielkonzept existiert.

- **is-aggregated-of**: Diese Funktion arbeitet in gleicher Weise wie *is-associated-with*, mit dem Unterschied das hier nicht Konzepte gesucht werden, die in einem assoziierten Verhältnis stehen, sondern Konzepte die in einer Aggregations-Beziehung zu den als Parameter übergebenen Konzepten stehen.
- **is-composed-of**: Diese Funktion betrachtet im Unterschied zu den vorangegangenen Funktionen bei gleicher Arbeitsweise die Kompositionsbeziehung.
- **is-generalization-of**: Diese Funktion ist identisch zu den vorangegangenen Funktionen mit dem Unterschied, dass die Generalisierung anstatt z. B. der Komposition berücksichtigt wird.

Die implementierten Algorithmen der Methoden sind im Anhang D.1.1 zu finden.

Unscharfe Suche

Methoden zur unscharfen Suche werden genutzt, um für unvollständige und strukturell nicht exakte Abfragen Ergebnisse aufzufinden. Durch die unscharfe Suche wird der Ergebnisraum mit weiteren Informationen aus dem ASN ergänzt, die nur teilweise die Suchbedingungen erfüllen. Eine weitere Methode der unscharfen Suche ist unabhängig von der Strukturierung der Daten, Informationen aufzufinden. Dabei werden im ASN alle Informationen gesucht, unabhängig ob sie als Konzept, Attribut oder Relation abgelegt sind.

Die Erweiterung der Sprachdefinition für die unscharfen Methoden sind im Anhang C.1 in Tabelle 18 dargestellt.

Die intern im Retrieval-Agenten verwendeten Methoden für die unscharfe Suche sind:

- **get-concept-unsharp**: Diese Funktion dient zum Auffinden aller Konzepte, deren Attribute teilweise mit denen der Anfrage übereinstimmen.
- **is-related-with**: Diese Funktion sucht alle Konzepte, die zueinander in Beziehung stehen, d. h. die miteinander über eine oder mehrere Relationen verbunden sind. Über die Vorgabe eines *Ranges* wird der Suchraum begrenzt. Die gefundenen Relationen werden gewichtet und damit die semantische Nähe, also der inhaltliche Verwandtschaftsgrad, bestimmt. Das Ergebnis ist damit im besten Fall eine Liste von Konzepten, geordnet nach der semantischen Nähe. Durch die Definition eines Zielkonzepts neben dem Startkonzept wird der Suchraum auf alle Relationswege zwischen den beiden Konzepten begrenzt.

- **consists-of**: Diese Funktion fasst die Methoden der scharfen Suche *is-associated-with*, *is-aggregated-of* und *is-composed-of* zusammen, wodurch alle Konzepte gefunden werden, die in irgendeiner direkten Verbindung zueinander stehen.
- **is-similar-to**: Diese Funktion sucht zu einem übergebenen Konzept ähnliche Konzepte. In einem ersten Schritt werden alle Konzepte anhand ihrer semantischen Beziehung zum übergebenen Konzept gesucht. Anschließend werden die gefundenen Konzepte und das zu vergleichende Konzept in einen Vektorraum, sortiert nach der Anzahl ihrer übereinstimmenden Attribute, eingetragen. Die Gewichtung der Attribute ergibt sich aus dem Wert der Attribute, falls dieser quantifizierbar ist. Mit dem Kosinus-Maß [Fer 03a] wird die Ähnlichkeit für alle Konzepte bestimmt. Dabei wird ein n-dimensionaler Raum aufgespannt, wobei n die Anzahl der Attribute der zu vergleichenden Konzepte ist. Über die Werte der Attribute jedes Konzepts ist ein Vektor im Raum verbunden. Je kleiner der Winkel α zwischen zwei Vektoren K_1 und K_2 ist, desto ähnlicher sind die Konzepte, die durch diese Vektoren repräsentiert sind. Die Berechnung erfolgt nach der Formel:

$$\cos\alpha = \frac{\sum_{i=1}^{n} \vec{K}_{1_i} \vec{K}_{2_i}}{\sum_{i=1}^{n} \left|\vec{K}_{1_i}\right| \left|\vec{K}_{2_i}\right|}$$ wobei n=Anzahl der Attribute des Konzepts

Die implementierten Algorithmen der Methoden sind im Anhang D.1.2 zu finden.

Eine Zuordnung der Sprachelemente der Anfragesprache des Retrieval-Agenten zu den scharfen und unscharfen Retrieval-Methoden ist in Tabelle 8 dargestellt.

Semantik	scharfe Methoden	unscharfe Methoden
besteht-aus	is-aggregated-of is-composed-of	consists-of is-related-with
ist-ähnlich	–	is-similar-to
hat-Beziehung-zu	is-associated-with	is-related-with
finde-Konzept	get-concept	get-concept-unsharp
ist-abgeleitet	is-generalization-of	is-related-with

Tabelle 8: Abbildung der Semantik auf Retrieval-Methoden

Beispiel

In Bild 21 ist eine Beispielanfrage dargestellt, formuliert als direkte Anfrage im XML-Format an den Retrieval-Agenten inklusive der Kommunikationsparameter (siehe Abschnitt 5.2). Dabei wird folgender Frage nachgegangen: „Wie viele Motoren mit dem Namen MotorA stehen in Beziehung zu dem Konzept namens Auto?“. Die Su-

che soll auf fünf Konzepte beschränkt werden. Würde die Anfrage durch die Benutzung der scharfen Methode *is-associated-with* kein Ergebnis liefern, wird anschließend die unscharfe Methode *is-related-with* angewendet. Die Aufbereitung der Ergebnisse wird durch das Terminal *Wie viele* generiert, d. h. es werden nicht die gefundenen Konzepte zurückgegeben sondern die Anzahl.

Die für die Suchanfrage notwendigen Nachrichten, die mit der RPD-Middleware ausgetauscht werden, sind im Anhang E.1 beschrieben.

```
<REQUEST sender="me" destination="searcher1/RetrievalAgent/01" reqcontrol=""
dataType="ASNQLRequest">
<DATA>
      wieviele hat-Beziehung-zu(CONCEPT NAME MotorA, CONCEPT NAME
      Auto, 5)
</DATA>
```

Bild 21: Beispiel einer Anfrage an den Retrieval-Agenten

6.2 Aggregations-Agent

Eine Aggregation als Repräsentationsform, die eine Beziehung zwischen zwei Konzepten beschreibt, wird durch das Aggregat und seine Bestandteile definiert, d. h. ein Konzept besteht aus vielen Konzepten, die als Bestandteile des Hauptkonzeptes (Aggregat) anzusehen sind. Analog dazu können komplexe Problemstellungen in Problemteile P_1, P_2, … P_x unterteilt werden. Die vorliegenden Aggregationsteile werden nach einer Abarbeitung in Teilergebnisse E_1, E_2, … E_x transformiert und anschließend als Ergebnis innerhalb des Aggregations-Agenten zusammengeführt. Die aus dem RPD-Prozess gestellten Anforderungen an den Aggregations-Agenten sind sowohl die Umformung von komplexen Anfragen als auch die Auswahl einer anwendungsbezogenen Sicht bzw. der nutzerspezifischen Sicht, um eine weitere Verfeinerung des Ergebnisses zu erzielen (siehe Abschnitt 4.1). Zusätzliche Anforderungen betreffen das Performance-Problem, das im Zusammenhang mit der gesamten Systemperformance steht, z. B. die schnelle Abwicklung der Anfragen. Durch den Einsatz des Aggregations-Agenten werden die einzelnen Anfragen auf parallel arbeitende Retrieval-Agenten verteilt.

Obwohl der Aggregations-Agent durch seine eingebettete Intelligenz die Möglichkeit besitzt, mit allen anderen Agententypen zu interagieren, wird er ausschließlich für den Einsatz mit dem Retrieval-Agenten entworfen. Bei der Anfrage bzw. dem Anfrageergebnis spricht man über Aggregationsanfrage bzw. Aggregationsergebnis der RPD-Anwendung oder des RPD-Nutzers. Die von dem Aggregations-Agenten ausgehenden Anfragen werden an die Retrieval-Agenten weitergeleitet (siehe Bild 22), die die gewünschte Suche auf dem ASN durchführen und nach dem Abwickeln der Anfragen die Ergebnisse an den Aggregations-Agenten zurückliefern, der die entsprechende Aufbereitung des Aggregationsergebnisses übernimmt. Die be-

reits entworfene Agentenarchitektur innerhalb des Multiagentensystems stellt alle benötigten Komponenten für den Aggregations-Agenten wie z. B. die Multicast-Kommunikation zur Verfügung. Der Aggregations-Agent verhält sich gegenüber den Retrieval-Agenten wie eine RPD-Anwendung.

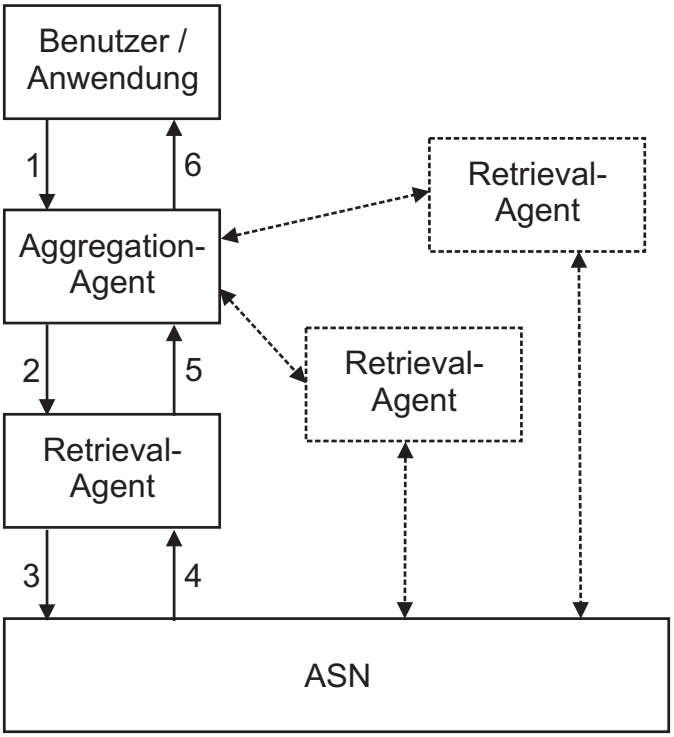


Bild 22: Kommunikationsablauf zwischen dem Aggregations-Agenten und den Retrieval-Agenten

Die zusätzliche Intelligenz des Aggregations-Agenten wird nicht nur über die Interpretation der gewünschten Sicht des Retrieval-Agenten-Ergebnisses widergespiegelt, sondern durch die Möglichkeit der expliziten Formulierung einer Anfrage an den Aggregations-Agenten und das daraus entstehende Ergebnis. Das neue Ergebnis besteht aus der Kombination von verschiedenen Konzepten und Attributen.

Anfragesprache

Mit Hilfe von XML wird eine adäquate Repräsentation der Aggregation erreicht. Da XML eine Metaebenen-Sprache ist, können für bestimmte Problembereiche verschiedene Auszeichnungssprachen spezifiziert werden. Die für die Aggregation entworfene Sprache ist in Form einer DTD im Anhang C.2 in Bild 47 dargestellt. Durch die Definition der Sprachelemente wird eine Beziehung zwischen den XML-Dokumenten und der DTD-Spezifikation hergestellt. Diese Beziehung ist eine Daten- zu Metadaten-Beziehung. Das XML-Dokument stellt die Semantik der Aggregationsanfrage mit der zusätzlichen Angabe über den gewünschten Typ des Ergebnisses als Parameter *resultType* dar. Die eigentlichen Such-Anfragen, die an die Retrieval-Agenten gestellt werden, werden zwischen den *QuerySource*-Tags in

der Form der Anfragesprache des Retrieval-Agenten formuliert. Die Aggregationsergebnisse werden aus dem Aggregations-Agenten entsprechend vorbereitet und dem RPD-Anwender bzw. der RPD-Anwendung in derselben Form präsentiert wie die Anfrage, d. h. die im Anhang C.2 in Bild 47 dargestellte DTD gilt auch für die Aggregationsergebnisse.

Beispiel

```
<Aggregation name="Teamleitung" type="" context="1">
        <Element name="Mitarbeiter" type="mitarbeiter">
                <Query id="0" name="Mitarbeiter" resultType="Mitarbeiter">
                        <QuerySource queryType="asnql">
                                welche findekonzept(type = Mitarbeiter, 0) alter < 36
                                AND  funktion = Teamleiter -
                        </QuerySource>
                </Query>
        </Element>

        <Element name="Entwicklung" type="int">
                <Query id="1" name="Entwicklung" resultType="int">
                        <QuerySource queryType="asnql">
                                wieviele bestehtaus(hatbeziehungzu(name =
                                Entwicklung, type = Team, 2), type = Projekt, 1) -
                        </QuerySource>
                </Query>
        </Element>
</Aggregation>
```

Bild 23: Beispiel einer Aggregationsanfrage

In Bild 23 ist ein Beispiel für das Finden eines bestimmten Teamleiters dargestellt. Dabei gilt es die Anfrage, einen Teamleiter zu finden, der unter 36 Jahre alt ist, mit der Suche nach der maximalen Anzahl an durchgeführten Projekten des Teams Entwicklung, zu vereinen. Durch das Minuszeichen am Ende der Anfrage wird spezifiziert, dass auf die Navigation verzichtet wird. Das Ergebnis (siehe Bild 24) ist die Zusammensetzung der gewünschten Eigenschaften des Teamleiters und der Informationen über das Team *Entwicklung*. Es werden keine Suchkriterien definiert, sondern es werden bereits vorhandene Teilaspekte, die im ASN verteilt abgelegt sind zu neuem Wissen zusammengeführt.

```
<Aggregation name="Teamleitung" type="" context="1">
        <Element name="Mitarbeiter" type="mitarbeiter">
                <Result queryId="0">
                        <concept name="Diederich" type="Mitarbeiter" id="39">
                                <searchinfo rating="0" retrievedby="findekonzept"
                                foundrelatedto="empty" direction="empty" />
                                <attributes>
                                        <attribute name="alter">32</attribute>
                                        <attribute name="vorname">Michael</attribute>
                                        <attribute name="teamrolle" />
                                        <attribute name="personalkosten">leer</attribute>
                                        <attribute name="nachname">Diederich</attribute>
                                        <attribute name="funktion">Teamleiter</attribute>
                                </attributes>
                                <relations>
                                        <relation name="Geschäftsadresse" multiplicity=
                                        "one" direction="Mitarbeiter" type="Aggregation">
                                        Geschäftsadresse Diederich</relation>
                                        <relation name="Teamzugehörigkeit" multiplicity=
                                        "one" direction="Team" type="Aggregation">
                                        AK 10</relation>
                                        <relation name="Abtzugehörigkeit" multiplicity=
                                        "one" direction="ungerichtet" type="Assoziation">
                                        IAT</relation>
                                </relations>
                        </concept>
                </Result>
        </Element>

        <Element name="Entwicklung" type="int">
                <Result queryId="1">2</Result>
        </Element>
</Aggregation>
```

Bild 24: Das Aggregationsergebnis des Beispiels

Die zwischen RPD-Anwendung und Aggregations-Agent, sowie zwischen Aggregations-Agent und den Retrieval-Agenten ausgetauschten Nachrichten sind im Anhang E.2 näher erläutert.

6.3 Monitor-Agent

Der Monitor-Agent hat zum Ziel, das ASN auf bestimmte Veränderungen zu überwachen und diese der RPD-Anwendung zu melden. Der Schwerpunkt wird hierbei auf die Entwicklung einer Überwachungsmethode gelegt, die es erlaubt, nicht nur

einzelne Attribute im ASN auf Veränderungen zu beobachten, sondern die auch komplexe Strukturen und Abhängigkeiten im ASN berücksichtigt bis hin zu einfachen Funktionen zur Aufbereitung der ASN-Inhalte, aber auch der ASN-Zugriffe (siehe Abschnitt 4.2).

Für die Überwachungsbedingung bestimmt der Monitor-Agent als Ergebnis den Wahrheitsgehalt. Das Ergebnis *true* oder *false* wird der RPD-Anwendung nach der Instanziierung des Monitor-Agenten erstmalig präsentiert, sowie bei jeder Veränderung des Ergebnisses. Die Berechnung des Wahrheitsgehalts findet bei jeder Veränderung von Werten bzw. bei Strukturanpassungen im ASN statt, sofern diese direkten Einfluss auf die an den Monitor-Agenten gestellte Bedingung haben. Ändert sich der Wahrheitsgehalt der Bedingung bei einer Veränderung im ASN nicht, so wird dies der RPD-Anwendung auch nicht angezeigt.

Anpassungen des ASN

Der Vorteil des Monitor-Agenten gegenüber der RPD-Anwendung beim Überwachen des ASN liegt darin, dass der Monitor-Agent, wie es die RPD-Anwendung müsste, nicht zyklisch das ASN abfragt, sondern gezielt durch das ASN über Veränderungen informiert wird. Zu diesem Zweck wurde das ASN derart angepasst, dass der Monitor-Agent sich in den einzelnen Netzen, Konzepten, Attributen und Relationen registrieren kann, wodurch das ASN dann den Agenten bei Veränderungen der entsprechenden ASN-Elemente informiert. Diese Registrierung hat nur über die Lebensdauer des Monitor-Agenten bestand und wird für jeden Monitor-Agenten getrennt durchgeführt.

Jede gemeldete Veränderung zwingt den Monitor-Agenten eine Neuberechnung der Bedingungsanweisung durchzuführen. Sollte sich dabei der Wahrheitswert der Bedingung ändern, wird dieser der RPD-Anwendung mitgeteilt. Durch diese Vorgehensweise wird ein aktives Warten der RPD-Anwendung und des Monitor-Agenten vermieden.

Anfragesprache

Um dieses Ziel zu erreichen wird eine einfache Anfragesprache entwickelt die den folgenden, durch die RPD-Anwendungen gestellten Anforderungen genügt.

1. Vergleich von konkreten Inhalten (Attribute) im ASN mit anderen Attributen oder Fixwerten.

2. Aufbau von komplexen Vergleichen, die mehrere Vergleiche berücksichtigen.

3. Aufbau von einfachen Funktionen zur Überwachung einfacher Vorgänge im ASN.

Die Anfragesprache orientiert sich an den in der Programmierung üblichen Bedingungsanweisungen (IF-Statements). Die erste Bedingung, der Vergleich von konkreten Inhalten, wird in der Anfragesprache durch einfache logische Vergleiche wie =, <>, <=, >=, < oder > erreicht. Die zweite Bedingung, der Aufbau von komplexen Vergleichen, wird dadurch erfüllt, dass die einfachen logischen Vergleiche durch aussagenlogische Operatoren wie *AND*, *OR*, *NAND*, *NOR*, *XOR* sowie die Negation *NOT* miteinander verknüpft werden können.

Bei der dritten Bedingung, der Überwachung einfacher Abläufe im ASN, werden drei Gruppen von Funktionen unterschieden. Die erste Gruppe bietet Funktionen zur Überwachung von Struktur- und Inhaltsänderungen des ASN an. Diese Funktionen werden im Folgenden Strukturfunktionen genannt. Die zweite Gruppe (Zugriffsfunktionen) überwacht den Zugriff auf das ASN. Bei der dritten Gruppe von Funktionen handelt es sich um mathematische Funktionen zur Berechnung einfacher Werte. Die Berechnungsergebnisse der Funktionen sind dann Bestandteil der Vergleichsoperationen.

1. Die Strukturfunktionen liefern als Berechnungsergebnisse aussagenlogische Werte, die entweder als einzige Bedingung dem Monitor-Agenten zur Auswertung übergeben oder in komplexen Vergleichen weiterverarbeitet werden können. Es stehen folgende Strukturfunktionen zur Auswahl:

    - **ChangeVersion**: Es wird überwacht, ob sich die Version eines Wissensknoten (Konzept), eines Inhalts (Attribut) oder eines semantischen Zusammenhangs (Relation) ändert.
    - **ChangeAttributeValue**: Es wird überwacht, ob sich der Wert eines bestimmten Attributs ändert.
    - **DeleteAttribute**: Es wird überwacht, ob ein bestimmtes Attribut gelöscht wird.
    - **DeleteConcept**: Es wird überwacht, ob ein bestimmtes Konzept gelöscht wird.
    - **DeleteNet**: Es wird überwacht, ob ein bestimmtes Netz gelöscht wird.
    - **DeleteRelation**: Es wird überwacht, ob eine bestimmte Relation gelöscht wird.
    - **IncludeConcept**: Es wird überwacht, ob eine bestimmte Relation ein bestimmtes Ziel-Konzept enthält. Unter einem Ziel-Konzept wird ein Konzept verstanden, auf das die Relation hinführt.

2. Die Zugriffsfunktionen liefern als Ergebnis Zahlenwerte zurück, die entweder direkt mit anderen Zahlenwerten auf ihre Ordnung hin verglichen oder als Parameter mathematischer Funktionen verwendet werden können. Es stehen folgende Zugriffsfunktionen zur Auswahl:

- **CountRead**: Es werden die Lese-Zugriffe auf das Value-Feld des Attributs gezählt, die seit der Instanziierung des Attributs vorgenommen wurden.
- **CountWrite**: Es werden die Schreib-Zugriffe auf das Value-Feld des Attributs gezählt, die seit der Instanziierung des Attributs vorgenommen wurden.
- **Version**: Es wird die Versionsnummer eines Attributs bestimmt. Die Versionsnummer hängt von der Anzahl der Schreib-Zugriffe auf die Felder des Attributs ab, d. h. es wird die Versionsnummer nicht nur bei Veränderung des Attribut-Wertes erhöht sondern auch beispielsweise bei Veränderung des Attribut-Typs. Des Weiteren ist eine direkte Manipulation der Versionsnummer möglich, sodass diese durch direkten Zugriff auf das ASN durch eine RPD-Anwendung gesetzt werden kann.

3. Die mathematischen Funktionen liefern als Ergebnis Zahlenwerte zurück, die entweder direkt mit anderen Zahlenwerten auf ihre Ordnung hin verglichen oder als Parameter weiterer mathematischer Funktionen verwendet werden können. Diese mathematischen Funktionen sind:

   - **Add**: Addition zweier Werte.
   - **Sub**: Subtraktion zweier Werte.
   - **Mul**: Multiplikation zweier Werte.
   - **Div**: Division zweier Werte.

Im Anhang C.3 in Tabelle 19 ist die Syntax der Anfragesprache dargestellt.

Beispiel

Im Beispiel Bild 25 ist die Anfrage dargestellt, die überprüft, ob ein Mitarbeiter *Person1* Mitglied des Teams *Entwicklung* und dessen Status gleichzeitig *anwesend* ist, wobei die Konzepte *Person1* und *Entwicklung* dem Netz *Luftdüse* angehören, das Attribut mit dem Namen *Status* den Status des Mitarbeiters wiedergibt und die Relation zwischen Team und Mitarbeiter den Namen *Teamzugehörigkeit* trägt. Dieses Beispiel zeigt sowohl die Strukturfunktion *IncludeConcept* als auch den direkten Vergleich zweier Werte vom Typ *String*. Die überwachten Daten orientieren sich an dem im Abschnitt 2.2.6 beschriebenen ASN.

```
(IncludeConcept (Luftdüse, Entwicklung, Teamzugehörigkeit) (Luftdüse, Person1))
AND (String(Value(Luftdüse, Person1, Status)) = String(anwesend))
```

Bild 25: Beispiel 1 einer Monitoranfrage

Im zweiten Beispiel Bild 26 meldet sich der Monitor-Agent, wenn sich der Status des Mitarbeiters *Person1* mehr als *fünfmal* geändert hat, wodurch die Benutzung der Zugriffsfunktionen (*CountWrite*) demonstriert wird.

CountWrite (Luftdüse, Person1, Status) > Integer(5)

Bild 26: Beispiel 2 einer Monitoranfrage

Kommunikationsprotokoll

In Bild 27 ist das Sequenz-Diagramm des Kommunikationsablaufs zwischen RPD-Anwendung, Agentensystem, Monitor-Agent und ASN für das Beispiel der zweiten Anfrage (Bild 26) dargestellt. Die Definitionen der ausgetauschten Nachrichten finden sich im Anhang E.3.

Der Kommunikationsablauf teilt sich in drei Abschnitte auf. Im Abschnitt A wird der Monitor-Agent gestartet und initialisiert. Im Abschnitt B erfolgt die eigentliche Arbeit des Agenten. Die Nachrichten im Abschnitt C dienen zum Beenden des Monitor-Agenten.

Im Abschnitt A richtet zuerst die RPD-Anwendung die Monitor-Anfrage (1) an das Agentensystem. Der Master-Agent empfängt die Multicast-Nachricht, instanziiert einen Monitor-Agenten und gibt diesem die Monitoranfrage (2) weiter. Dieser meldet der RPD-Anwendung seine Anwesenheit. Daraufhin schickt die RPD-Anwendung ihren Auftrag (3) an den Monitor-Agenten, der diese zuerst syntaktisch prüft (4) und die entsprechenden Überwachungspunkte im ASN setzt. Durch das Setzen der Überwachungspunkte im ASN wird eine semantische Überprüfung vorgenommen. Mit dessen Hilfe festgestellt wird, ob alle benötigten Informationselemente im ASN vorhanden sind. Ist dieser Vorgang fehlerfrei abgelaufen, wird dies der RPD-Anwendung mitgeteilt. Ansonsten wird eine Fehlernachricht übermittelt. Anschließend wird die Überwachungsbedingung erstmalig ausgewertet (5) und das Ergebnis der RPD-Anwendung mitgeteilt (6).

Während der Monitoringphase B werden die laufend empfangenen Veränderungen im ASN (7) bewertet und bei Änderung des Wahrheitsgehalts der Bedingung der neue Wahrheitswert der RPD-Anwendung mitgeteilt (8). Parallel hierzu wird das ASN zyklisch auf Verfügbarkeit getestet und im Fehlerfalle die RPD-Anwendung informiert.

Benötigt die RPD-Anwendung den Monitor-Agenten nicht mehr, teilt sie dies dem Agenten mit (9), der daraufhin die Überwachungspunkte im ASN entfernt und sich selbst beendet (siehe Abschnitt C).

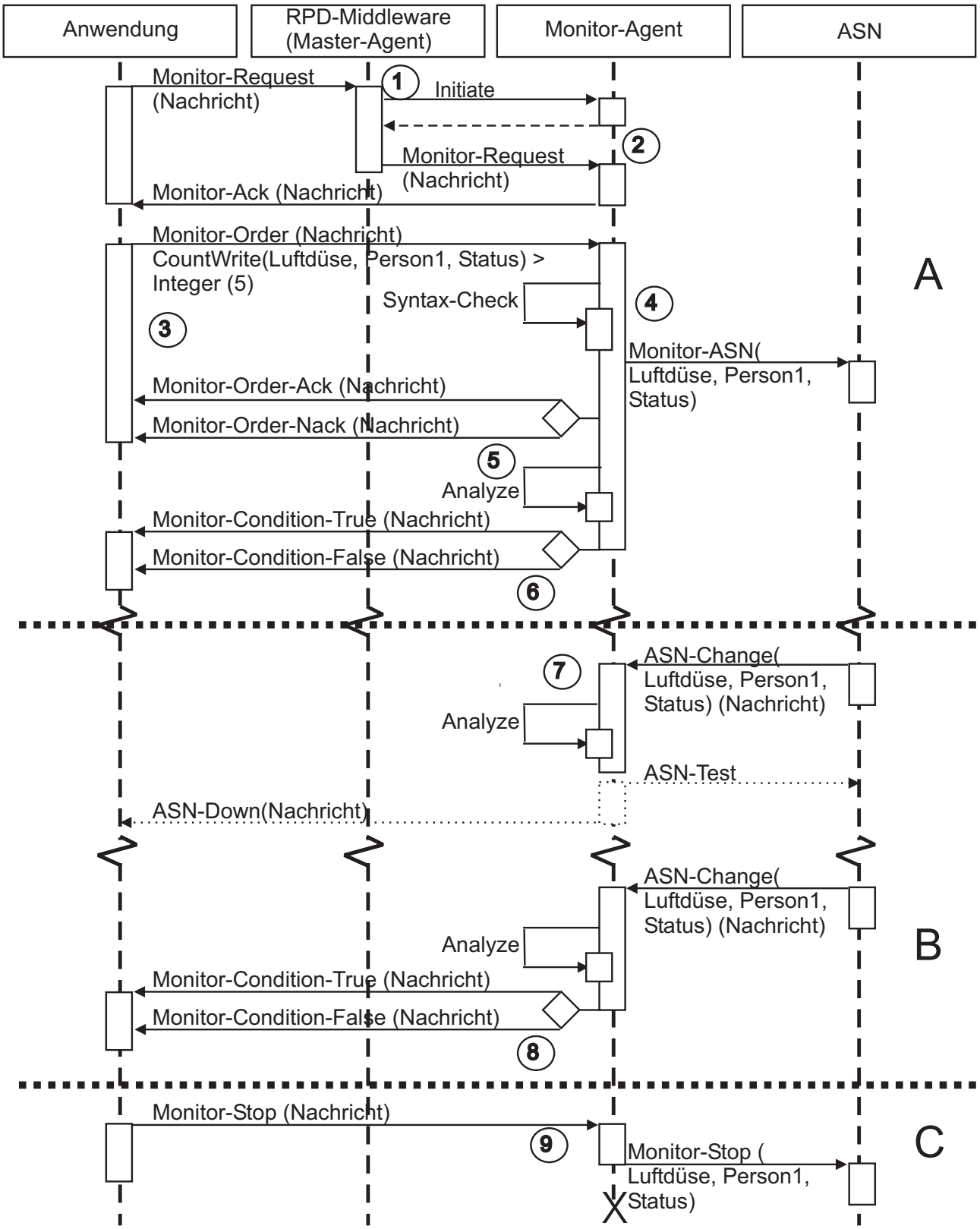


Bild 27: Sequenz-Diagramm eines typischen Kommunikationsablaufs für den Monitor-Agenten

6.4 Koordinations-Agent

Die Zielsetzung des Koordinations-Agenten ist, Abläufe innerhalb einer RPD-Anwendung, zwischen Instanzen einer RPD-Anwendung, aber auch zwischen mehreren verschiedenen RPD-Anwendungen zu koordinieren. Da es sich hierbei um Anwendungen unterschiedlicher Domänen handelt und die Koordinationsaufgaben sich ständig ändern (siehe Abschnitt 4.3), ist ein starr vorgegebenes Koordinationsprotokoll nicht sinnvoll. Der Koordinations-Agent muss daher in der Lage sein, ein frei definiertes Koordinationsprotokoll abarbeiten zu können.

Als flexible Beschreibungsform eines Koordinationsprotokolls wird das Zustandübergangsdiagramm gewählt. Die Zustände repräsentieren Situationen im Koordinationsprotokoll, die Übergänge werden mit Hilfe von bestimmten durch die RPD-Anwendung vorgegebenen Nachrichten initiiert und es werden an die RPD-Anwendungen bestimmte, vordefinierte Ausgabenachrichten signalisiert. Die Ein- und Ausgabenachrichten bestehen aus einem Textbaustein. Die Eingabenachricht wird durch den bzw. die Sender gekennzeichnet, während die Ausgabenachricht durch den bzw. die Empfänger markiert sind. Die Kennzeichnung der Nachrichten hat den Zweck, einen Zustandsübergang abhängig vom Sender auslösen zu können, d. h. der Weg durch das Koordinationsprotokoll ist abhängig vom Sender, der den Koordinationsschritt initiiert hat. Durch die Markierung der Ausgabenachricht mit Hilfe einer Empfängerliste wird der Kreis der Empfänger, der über den Zustandswechsel zu informieren ist, festgelegt. Natürlich ist es möglich, die Ausgabenachrichten an alle Koordinationspartner zu schicken und die Eingabenachrichten von allen Koordinationspartnern zu akzeptieren. Diese Vorgehensweise ist von Bedeutung bei der Koordination zweier oder mehrerer Koordinationspartner. Es kann dadurch die Reihenfolge, wie ein Prozess koordiniert werden soll, festgelegt werden.

Das Zustandübergangsdiagramm wird im Koordinations-Agenten durch einen endlichen Automaten [Büc 89], [Car 89], [Pin 89] realisiert. Die Definition der Zustandsübergänge des endlichen Automaten wird gegenüber der Standarddefinition des Zustandübergangsdiagramm so erweitert, dass es möglich ist, die Eingabe vom Sender abhängig zu machen und die Ausgabe auf eine Empfängermenge beschränken zu können. Zu diesem Zweck wird jeder Eingabe eine Liste von möglichen Sendern und jeder Ausgabe eine Liste von Empfängern hinzugefügt.

Bei der Instanziierung des Koordinationsprotokolls entstehen zwei Probleme. Zum einen muss allen beteiligten RPD-Anwendungen das Zustandübergangsdiagramm und damit das Koordinationsprotokoll bekannt sein, und zum anderen muss der benutzte Koordinations-Agent allen Koordinationspartnern und umgekehrt bekannt sein (siehe Abschnitt 4.3).

Das erste Problem lässt sich nur durch die RPD-Anwendung lösen. So müssen allen an der Koordination beteiligten RPD-Anwendungen das Koordinationsprotokoll

und damit das Zustandübergangsdiagramm bekannt sein. Dieses Problem ist angesichts der Tatsache, dass das Koordinationsziel von den RPD-Anwendungen vorgegeben wurde, als gering zu betrachten.

Das zweite Problem, den richtigen Koordinations-Agenten aufzufinden, wird dadurch gelöst, dass das Koordinationsprotokoll mit einem eindeutigen, allen Koordinationspartnern bekannten Namen versehen wird. Dieser Name wird von einer RPD-Anwendung festgelegt und muss dabei ebenso wie das Koordinationsprotokoll allen beteiligten RPD-Anwendungen bereits vor der Instanziierung und Benutzung des Koordinations-Agenten bekannt sein. Durch Senden einer Anmeldenachricht an die RPD-Middleware, die den Namen des Koordinationsprotokolls beinhaltet, wird der korrekte Koordinations-Agent aufgefunden. Alle Nachrichten an den Koordinations-Agenten werden durch den Namen des Koordinationsprotokolls ausgezeichnet. Dadurch werden die Nachrichten nur von dem Koordinations-Agenten bearbeitet, der für dieses Koordinationsprotokoll verantwortlich ist. Es ist zu beachten, dass der Name des Koordinationsprotokolls innerhalb der RPD-Middleware eindeutig sein muss.

Um eine Bewertung der eingehenden Nachrichten nach Sendern, sowie die Zustellung der ausgehenden Nachrichten an die richtigen Empfänger durchführen zu können, müssen alle an der Koordination beteiligten RPD-Anwendungen mit einem innerhalb des Koordinationsprotokolls eindeutigen Namen versehen werden. Diese Koordinationspartner-IDs werden bereits mit der Definition des Koordinationsprotokolls festgelegt und müssen von allen Koordinationspartnern benutzt werden. Als Koordinationspartner-IDs können beispielsweise die Unique Agent Identifier (UAI) zum Einsatz kommen.

Automatendefinition / Anfragesprache

In Tabelle 20 im Anhang C.4 ist eine Sprache zur Beschreibung des Zustandübergangsdiagramms angegeben. Dieses Zustandübergangsdiagramm wird dann dem Koordinations-Agenten als Koordinationsprotokoll zur weiteren Bearbeitung übergeben. Zum einen werden die Zustände mit eindeutigen Namen angegeben und zum anderen alle möglichen Zustandsübergänge mit deren Ein- und Ausgabenachrichten sowie deren Sender- und Empfängerlisten versehen. Es muss genau ein Startzustand und mindestens ein Endzustand vorhanden sein und es müssen alle Endzustände vom Startzustand aus erreichbar sein. Diese Bedingung resultiert aus der Vorgabe, die Koordination zu Ende führen zu können. Des Weiteren muss der Startzustand ungleich dem Endzustand sein, da ansonsten das Koordinationsprotokoll aus einem Zustand bestehen würde und dieser beim Start der Koordination sofort erreicht wäre, was bedeutet, dass keine Koordination stattfinden würde.

Ausgetauschte Nachrichten

Neben dem Zustandübergangsdiagramm sind zum Betrieb des Koordinations-Agenten folgende Nachrichten von Bedeutung, die zwischen der RPD-Anwendung und dem Koordinations-Agent ausgetauscht werden. Es wird dabei zwischen unterschiedlichen Gruppen von Nachrichten unterschieden, je nachdem in welchem Zustand sich die Koordinationspartner im Zusammenspiel mit dem Koordinations-Agenten befinden. Die Definition der ausgetauschten Nachrichten findet sich im Anhang E.4.

1. Agenteninstanziierung:
    - **Initial-Coordination-Agent**: Mit dieser Nachricht wird ein neuer Koordinations-Agent erzeugt. Diese Nachricht wird vom Master-Agenten empfangen, worauf ein Koordinations-Agent instanziiert wird und ihm diese Nachricht zur weiteren Bearbeitung zugesandt wird.
    - **Initial-Coordination-Agent-Ack**: Die Nachricht ist die Antwort des neuen Koordinations-Agenten an die anfordernde RPD-Anwendung. Sie ist die direkte Quittierung der Nachricht *Initial-Coordination-Agent*.
2. Agentenauffindung:
    - **Request-Coordination-Agent**: Diese Nachricht wird von den RPD-Anwendungen, die an der Koordination beteiligt sind, aber nicht das Koordinationsprotokoll festgelegt haben, an die RPD-Middleware geschickt. Damit wird dem Koordinations-Agenten angezeigt, dass diese RPD-Anwendung an der Koordination teilnimmt.
    - **Request-Coordination-Agent-Ack**: Diese Nachricht wird als Antwort auf die Nachricht *Request-Coordination-Agent* an die RPD-Anwendung übermittelt. Sie dient als Bestätigung für die anfragende RPD-Anwendung, womit dieser angezeigt wird, dass der entsprechende Koordinations-Agent existiert. Dies ist in sofern von Bedeutung, da der Koordinations-Agent erst dann *Request-Coordination-Agent*-Nachrichten empfangen kann, wenn er durch eine RPD-Anwendung mit Hilfe der Nachricht *Initial-Coordination-Agent* instanziiert wurde.
3. Koordinationsprotokoll-Spezifikation sowie Abfrage:
    - **FSM-Definition**: Mit dieser Nachricht wird der endliche Automat in der oben beschriebenen Form durch die RPD-Anwendung definiert und dem Koordinations-Agenten übermittelt.
    - **FSM-Definition-Ack**: Mit dieser Nachricht wird der RPD-Anwendung der Empfang des Zustandübergangsdiagramms nach einer syntaktischen und semantischen Prüfung positiv quittiert. Es ist die direkte Antwort auf die Nachricht *FSM-Definition*.

- **FSM-Definition-Nack**: Mit dieser Nachricht wird der RPD-Anwendung mitgeteilt, dass die syntaktische und semantische Überprüfung des Zustandübergangsdiagramms negativ ausfiel. Es ist die direkte Antwort auf die Nachricht *FSM-Definition*.
- **Request-FSM**: Diese Nachricht dient zur Abfrage des Zustandübergangsdiagramms. Mit Hilfe dieser Nachrichten können Koordinationspartner, die nicht den Automaten definiert haben, diesen zur weiteren internen Bearbeitung und Auswertung vom Koordinations-Agenten erfragen.
- **FSM**: Diese Nachricht enthält das Zustandübergangsdiagramm und ist die Antwort des Koordinations-Agenten auf die Nachricht *Request-FSM*. Auch hier wird auf die oben angegebene Sprachdefinition zurückgegriffen.

4. Austausch der Ein- und Ausgabenachrichten:

   - **Input**: Diese Nachricht erzwingt einen Zustandsübergang. Ist im aktuellen Zustand die Eingabe unzulässig, beispielsweise weil der Sender keine Eingabenachricht mit diesem Text verschicken darf, so erhält der Sender dies als Fehler (*Wrong-Input*) mitgeteilt.
   - **Input-Ack**: Diese Nachricht wird an den Sender einer Eingabenachricht als Empfangsbestätigung verschickt. Das ist wichtig, da der Sender nicht zwingend in der Liste der zu empfangenden Ausgabenachrichten enthalten ist und daher nicht unbedingt eine Quittierung in Form einer *Output*-Nachricht erhält.
   - **Wrong-Input**: Diese Nachricht wird verschickt, wenn die Eingabe im aktuellen Zustand unzulässig ist. Dies ist dann der Fall, wenn entweder kein Zustandsübergang existiert, der denselben Eingabetext besitzt wie die Eingabenachricht oder wenn der Sender nicht in der Liste der erlaubten Sender des Zustandübergangs definiert ist.
   - **Output**: Diese Nachricht wird beim Zustandsübergang erzwungen. Sie enthält die Ausgabenachricht und den neuen Zustandsnamen. Die Nachricht wird nur an die Koordinationspartner verschickt, die in der Liste der Empfänger vermerkt sind.
   - **End-State**: Diese Nachricht wird verschickt, wenn ein Endzustand des Koordinationsprotokolls erreicht wurde.

5. Zustandsabfrage:

   - **Request-State**: Mit dieser Nachricht kann die RPD-Anwendung den aktuellen Zustand beim Koordinations-Agenten abfragen und erhält als Antwort die *Current-State*-Nachricht.
   - **Current-State**: Diese Nachricht übermittelt den angefragten, aktuellen Zustand an die RPD-Anwendung und ist die Antwort auf die *Request-State*-Nachricht.

Beispiel

Im Folgenden wird kurz anhand einer einfachen Koordination zweier RPD-Anwendungen die Funktionsweise und die Verwendung der ausgetauschten Nachrichten erläutert. Das Beispiel zeigt den Abstimmungsprozess (siehe Bild 28), wie er bei der Problemlösung zwischen zwei Partnern auftritt. Ein weiterführendes und ausführliches Beispiel findet sich in Kapitel 8.

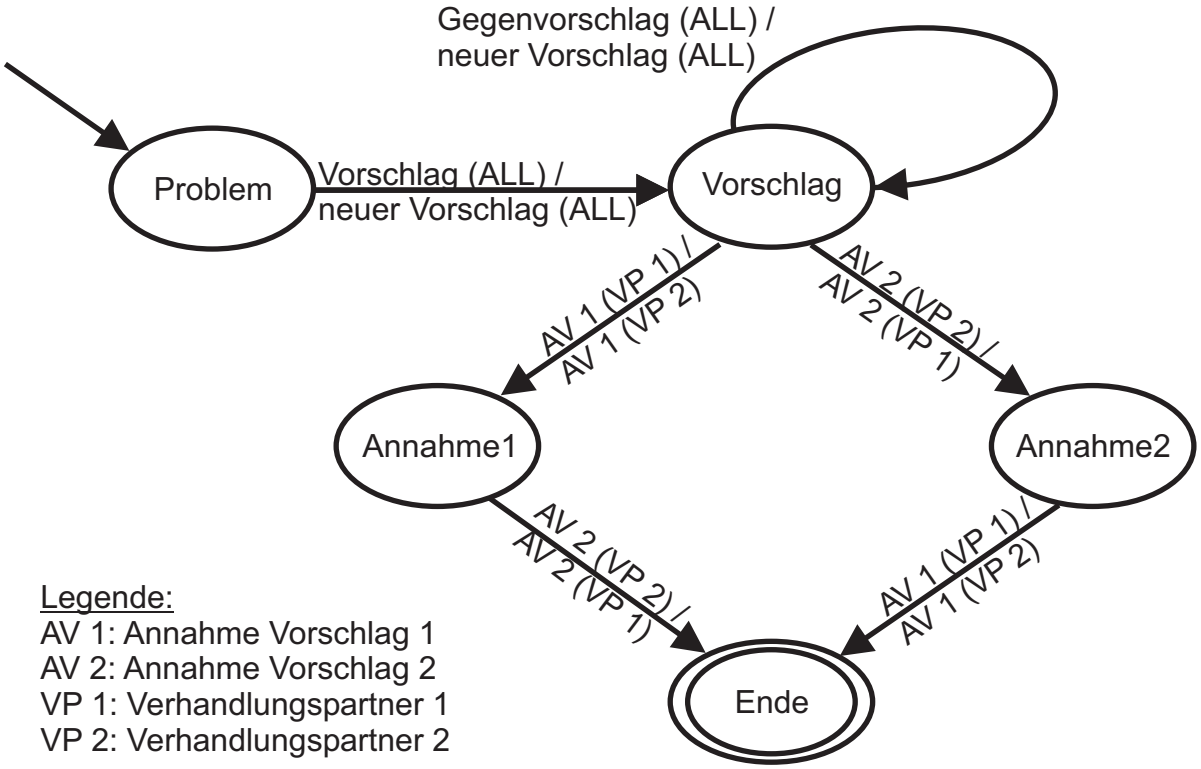


Bild 28: Beispiel eines Koordinationsprotokolls als Zustandübergangsdiagramm

Vom Startzustand *Problem* kann einer der beiden Verhandlungspartner (VP) einen Vorschlag machen. Der Eingang des Vorschlags wird beiden Verhandlungspartnern durch die Ausgabenachricht *neuer Vorschlag* angezeigt und in den Zustand *Vorschlag* gewechselt. In diesem Zustand können beliebig viele Gegenvorschläge durch die beiden Verhandlungspartner vorgenommen werden. Entschließt sich ein Verhandlungspartner den aktuellen Vorschlag bzw. Gegenvorschlag anzunehmen, wird der Abstimmungsprozess angestoßen. Hierzu wird je nach Verhandlungspartner in die Zustände *Annahme1* oder *Annahme2* verzweigt und dem jeweils anderen Verhandlungspartner die Annahme des Vorschlags (AV) durch eine entsprechende Ausgabenachricht angezeigt. Die Zustände *Annahme1* und *Annahme2* können nur noch in Richtung Endzustand, durch die Annahme des Vorschlags durch den jeweils anderen Verhandlungspartner, verlassen werden. Dadurch werden beide Verhandlungspartner zur Akzeptanz des Vorschlags gezwungen bzw. beide Verhandlungspartner haben sich auf diesem Wege abgestimmt.

In Bild 29 und Bild 30 ist ein Sequenzdiagramm dargestellt, an dem exemplarisch ein möglicher Koordinationsablauf erklärt wird. Die Arbeit des Koordinations-Agenten teilt sich in drei Abschnitte auf.

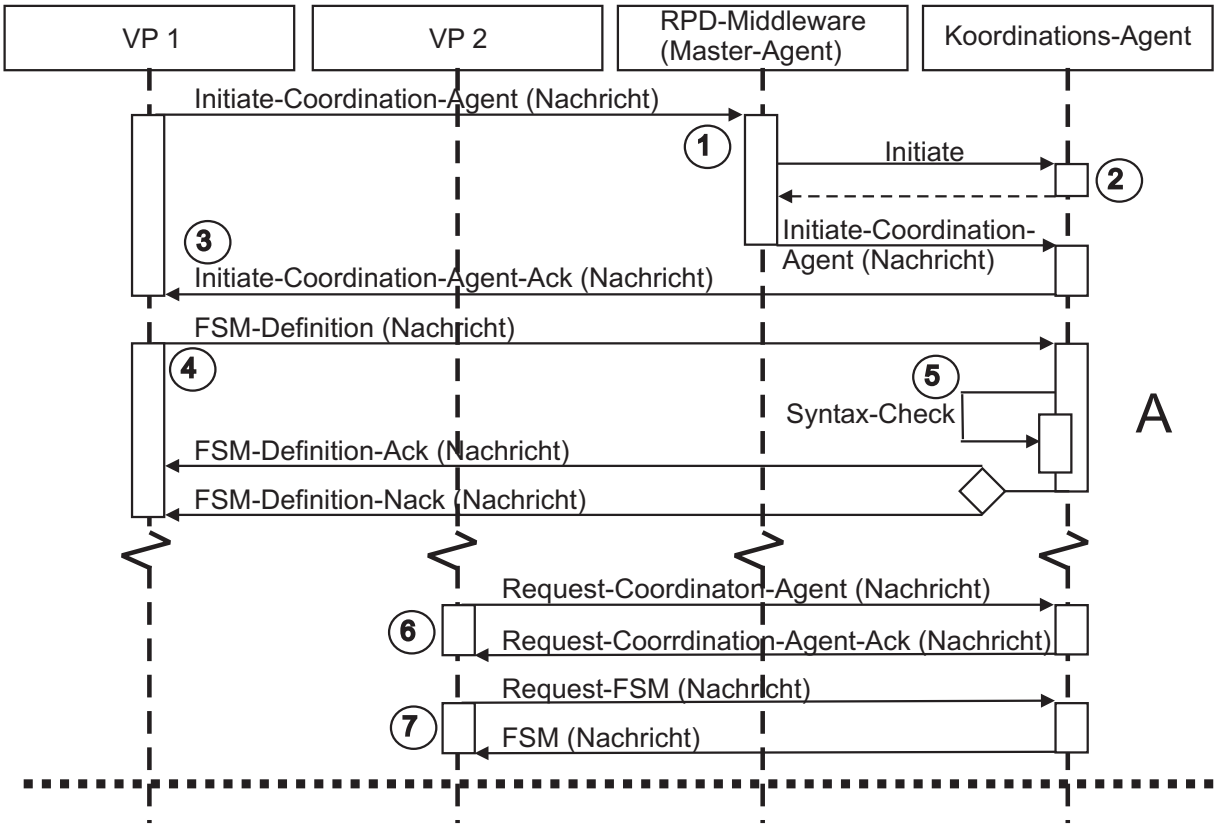


Bild 29: Sequenzdiagramm eines typischen Kommunikationsablaufs für den Koordinations-Agenten (1)

Im Abschnitt A (siehe Bild 29) wird der Agent mit Hilfe der Nachricht *Initiate-Coordination-Agent*, die an die RPD-Middleware geschickt wird (1), instanziiert (2). Der neu instanziierte Koordinations-Agent erhält die Anforderung vom Master-Agenten und teilt seine Empfangsbereitschaft dem *VP 1* mit (3). Das Zustandübergangsdiagramm wird von dem *VP 1* generiert und dem Koordinations-Agenten übermittelt (4), der dieses einer Syntax- und Semantiküberprüfung unterzieht (5) und bei positiver Begutachtung dieses der RPD-Anwendung mit Hilfe der *FSM-Definition-Ack*-Nachricht anzeigt. Damit wurde dem Koordinations-Agenten seine Aufgabe mitgeteilt. Mit Hilfe der nächsten zwei Nachrichtenpaare meldet sich der *VP 2* beim Koordinations-Agenten (*Request-Coordination-Agent* und *Request-Coordination-Agent-Ack*) an (6) und fragt das Koordinationsprotokoll in Form des Zustandübergangsdiagramms ab (*Request-FSM* und *FSM*) (7).

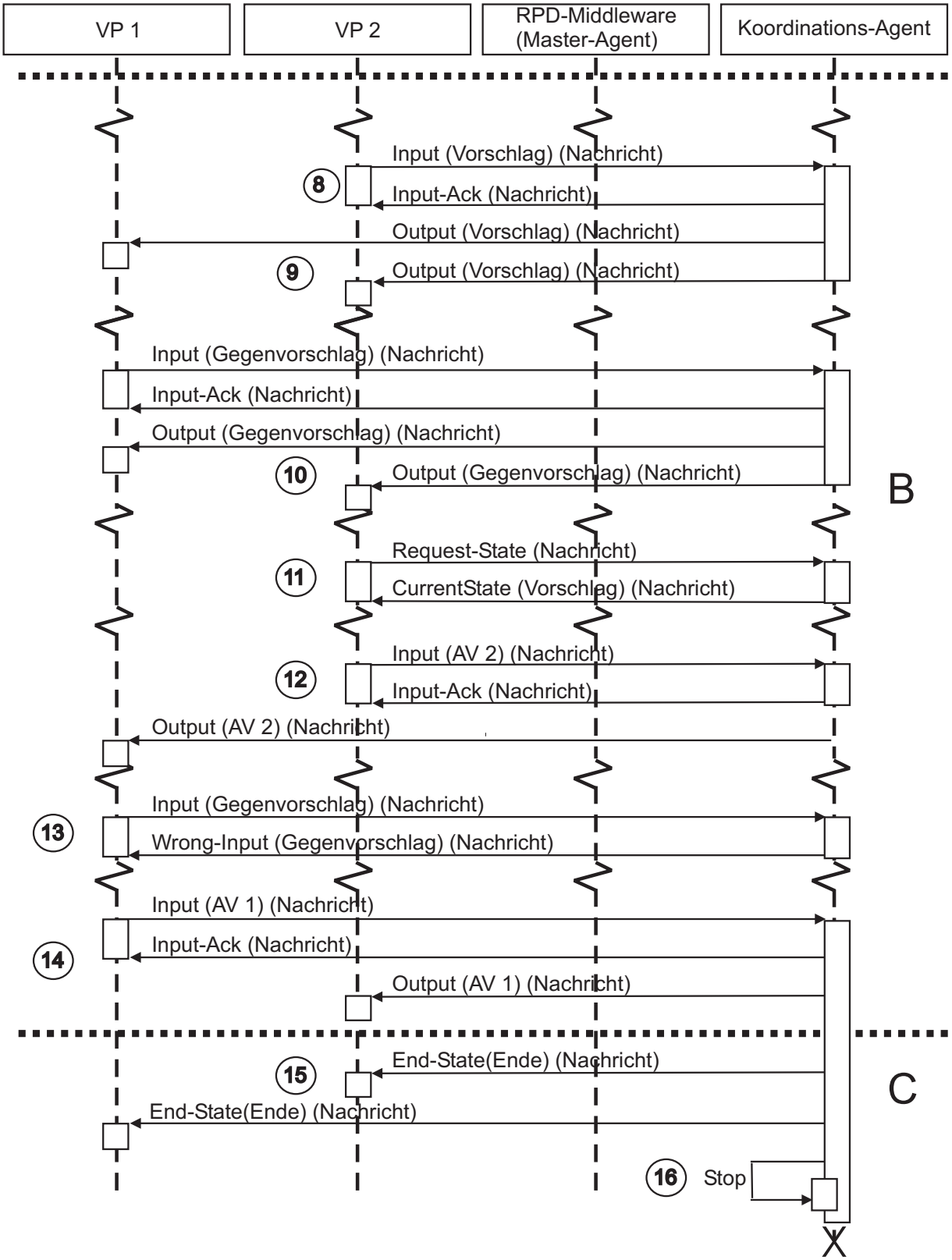


Bild 30: Sequenzdiagramm eines typischen Kommunikationsablaufs für den Koordinations-Agenten (2)

Im zweiten Abschnitt B (siehe Bild 30) findet die Abarbeitung des Koordinationsprotokolls statt. Hier werden die Anfragen empfangen, die Ausgaben produziert und die Zustände gewechselt. So macht in diesem beispielhaften Durchlauf des Koordinationsprotokolls der *VP 2* den ersten Vorschlag (*Input (Vorschlag)*), der ihm vom Koordinations-Agenten quittiert (*Input-Ack*) (8) und an alle beiden Verhandlungspartnern verschickt wird (*Output (Vorschlag)* (9)), worauf *VP 1* einen Gegenvorschlag unternimmt (10), angedeutet durch die nächsten vier Nachrichten *Input (Gegenvorschlag)*, *Input-Ack*, *Output (Gegenvorschlag)*, *Output (Gegenvorschlag)*. Zwischendurch wird durch den *VP 2* mit Hilfe der Nachricht *Request-State* der aktuelle Zustand abgefragt, der Koordinations-Agent beantwortet durch die Nachricht *Current-State (Vorschlag)* (11) diese Anfrage. Nach Abfrage des aktuellen Zustands tritt der *VP 2* zuerst in die Annahme des Vorschlags ein (*Input (AV 2)*, *Input-Ack*, *Output (AV 2)*) (12). Hierbei ist zu beachten, dass nach Definition des Koordinationsprotokolls die Annahme des Vorschlags durch den *VP 2* nur dem *VP 1* angezeigt wird. Anstatt der Annahme des Vorschlags zuzustimmen, unterbreitet *VP 1* einen weiteren Gegenvorschlag (*Input (Gegenvorschlag)*) (13). Dieser wird durch den Koordinations-Agenten als falsche Eingabe abgewiesen (*Wrong-Input (Gegenvorschlag)*), da diese Eingabe im aktuellen Zustand (*Annahme2*) nicht erlaubt ist. Hierauf stimmt *VP 1* ebenfalls dem Vorschlag zu (*Input (AV 1)*, *Input-Ack*, *Output (AV 1)*) (14).

In Abschnitt C (siehe Bild 30) wird der Koordinations-Agent nach erfolgreicher Koordination, gekennzeichnet durch das Erreichen eines Endzustands (End-State(Ende), End-State(Ende)) (15), beendet (16).

Sind während der Koordinationsphase dem Koordinations-Agenten nicht alle Koordinationspartner bekannt, so werden die Ausgabenachrichten für diese RPD-Anwendungen verworfen, um zu verhindern, dass inhaltlich überholte Nachrichten versendet werden.

7 Realisierung der RPD-Middleware

7.1 Architektur und prototypische Realisierung

In Bild 31 ist die vollständige Architektur der RPD-Middleware abgebildet. Sie gliedert sich dabei in die Bereiche Agentensystem, RPD-Anwendung, ASN, Multicast-Umgebung, den Schnittstellen zwischen den Bereichen sowie deren Bestandteilen (siehe Kapitel 5).

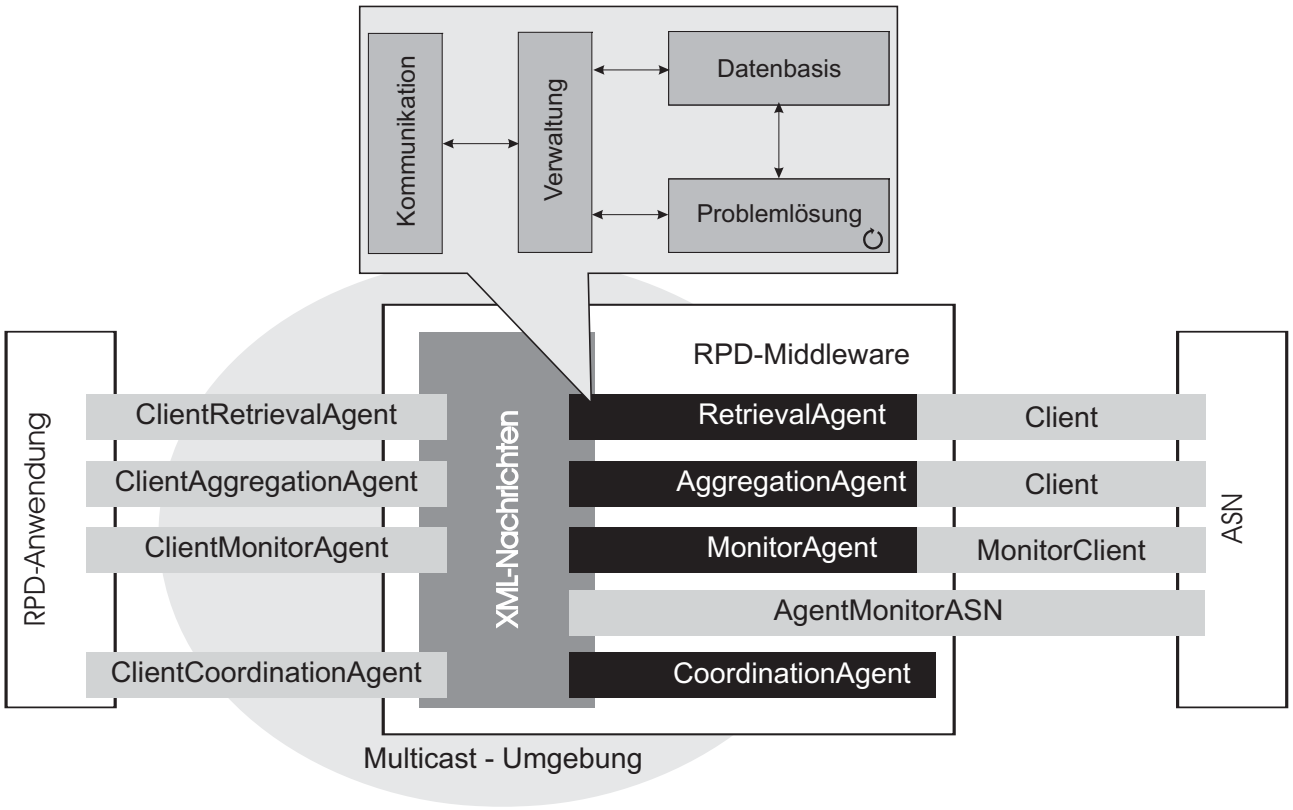


Bild 31: Architektur der RPD-Middleware

Agentensystem

Das Agentensystem beinhaltet die vier RPD-Agenten, wie sie im Kapitel 6 beschrieben wurden. Die XML-Nachrichten, die für die Kommunikation zwischen RPD-Anwendung und Agent, sowie zwischen den Agenten benötigt werden, sind je Agententyp gruppiert, sodass im Falle einer Erweiterung durch weitere Agenten bzw. auf Seiten der RPD-Anwendung der komplette Nachrichtensatz und damit das Kommunikationsprotokoll einfach eingebunden werden kann.

Des Weiteren gehört zum Agentensystem das Kommunikationssystem mit einem einheitlichen Adressierungsschema für die Kommunikationspartner, dem UAI (siehe Abschnitt 5.2). Dabei ist zu beachten, dass als eindeutige ID des UAI durch die RPD-Clientschnittstelle bzw. durch den Master-Agenten die Systemzeit des instan-

ziierenden Systems verwendet wird. Das instanziierende System ist entweder der Master-Agent oder die RPD-Anwendung. Für die Instanziierung von neuen Agenten ist die Masterfunktionalität innerhalb der Agenten verantwortlich. Wie im Abschnitt 5.4 beschrieben ist zur Laufzeit des Agentensystems immer nur die Masterfunktionalität eines Agenten aktiv, der daher Master-Agent genannt wird. Der Master-Agent entscheidet anhand der UAI, ob ein Agent instanziiert werden muss, dies geschieht dann, wenn innerhalb des Agentensystems die UAI unbekannt ist. Dazu verfügt der Master-Agent über eine Liste aller sich im System befindlichen Agenten. Ist die Instanziierung eines Agenten erforderlich, dann wird der Agenten-Typ anhand des Feldes *AgentType* des UAI entschieden.

Interner Aufbau der RPD-Agenten

Die vier RPD-Agenten bestehen aus den vier Modulen Kommunikation, Verwaltung, Problemlösung und Datenbasis (siehe Abschnitt 5.5).

- **Kommunikation**: Das Kommunikationsmodul überprüft, ob die Nachrichten, die an die Multicast-Adresse geschickt wurden, für diesen Agenten bestimmt sind und gibt sie dann an das Verwaltungsmodul weiter. Dabei treten zwei Klassen von Nachrichten auf. Die erste Klasse repräsentiert die Nachrichten, die für die Aufrechterhaltung der Masterfunktionalität benötigt werden. Diese Nachrichten werden von allen Agenten bearbeitet. Bei der zweiten Klasse der Nachrichten handelt es sich um die Agententyp-spezifischen Nachrichten. Hierbei wird durch das Kommunikationsmodul zuerst überprüft, ob die Empfänger-UAI mit der eigenen UAI übereinstimmt, d. h. ob der richtige Agent angesprochen wird. Wenn dies der Fall ist, wird in einem zweiten Schritt geprüft, ob der Agent in der Lage ist die Nachricht weiterzuverarbeiten. Dies geschieht durch die Überprüfung, ob die eingegangene Nachricht im aktuellen Zustand des Agenten weiter verarbeitet werden kann.
- **Verwaltung**: Das Verwaltungsmodul konvertiert die Nachricht in eine interne Repräsentationsform, die je nach Agententyp unterschiedlich ist. Das Ergebnis der Konvertierung wird in der lokalen Datenbasis abgelegt und je nach Nachricht der entsprechende Teil des Problemlösungsmoduls gestartet bzw. wird das Ergebnis des Problemlösungsmoduls zurück in eine Nachricht konvertiert und über das Kommunikationsmodul an den Empfänger verschickt.
- **Problemlösung**: Das Problemlösungsmodul bearbeitet aufgrund der Daten aus der Datenbasis seine Aufgabe, schreibt die Ergebnisse zurück in die Datenbasis und teilt dies der Verwaltung zur weiteren Bearbeitung mit, z. B. wertet das Problemlösungsmodul eines Monitor-Agenten die Überwachungsbedingung aus.
- **Datenbasis**: Die Datenbasis besteht aus einem Satz von Objekten und beinhaltet die lokalen Daten des Agenten, die abhängig vom Typ des Agenten sind. Im Fall eines Koordinations-Agenten ist dies die Repräsentation des endlichen Automaten.

Client-Agent

Für jeden dieser vier RPD-Agenten gibt es einen Client-Agenten (siehe Bild 31). Dabei handelt es sich um einen Schnittstellen-Agenten für die RPD-Anwendung, der die Kommunikation mit dem Agenten vereinfacht. Die Client-Agenten besitzen daher nur ein stark vereinfachtes Verwaltungsmodul, das lediglich die Befehle der RPD-Anwendung entgegennimmt und diese für die Kommunikation an das Kommunikationsmodul weitergibt. Das Kommunikationsmodul entspricht dem Kommunikationsmodul eines vollwertigen Agenten mit Ausnahme der Masterfunktionalität. Der Client-Agent wird von der RPD-Anwendung gestartet und befindet sich damit auf demselben Rechnersystem wie die RPD-Anwendung. Des Weiteren ist der Client-Agent in Java implementiert und kann, bedingt durch die direkte Interaktion mit der RPD-Anwendung, nur in Java-implementierten RPD-Anwendungen genutzt werden.

RPD-Anwendung

Ohne die RPD-Middleware ist die RPD-Anwendung gezwungen die angebotene Schnittstelle des ASN zu nutzen. Diese ist, wie das gesamte ASN, in Java implementiert, wodurch die RPD-Anwendung ebenfalls in Java implementiert sein muss oder eine Umsetzung durch die RPD-Anwendung erfolgen muss.

Die RPD-Middleware bietet neben den funktionalen Erweiterungen gegenüber dem ASN zwei Zugänge für die RPD-Anwendungen an. Die erste Schnittstelle ist ähnlich der ASN-Schnittstelle in Java implementiert. Es handelt sich hierbei um eine Klasse und ein Interface je Agententyp. Die Klasse repräsentiert den Client-Agenten und bietet für jeden Kommunikationsschritt eine Methode an, die dann die Parameter in die entsprechende XML-basierte Nachricht umwandelt und über sein Kommunikationsmodul dem RPD-Agenten zustellt. Das Interface hingegen präsentiert die Antworten der Agenten der RPD-Anwendung. Hierbei handelt es sich um Methoden, die durch die RPD-Anwendung ausprogrammiert werden müssen und durch den Client-Agent beim Empfang durch die entsprechende Nachricht aufgerufen werden. Dadurch wird zum einen eine enge Kopplung der RPD-Anwendung mit der RPD-Middleware erreicht und zum anderen muss die RPD-Anwendung nicht zyklisch den Agenten auf Ergebnisse abfragen oder Netzwerkverbindungen prüfen.

Für die RPD-Anwendungen, die nicht in Java implementiert sind oder den Client-Agent nicht nutzen können, besteht die Möglichkeit die XML-basierte Nachrichtenschnittstelle direkt anzuwenden. Das Protokoll, also die Abfolge der Nachrichten (siehe Kapitel 6) und die Nachrichtenformate (siehe Anhang E) sind offen gelegt und unabhängig von der Implementierungssprache der RPD-Anwendung. Die RPD-Anwendung muss dann allerdings das Kommunikationsmodul des Client-Agenten nachbilden um die selbst erzeugten XML-Nachrichten über die Multicast-Umgebung an die Agenten zu verschicken. Ebenso müssen die empfangenen Nachrichten in eine interne Repräsentationsform umgewandelt werden. Auf diese

Art und Weise kann jede beliebige RPD-Anwendung mit geringem Aufwand die RPD-Middleware nutzen.

ASN

Das ASN ist unabhängig von der RPD-Middleware und wurde weitgehend unverändert belassen. Die einzige Veränderung, die vorgenommen wurde ist durch den Monitor-Agent impliziert. Es wurden im ASN die Informationselemente wie *Net*, *Concept*, *Attribute*, *Relation* so erweitert, dass sich ein Monitor-Agent an den Informationselementen registrieren kann. Werden die Informationselemente verändert, teilt das ASN dies den registrierten Monitor-Agenten mit. Diese Vorgehensweise vermeidet das aktive Überwachen des ASN durch den Monitor-Agenten. Das ASN setzt für die Benachrichtigung des Monitor-Agenten ebenfalls einen Client-Agenten ein. Die Funktionsweise der Methoden der Informationselemente des ASN ist mit oder ohne registrierte Monitor-Agenten unverändert geblieben.

Multicast-Umgebung

Die Multicast-Umgebung implementiert das Nachrichtensystem. Dabei erfolgt die Kommunikation, ähnlich dem OSI-Sieben-Schichtenmodell [Day 83], durch vier Kommunikationsebenen (siehe Bild 32).

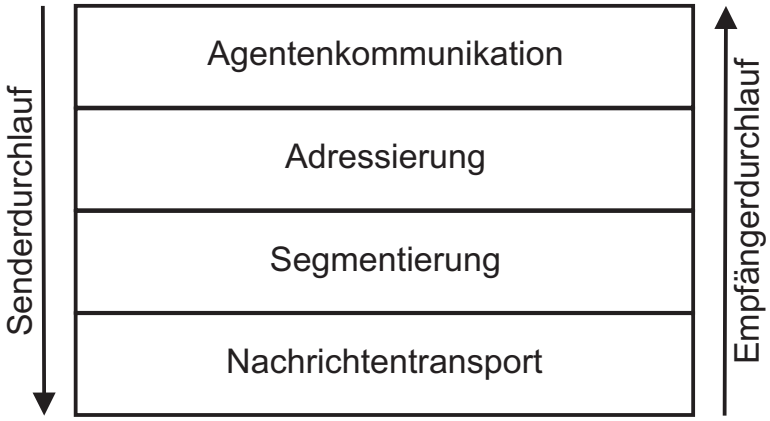


Bild 32: Schichtenmodell der Kommunikationsumgebung

Die unterste Schicht, die Nachrichtentransportschicht, versendet die Nachrichten über ein Multicast-Protokoll, in diesem Fall LRMP von Inria [Lia 00] (siehe Abschnitt 7.2). Aufgrund der Größenbeschränkung von IP-Paketen über Multicast-Verbindungen, ist eine Segmentierung auf Seiten des Senders und ein Zusammenfügen auf Seiten des Empfängers notwendig (Segmentierungsschicht). Innerhalb der Adressierungsschicht wird mit Hilfe der Sender- bzw. Empfänger-UAI die Zustellung der Nachricht gesteuert. Die vierte und oberste Schicht ist der Agentenkommunikation vorbehalten. Über diese Schicht werden die spezifischen Nachrichten je Agententyp ausgetauscht.

Diese vier Schichten, werden wie im OSI-Sieben-Schichtenmodell auf Seiten des Senders von oben nach unten und auf Seiten des Empfängers von unten nach oben durchlaufen.

Schnittstellen

Die RPD-Middleware besitzt aufgrund ihrer Einbettung in die Architektur zwei Schnittstellen. Zum einen zwischen den RPD-Anwendungen und der RPD-Middleware und zum anderen zwischen der RPD-Middleware und dem ASN (siehe Bild 31).

Die Schnittstelle zwischen RPD-Anwendung und RPD-Middleware basiert auf wohl definierten Nachrichten im XML-Format und dem Kommunikationsprotokoll, also der Abfolge der Nachrichten. Des Weiteren wurde, der Implementierungssprache der RPD-Middleware folgend, eine vereinfachte Java-basierte Schnittstelle für die RPD-Anwendungen realisiert.

Bei der zweiten Schnittstelle zwischen RPD-Middleware und ASN handelt es sich um eine Java-basierte Schnittstelle, die einen schnelleren und direkten Zugriff auf das ASN erlaubt. Eine Ausnahme bilden hierbei lediglich Meldungen des ASN auf hinterlegte Anfragen des Monitor-Agenten, hier musste, damit das ASN proaktiv reagieren kann, ein Client-Agent eingesetzt werden.

7.2 Infrastruktur

Dieser Abschnitt beschäftigt sich mit der Hardware- und Software-Umgebung auf deren Basis die RPD-Middleware entwickelt wurde. Dabei wurden auch die Randbedingungen, die durch die bereits existierenden RPD-Anwendungen und das ASN vorgegeben sind, berücksichtigt.

Hardwareumgebung

Die RPD-Middleware wurde plattformunabhängig konzipiert und realisiert. Hierzu wurde das System auf PC-Basis sowohl unter Microsoft Windows in der jeweils aktuellen Version (Windows 2000 / Windows 2003 Server / XP) und unter Linux (Suse 7.3 – 9.2) entwickelt. Aufgrund der hohen Anzahl von Prozessen, die durch die Agenten begründet sind und parallel bearbeitet werden können und sollten, wurde der Betrieb der RPD-Middleware auf Mehrprozessorrechnern erprobt. Dies waren eine vier Prozessormaschine von Sun mit dem Betriebssystem Solaris 9 sowie Server aus dem Hause IBM und HP, wahlweise mit den Betriebssystemen Windows 2003 Server bzw. Suse Linux 9.2.

Softwareumgebung

Die RPD-Middleware wurde, um die enge und effiziente Kopplung mit dem ASN zu erreichen, in Java implementiert. Dabei wurden so wenig wie möglich spezielle Zusatzpakete verwendet, um einen plattformunabhängigen Einsatz des Systems zu erreichen. Die einzige Ausnahme macht hierbei die Multicast-Umgebung, die dem Softwarepaket beigefügt wurde. Die einzelnen Softwarekomponenten RPD-Anwendungen, RPD-Middleware und ASN müssen dabei nicht auf einem Rechnersystem betrieben werden. Die RPD-Middleware und das ASN können auf getrennten Rechnersystemen ausgeführt werden. Allerdings ist darauf zu achten, dass die Rechnersysteme für den Betrieb der RPD-Middleware und des ASN benachbart und über ein schnelles Netzwerk verbunden sind, um hohe Performanceverluste zu vermeiden. Die RPD-Anwendungen sind in der Regel nicht serverbasiert und werden auf den entsprechenden Systemen, für deren Einsatz sie konzipiert wurden, betrieben.

Netzwerkinfrastruktur

Als Netzwerkinfrastruktur wurde Fast- und Gigabit-Ethernet eingesetzt. Die Multicast-Umgebung erzwingt an den Routinggrenzen zusätzliche Konfigurationsaufwände im Netzwerkbereich, da Multicast-Pakete speziellen Routing-Mechanismen unterliegen.

Light-weight Reliable Multicast Protocol (LRMP)

Im Standard Multicast-Verkehr wird weder die Zustellung noch die Reihenfolge der Zustellung der Datenpakete garantiert. Dies bedeutet, dass entweder innerhalb des Kommunikationsmoduls der Agenten Mechanismen realisiert werden müssen, um die Zustellung und die Reihenfolge bei segmentierten Nachrichten sicher zu stellen oder dass das Multicast-Protokoll um die entsprechenden Mechanismen erweitert werden muss. Die zweite Variante wurde vor dem Hintergrund gewählt, das Kommunikationsmodul möglichst einfach zu halten und die Zuverlässigkeit der Kommunikation bereits auf den unteren Kommunikationsebenen sicherzustellen, wodurch die Performance des Nachrichtenaustauschs nicht in dem Maße beeinträchtig wird, wie es bei einer Abwicklung zwischen den Agenten die Folge gewesen wäre. Es wurde hierfür ein frei verfügbares Softwarepaket, das Light-weight Reliable Multicast Protocol (LRMP) von Inria [Lia 00], eingesetzt.

8 Anwendungsbeispiel: Entwicklung einer Luftdüse

In diesem Anwendungsbeispiel wird sowohl die Funktionsweise als auch der Nutzen der RPD-Middleware anhand der Arbeitsweise von vier verschiedenen RPD-Nutzern bei ihrer Zusammenarbeit verdeutlicht. Dabei handelt es sich um einen Projektplaner, einen Qualitätsmanager, einen Kostenrechner und einen Prototypenbauer. Gegenstand des Anwendungsbeispiels ist die Entwicklung eines Prototyps für eine Luftdüse (siehe Bild 33), die sich im Fahrzeuginnenraum befindet. Um das Beispiel überschaubar zu halten wird ein beliebiger Arbeitstag der vier RPD-Nutzer betrachtet und die Vorgehensweise ohne den Einsatz der RPD-Middleware und die Verbesserungen, die durch den Einsatz der RPD-Middleware entstehen, gegenübergestellt. Die einzelnen Kommunikationsschritte und der damit verbundene Nachrichtenaustausch zwischen RPD-Anwendungen und der RPD-Middleware, sowie die entsprechenden Screenshots der RPD-Middleware sind im Anhang F detaillierter aufbereitet.

Bild 33: Anwendungsbeispiel: physischer Prototyp der Luftdüse

8.1 Rollen-unabhängige Vorgehensweise

Die Rollen-unabhängige Vorgehensweise beinhaltet alle Schritte, die ein RPD-Nutzer zu Beginn seiner Tätigkeit erledigen muss. Hierzu zählen (siehe Bild 34) unter anderem das Anmelden an dem RPD-Portal (1), das die Aufgabe hat, neben einem einheitlichen Zugriff, die rollenspezifischen Voreinstellungen und die Auswahl der korrekten RPD-Anwendungen vorzunehmen. Hierbei werden Loginname und Passwort mit den gespeicherten Informationen des ASN abgeglichen. Im zweiten Schritt (2) wird die Teaminformation, wie alle Mitglieder des Teams, der Teamleiter inklusive deren Status aus dem ASN beschafft. Im Schritt (3) wird das Rollenprofil aus dem ASN ausgelesen und das Portal dementsprechend konfiguriert.

Die im Schritt (2) und (3) beschafften Informationen aus dem ASN müssen ohne den Einsatz der RPD-Middleware direkt Anfrage für Anfrage ausgelesen werden. So werden für Schritt (2) beispielsweise bei einer Teamstärke von zehn Mitgliedern 32 Anfragen (eine für das Team, jeweils drei für die Attribute Vorname, Nachname, Status der Mitglieder, eine für den Teamleiter) an das ASN gestellt. Wird hingegen der Retrieval-Agent aus der RPD-Middleware eingesetzt, so ist nur eine kombinierte Suchanfrage (siehe Anhang F.1 und F.2) an den Aggregations-Agenten notwendig. Im Schritt (3) ist die Sachlage dieselbe. Auch hier muss eine Vielzahl von Anfragen gestellt werden um das Rollenprofil aus dem ASN auszulesen, während dies durch die Kombination Retrieval-/Aggregations-Agent vereinfacht werden kann.

Der Schritt (4) symbolisiert die rollenspezifische Weiterführung durch die entsprechenden RPD-Nutzer, die in den nächsten vier Abschnitten näher betrachtet werden.

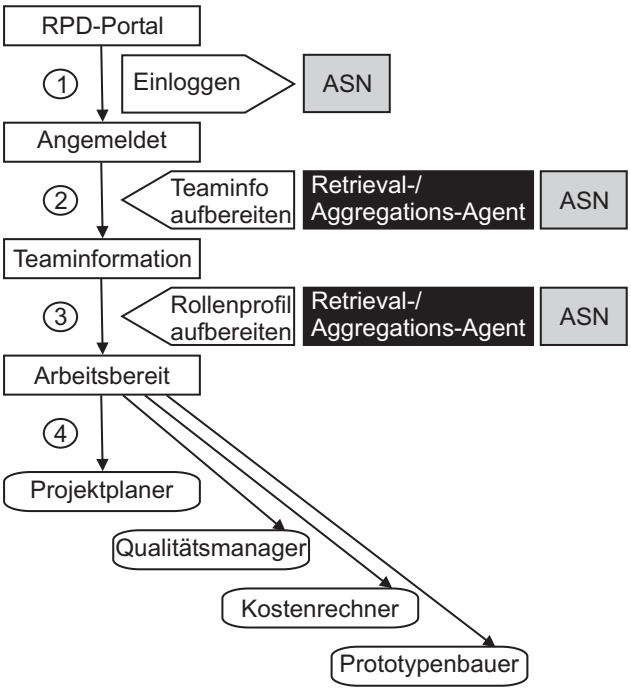


Bild 34: Anwendungsbeispiel: Anmeldung

8.2 Projektplaner

Der Projektplaner hat in diesem Beispiel die Aufgabe die Einhaltung des Projektplans für die Erstellung des physischen Prototyps für eine Fahrzeug-Luftdüse zu überwachen [Ley 00], [Ley 01], [Die 00]. Wird der Projektplan verletzt. ist eine Koordination aller an der Entwicklung der Luftdüse Beteiligten notwendig.

Ohne den Einsatz der RPD-Middleware wird im Schritt (1) der aktuelle Projektplan aus dem ASN abgerufen (siehe Bild 35). Der Status der Aktivität *Simulation_Prototyp_Luftdüse* wird durch die Schritte (2), (4), (5) und (8) zyklisch überwacht. Ist die Fertigstellung der Simulation termingerecht erfolgt, so wird die Über-

wachung durch den Schritt (3) beendet. Ist der Termin überschritten (Schritt (6)) wird eine Abstimmung aller Beteiligten notwendig. Die daraus resultierenden Anpassungen, wie die Korrektur des Projektplans werden im Schritt (7) vorgenommen bevor die Überwachung von neuem aufgenommen wird.

Die Vorgehensweise ohne Einsatz der RPD-Middleware zeigt, neben der umständlichen Abfrage des Projektplans, zwei weitere Schwierigkeiten im Umgang mit dem ASN auf. Zum einen muss das ASN zyklisch auf Veränderungen (Schritt (2), (4), (5), (8)) überwacht werden und zum anderen ist keine Vorgehensweise definiert, wie bei Nichteinhaltung des Projektplans zu verfahren ist.

Durch den Einsatz der RPD-Middleware ist es möglich, mit Hilfe des Monitor-Agenten das ASN aktiv zu überwachen, die Schleife über Schritt (8) entfällt und muss lediglich durch eine parallele Zeitabfrage ersetzt werden. Die Abfrage für den Monitor-Agenten würde beispielsweise *String(Value(Luftdüse, Simulation_Prototyp_Luftdüse, Bearbeitungsstatus)) = String(fertig)* lauten (siehe Anhang F.3). Zusätzlich ändert sich der Informationsfluss im Schritt (2), da der Monitor-Agent aktiv die Veränderungen dem Projektplanungswerkzeug liefert. Dadurch wird das Projektplanungswerkzeug sowohl in seinem logischen Aufbau als auch in der Ausführung stark entlastet.

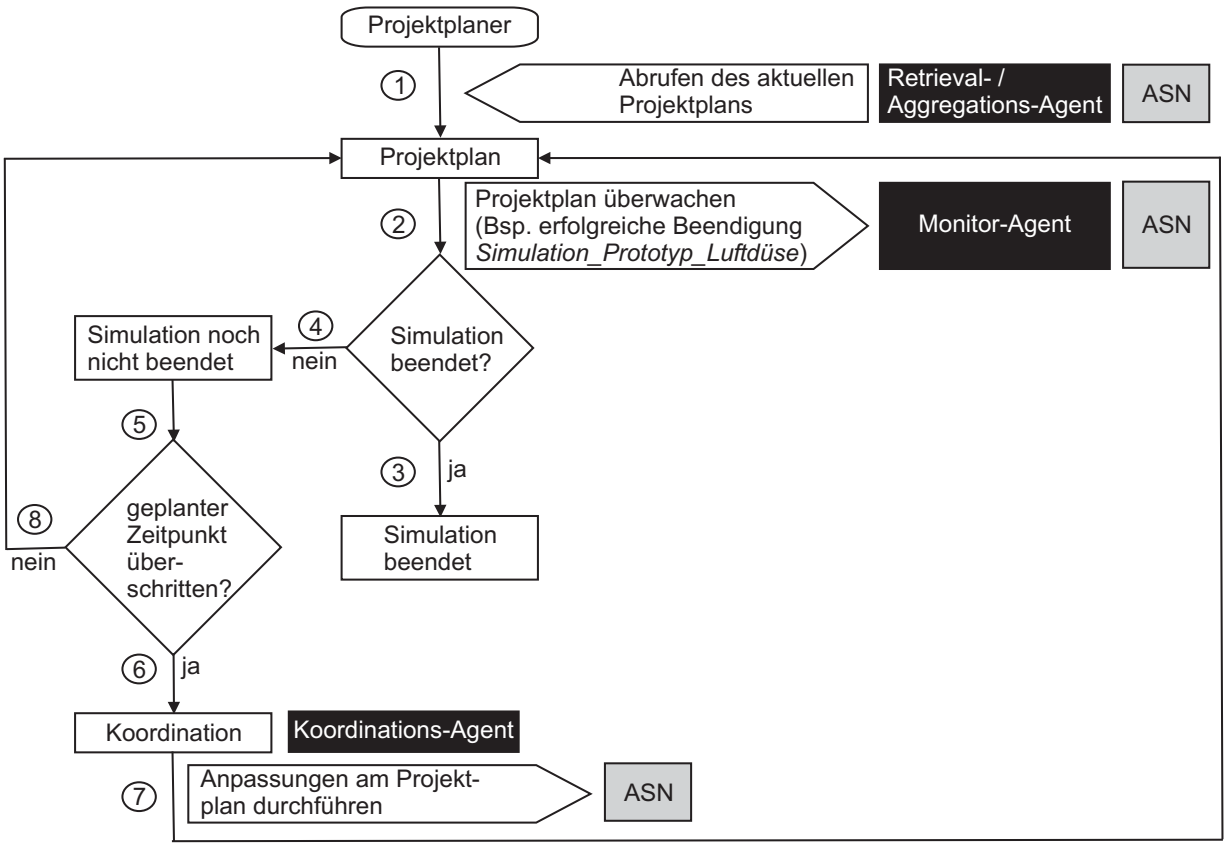


Bild 35: Anwendungsbeispiel: Projektplaner

Der Abstimmungsprozess, der bei Nichteinhaltung des Projektplans notwendig wird, kann durch den Koordinations-Agenten erstmalig unterstützt werden. Hierbei ist durch die Koordinationspartner, hier Projektplaner und Prototypenbauer, ein Koordinationsprotokoll zu definieren, das festlegt, wie in so einem Fall zu verfahren ist (siehe Abschnitt 8.6). Dadurch entfällt langwieriges Emailschreiben, Telefonieren oder Besprechungen, wobei die letzten beiden Koordinationsinstrumente eine ausschließlich gleichzeitige Anwesenheit aller Beteiligten voraussetzen, die bisweilen nur schwer zu erreichen ist.

8.3 Qualitätsmanager

Ähnlich dem Projektplaner hat der Qualitätsmanager Sorge zu tragen, dass ein bestimmter Qualitätsstandard des Prototyps der Luftdüse eingehalten wird.

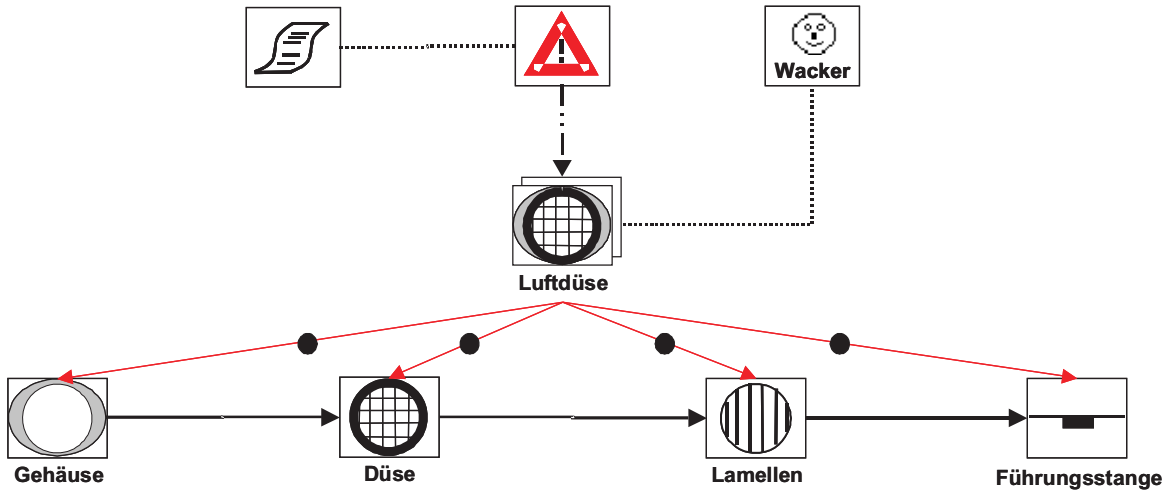


Bild 36: Anwendungsbeispiel: Aktives Semantisches Zuverlässigkeits-Informationsnetz (ASZI) [Ber 04]

Ohne Einsatz der RPD-Middleware sieht der Prozess dafür wie in Bild 37 dargestellt aus. Zuerst (Schritt (1)) werden die Eigenschaften des physischen Prototyps aus dem ASN ausgelesen. Im Schritt (2) und (3) wird das ASN solange zyklisch auf Veränderungen der Prototypeneigenschaften überwacht bis diese eingetreten sind (Schritt (4)). Anschließend werden die veränderten Eigenschaften aus dem ASN ausgelesen (Schritt (5)) und dann wird aufgrund der Bestandteile des Prototyps das Aktive Semantische Zuverlässigkeits-Informationsnetz (ASZI) (siehe Bild 36) gebildet und darüber die Zuverlässigkeit und die Qualität der Luftdüse bestimmt. Liegt die Qualität im Rahmen der Vorgaben, so wird die zyklische Überwachung des ASN fortgeführt (Schritt (7)). Ist der Qualitätsstandard verletzt und damit ein Qualitätsproblem entstanden, wird eine Koordination zwischen dem Qualitätsmanager und dem Prototypenbauer notwendig (Schritt (8)). Werden dabei Kosten verändert bzw. gerät der Projektplan durch die veränderte Situation in Gefahr, so muss der Kostenrechner und der Projektplaner zur Koordination hinzugezogen werden. Die-

se Koordination findet ohne den Einsatz der RPD-Middleware telefonisch oder als Besprechung, etc. statt. Die durch die Koordination erzwungenen Veränderungen werden im Schritt (9) eingepflegt.

Durch den Einsatz der RPD-Middleware kann die Arbeitsweise des Qualitätsmanagers stark vereinfacht werden. Zum einen wird im Schritt (1) und (5) durch den Einsatz des Retrieval- / Aggregations-Agenten die Abfrage der benötigten Daten effizienter gestaltet. Des Weiteren entfällt durch den Einsatz des Monitor-Agenten die zyklische Überwachung des ASN und der Koordinationsprozess zwischen den RPD-Beteiligten kann durch den Koordinations-Agenten unterstützt werden. Hierzu ist im Abschnitt 8.6 exemplarisch ein mögliches Koordinationsprotokoll ausgeführt.

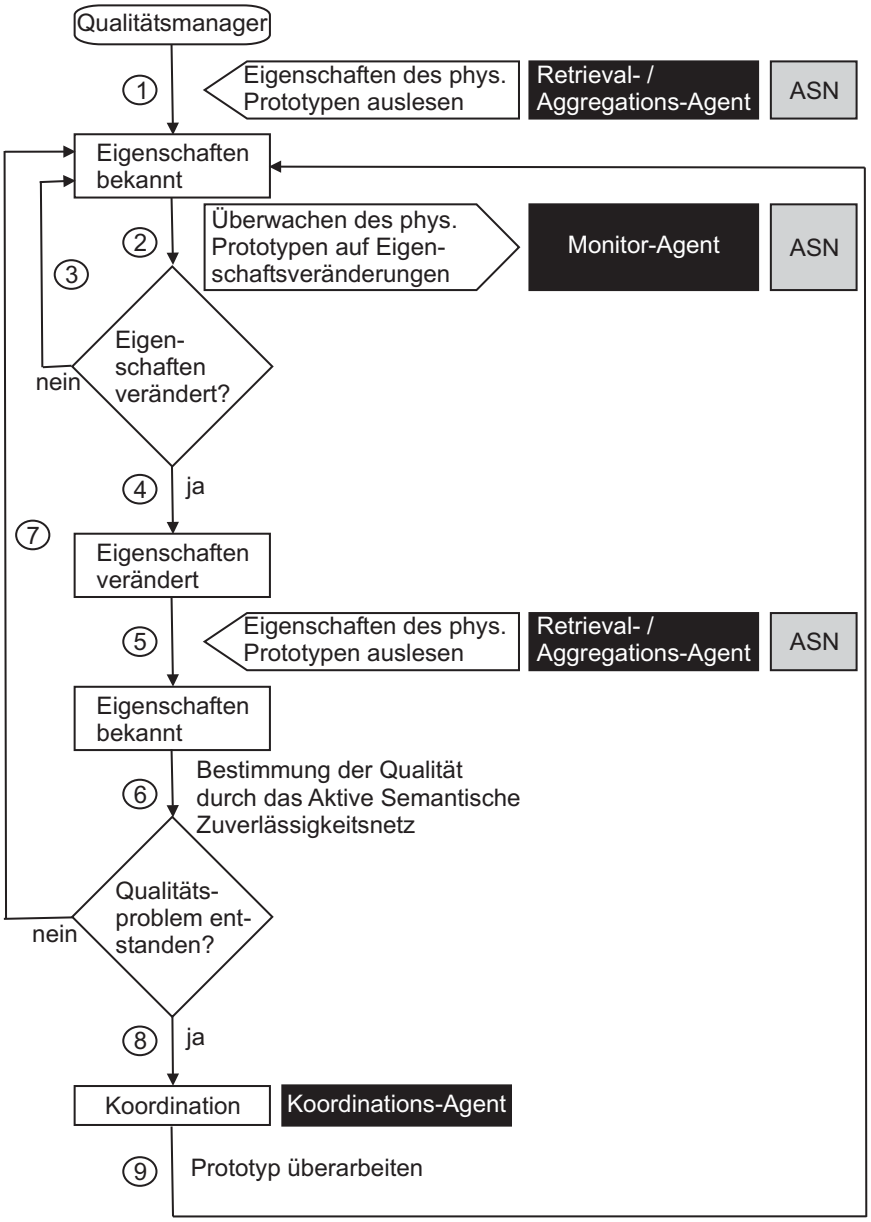


Bild 37: Anwendungsbeispiel: Qualitätsmanager

8.4 Kostenrechner

Der Kostenrechner hat zur Aufgabe, die Kosten des Prototyps zu überwachen und bei Überschreitung einzugreifen bzw. eine Lösung zusammen mit dem Prototypenbauer und eventuell dem Qualitätsmanager und dem Projektplaner zu erarbeiten. Er benutzt dafür den Prozess aus Bild 38 und setzt als RPD-Werkzeug das Marktorientierte Kostengestaltungsmodell ein [Cas 02].

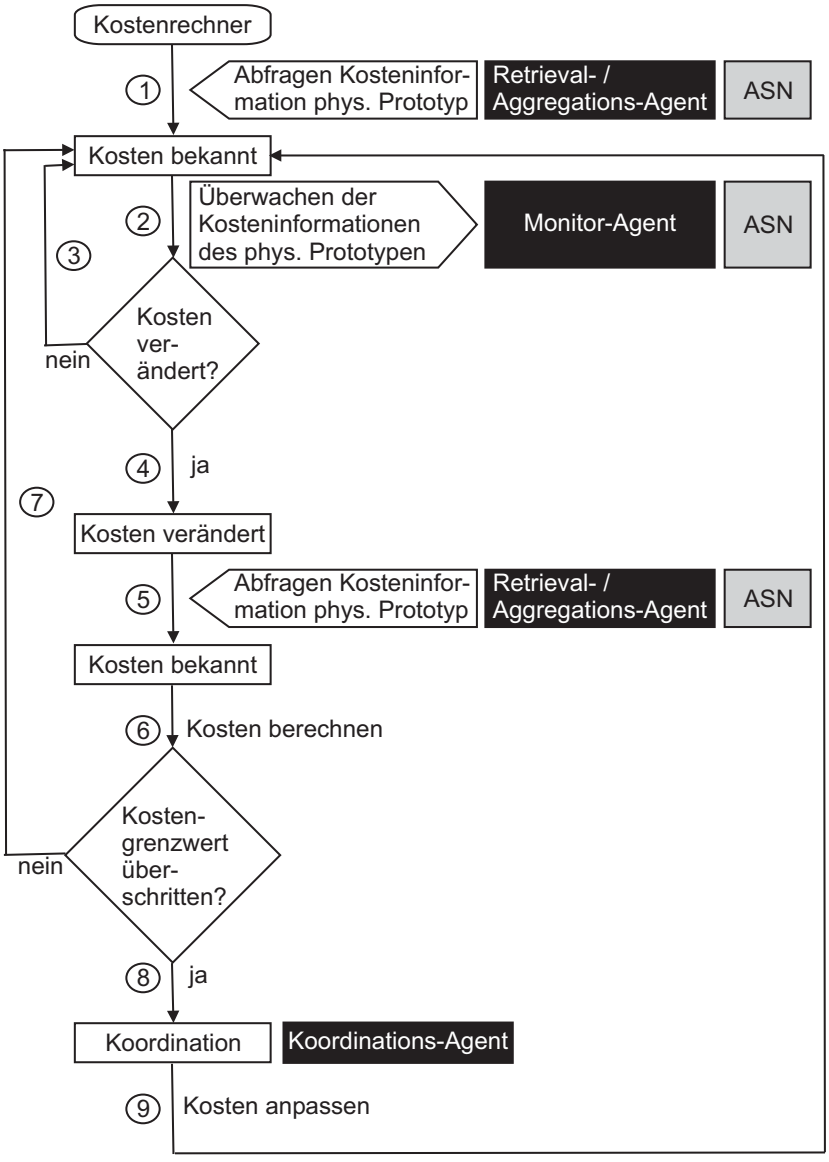


Bild 38: Anwendungsbeispiel: Kostenrechner

Ohne den Einsatz der RPD-Middleware werden zuerst die benötigten Kosteninformationen aus dem ASN ausgelesen (Schritt (1)), bevor diese dann zyklisch auf Veränderungen überwacht werden (Schritt (2) und (3)). Wurden die Kosten verändert, beispielsweise durch Austausch des Materials aus dem der Prototyp gefertigt wird, so werden die veränderten Kosten aus dem ASN beschafft und eine Neukal-

kulation der Gesamtkosten durchgeführt (Schritt (4) bis (6)). Sind die maximalen Kosten nicht überschritten, so wird mit der Überwachung der Kosteninformationen fortgefahren (Schritt (7)). Wurde allerdings der Grenzwert überschritten, so ist mindestens eine Koordination zwischen Kostenrechner und Prototypenbauer notwendig um die Kosten zu senken (Schritt (8)). Von den Ergebnissen dieser Koordination kann auch die Qualität und / oder der Projektplan betroffen sein, sodass auch der Qualitätsmanager und der Projektplaner in die Koordination einzubeziehen ist (Schritt (9)).

Durch den Einsatz der RPD-Middleware wird der Kostenrechner an mehreren Punkten unterstützt. Zum einen wird das Auslesen der Kosteninformationen mit Hilfe des Retrieval-/Aggregations-Agenten in den Schritten (1) und (5) vereinfacht und effizienter. Zum anderen entfällt auch hier das zyklische Überwachen der Informationen ebenso wie beim Qualitätsmanager (siehe Abschnitt 8.3). Das Koordinationsprotokoll zur Abstimmung der Kostenproblematik ist zusammen mit den anderen Koordinationsmechanismen im Abschnitt 8.6 ausgeführt.

8.5 Prototypenbauer

Der Prototypenbauer ist mit der Fertigung und Beurteilung der physischen und virtuellen Prototypen beauftragt. In Bild 39 ist die Prozesskette der einzelnen Prototypenarten und deren Erstellung dargestellt.

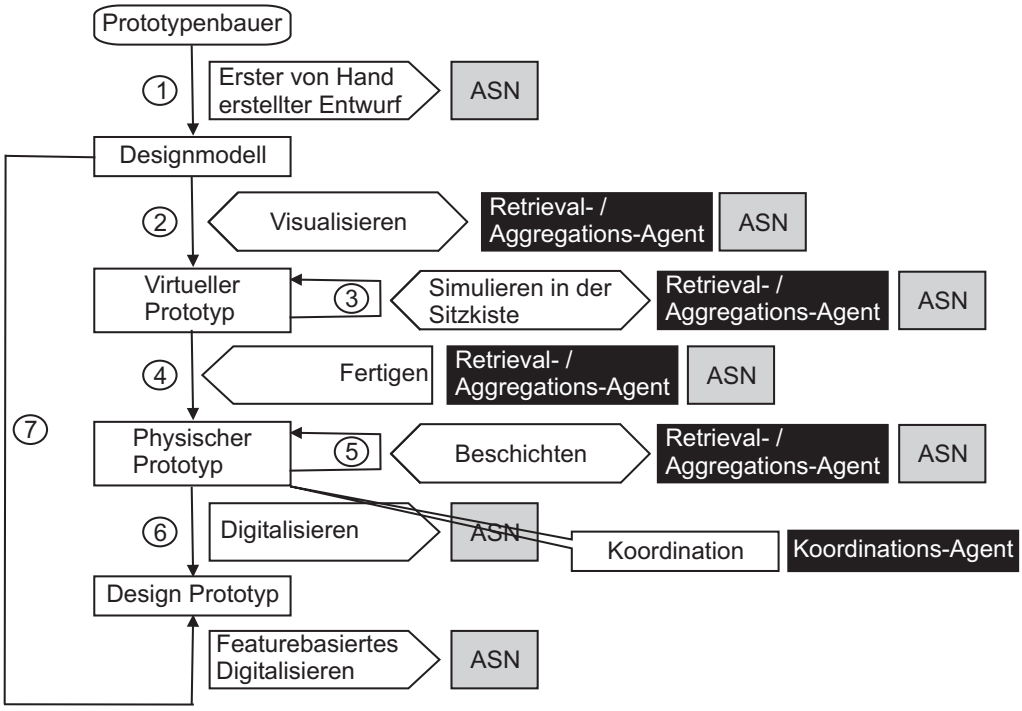


Bild 39: Anwendungsbeispiel: Prototypenbauer

Ohne Nutzung der RPD-Middleware wird in Schritt (1) ein erster Entwurf von Hand erstellt. Das so entstandene Designmodell wird nun entweder featurebasiert digita-

lisiert und in den *Design Prototypen* überführt (Schritt (7)) oder mit Visualisierungsverfahren in einen *virtuellen Prototyp* transformiert (Schritt (2)). Mit Hilfe des *virtuellen Prototyps* (siehe Bild 40) können nun verschiedene Testszenarien simuliert werden (Schritt (3)), z. B. eine Luftströmungssimulation in einer Sitzkiste. Im Zuge der Simulation werden die Ergebnisse und die daraus resultierenden Veränderungen im ASN abgelegt bzw. bei Bedarf ausgelesen. Hat der *virtuelle Prototyp* die einzelnen Simulationen erfolgreich durchlaufen, so wird auf der Basis seines Datensatzes ein *physischer Prototyp* (siehe Bild 41) in Schritt (4) erstellt. Anhand des *physischen Prototyps* werden nun z. B. verschiedene Beschichtungen getestet (Schritt (5)) und deren Ergebnisse und Auswirkungen auf den *physischen Prototypen* im ASN abgelegt. Im Schritt (6) ist es möglich den *physischen Prototypen* durch Digitalisierung wiederum in einen *Design Prototypen* zu überführen.

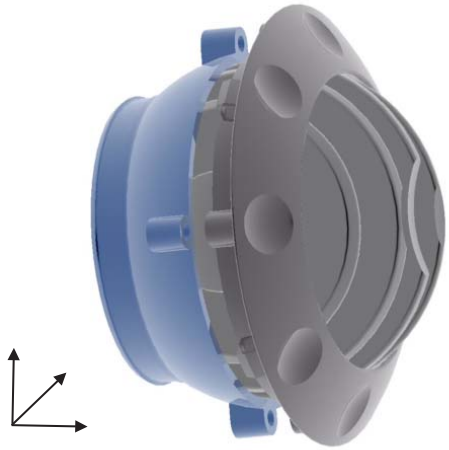


Bild 40: Anwendungsbeispiel: virtueller Prototyp der Luftdüse

Bild 41: Anwendungsbeispiel: physischer Prototyp der Luftdüse

Dieser Prozess wird durch die RPD-Middleware auf zwei Arten unterstützt. Zum einen wird durch den Einsatz der Retrieval- / Aggregations-Agenten die Informationsbeschaffung während der Prototyperstellung vereinfacht. Dabei werden die benötigten Geometriedaten aus dem ASN ausgelesen. Zum anderen werden während der Simulation und Fertigung des Prototyps laufend Veränderungen an dem Prototypen vorgenommen, wodurch Randbedingungen, wie Qualität, Kosten und Projektplan berührt werden. Die dadurch entstehenden Interessenkonflikte werden mit Hilfe von Monitor-Agenten aufgedeckt (siehe Abschnitt 8.2, 8.3 und 8.4) und mit Hilfe des Koordinations-Agenten aufgelöst (siehe Abschnitt 8.6). Dies bedeutet, dass z. B. der Qualitätsmanager (siehe Abschnitt 8.3) die Veränderungen am Prototyp sofort erfährt und darauf reagieren kann. Ohne Einsatz der RPD-Middleware wäre dies nicht möglich. Ein möglicher Koordinationspunkt ist hier am *physischen Prototyp* markiert. Eine Koordination kann aber auch bei der Bearbeitung des *virtuellen Prototyps* stattfinden.

8.6 Koordinationsprotokoll

Ein mögliches Koordinationsprotokoll ist zur Koordination der Kosten-, Projektplanungs- und Qualitätskonflikte notwendig und wird im Folgenden vorgestellt. Dieses Protokoll ist in Absprache mit allen Beteiligten erstellt worden und bildet die Grundlage des Koordinations-Agenten.

Zur besseren Übersicht ist das Koordinationsprotokoll als Zustandübergangsdiagramm dargestellt und in die vier Teile Start (siehe Bild 42), Projektplaner (siehe Bild 43), Kostenrechner (siehe Bild 44) und Qualitätsmanager (siehe Bild 45) aufgeteilt. Diese sind über die Konnektoren A bis F verbunden. Für die beteiligten Domänen werden die Abkürzungen Prototypenbauer (PB), Projektplaner (PP), Kostenrechner (KR), Qualitätsmanager (QM) in den Sender- und Empfängerlisten verwendet. Das vollständige Protokoll findet sich in der für den Koordinations-Agenten lesbaren XML-Notation im Anhang F.4.1. Im Anhang F.4.3 wird ein möglicher Beispieldurchlauf des Koordinationsprotokolls anhand der Automatendefinition und des Sequenzdiagramms erläutert.

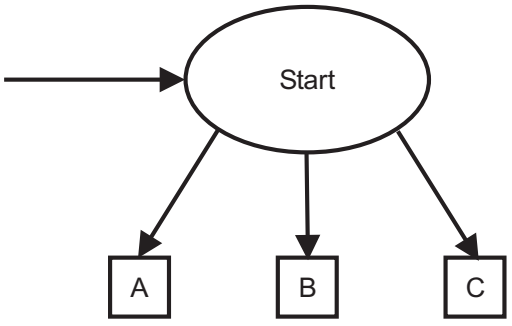


Bild 42: Anwendungsbeispiel: Koordinationsprotokoll – Start

Aus dem Startzustand (siehe Bild 42) wird je nach entstandener Konfliktsituation in den Bereich des Projektplaners, Kostenrechners oder Qualitätsmanager verzweigt.

Koordination: Projektplaner

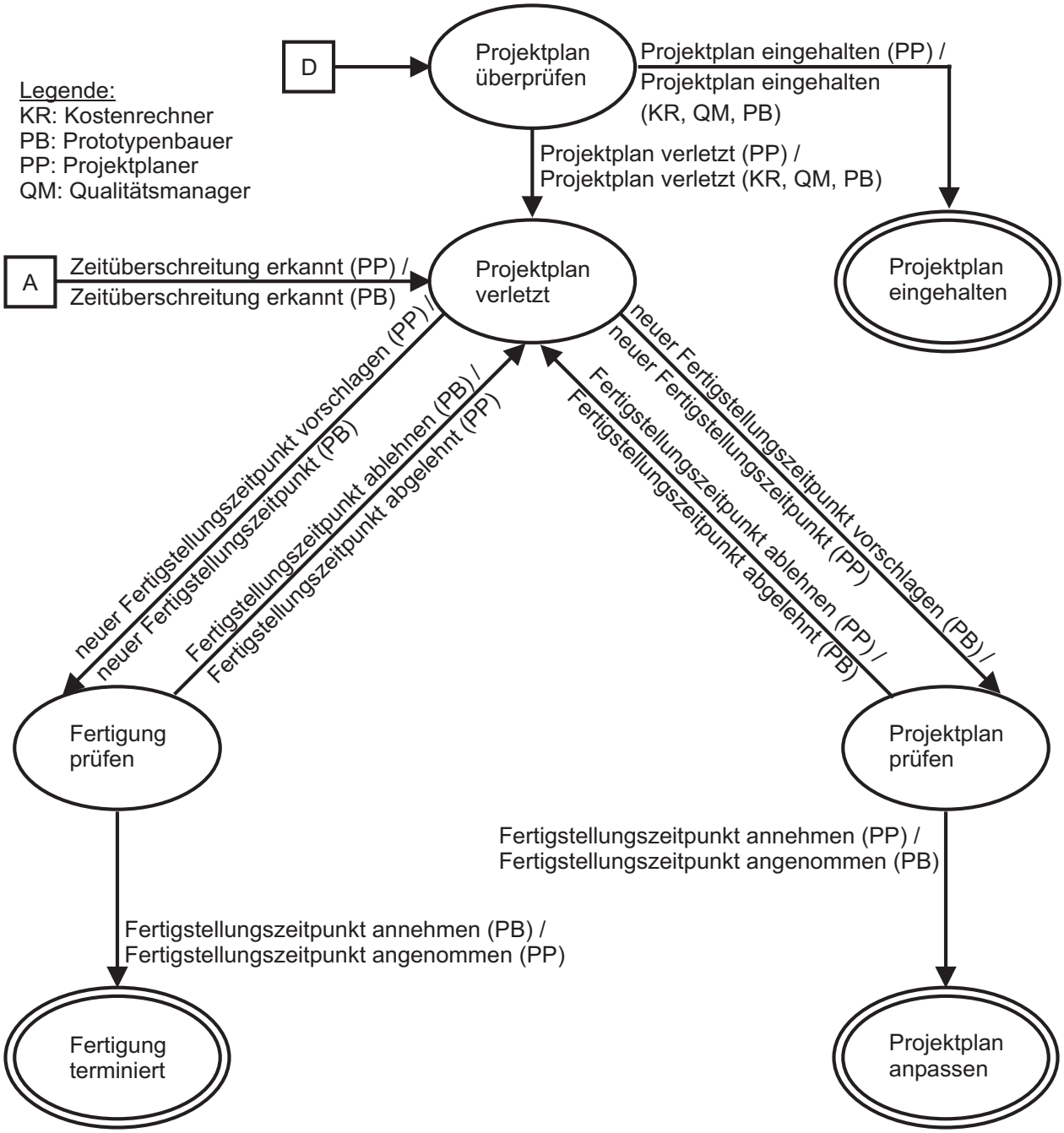


Bild 43: Anwendungsbeispiel: Koordinationsprotokoll – Projektplaner

Im Bereich des Projektplaners wird durch die oben beschriebene Vorgehensweise ein Zeitkonflikt entdeckt, dieser dem Prototypenbauer mitgeteilt und damit in den Zustand *Projektplan verletzt* gewechselt. Aus diesem Zustand heraus besteht die Möglichkeit, von beiden Seiten einen Vorschlag zur Lösung des Problems zu tätigen bzw. den Vorschlag des Koordinationspartners abzulehnen oder anzunehmen. Dies wird besonders deutlich durch dieselbe Eingabenachricht, aber durch die un-

terschiedliche Sender- bzw. Empfängerliste im Zustand *Projektplan verletzt*. Durch die Annahme wird der Endzustand des Koordinationsprotokolls erreicht. Die notwendigen Aktivitäten auszuführen, wie den Projektplan bzw. die Fertigung anzupassen, ist dann Aufgabe der entsprechenden RPD-Anwendung. Dieser Teil des Koordinationsprotokolls kann nur beendet werden, wenn eine beiderseitige Annahme erfolgt ist und hat damit eine Abstimmungswirkung aller Beteiligten.

Koordination: Kostenrechner

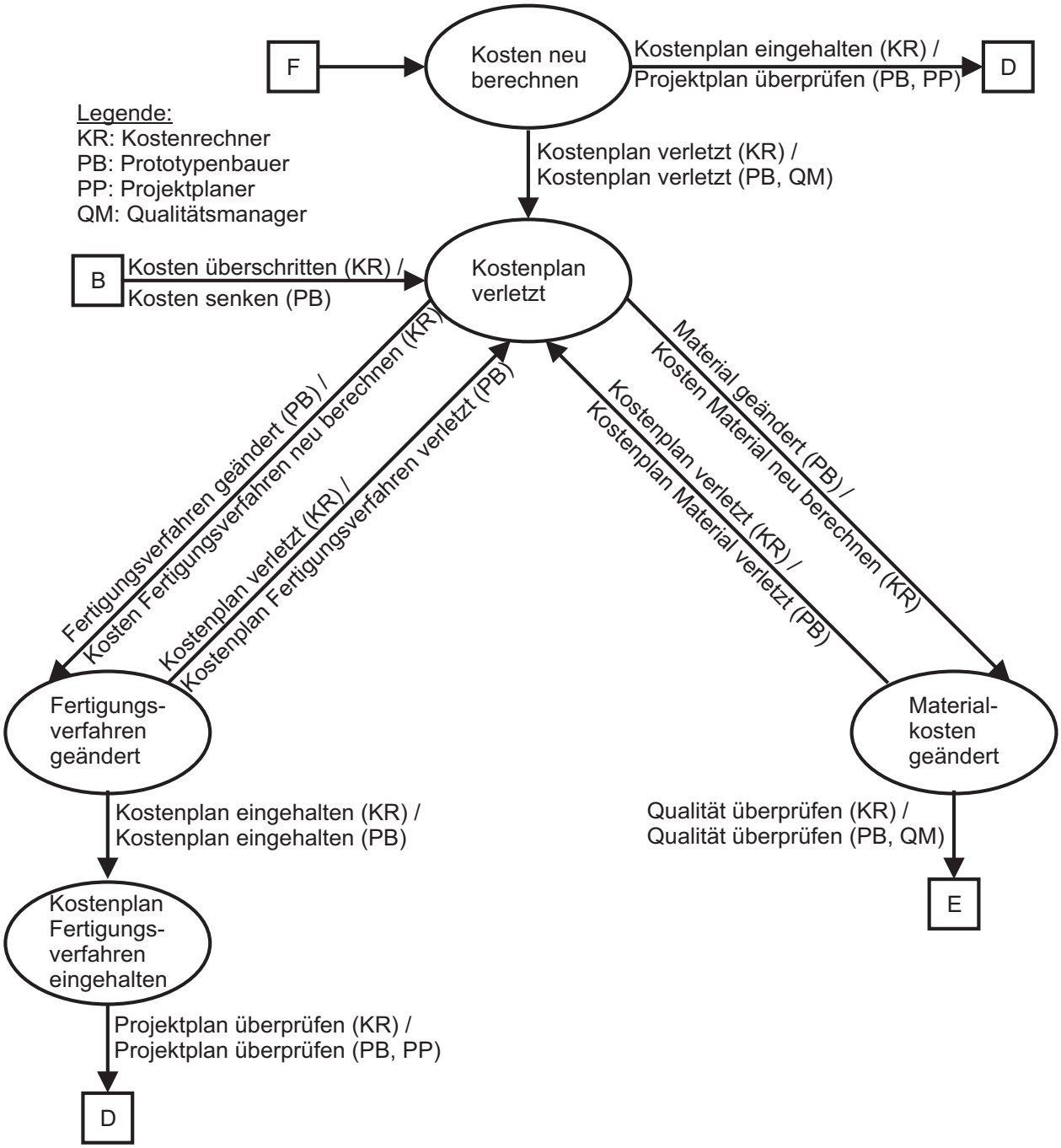


Bild 44: Anwendungsbeispiel: Koordinationsprotokoll – Kostenrechner

Im Bereich des Kostenrechners (siehe Bild 44) wird ähnlich verfahren wie beim Projektplaner. Auch hier wird durch den Kostenrechner erkannt, dass der Kostenplan verletzt wurde und damit eine Koordination eingeleitet. Das Koordinationsprotokoll sieht zwei Möglichkeiten vor die Kosten zu senken. Zum einen durch die Verwendung eines anderen Materials beim Prototypenbau und zum anderen durch das Verändern des Fertigungsverfahren. Beide Veränderungen, die durch den Prototypenbauer ausgeführt werden und damit auch Änderungen im ASN mit sich ziehen, werden durch den Kostenrechner validiert. Wird das Fertigungsverfahren verändert, wird hier davon ausgegangen, dass die Qualität des Prototyps erhalten bleibt, sodass mit der Überprüfung des Projektplans fortgefahren wird. Im anderen Fall, der Änderung des Materials wird zusätzlich noch die Qualität verifiziert. In beiden Fällen wird die angestoßene Verifikation auch dem Prototypenbauer mit den Nachrichten *Projektplan überprüfen* bzw. *Qualität überprüfen* mitgeteilt.

Koordination: Qualitätsmanager

In Bild 45 ist der Teil des Koordinationsprotokolls dargestellt, das sich mit dem Bereich des Qualitätsmanagers befasst. Hierzu wird durch den Qualitätsmanager entweder bei der Herstellung des Prototyps oder durch eine Veränderung des Prototyps, die während einer Kostensenkung entstanden ist, der Qualitätskonflikt aufgedeckt und dem Prototypenbauer mitgeteilt. Auch hier bestehen verschiedene Möglichkeiten die Qualität zu steigern. Zum einen kann ebenfalls, wie bei der Kostenproblematik, das Material geändert werden oder zum anderen einzelne Bauteile des Prototypen gegen andere Bauteile ausgetauscht bzw. ergänzt werden, um die Zuverlässigkeit und damit die Qualität zu steigern. Anschließend sind die Kosten und der Projektplan zu überprüfen.

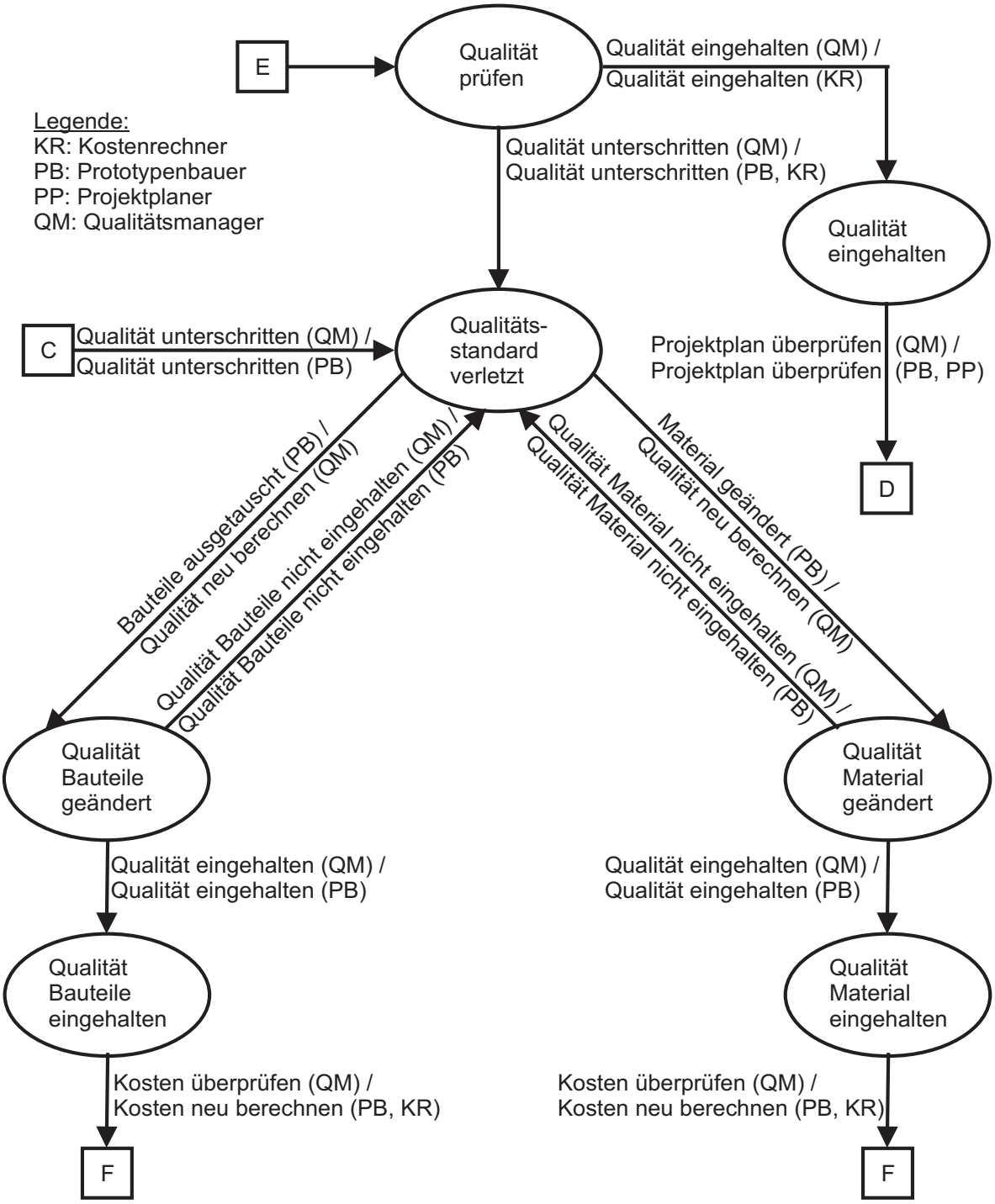


Bild 45: Anwendungsbeispiel: Koordinationsprotokoll – Qualitätsmanager

8.7 Bewertung der RPD-Middleware

Die RPD-Middleware muss sowohl quantitativ als auch qualitativ bewertet werden. Bei der qualitativen Bewertung wird untersucht, wie viel einfacher der Umgang mit dem ASN für den RPD-Nutzer durch die RPD-Middleware wurde. Im Rahmen der

qualitativen Bewertung werden die durch die RPD-Middleware hinzugewonnenen Funktionalitäten untersucht.

8.7.1 Quantitative Bewertung der RPD-Middleware

Bei der Betrachtung der RPD-Middlewarekomponente Informationsbeschaffung und -aufbereitung, dem Retrieval- und Aggregations-Agent, ist die Aufwandsreduktion in der Kommunikation gut messbar. Betrachtet man beispielsweise das *Abrufen des aktuellen Projektplans* aus dem Anwendungsbeispiel (siehe Abschnitt 8.2) so ist erkennbar, dass mit einer Anfrage an den Aggregations-Agenten der gesamte Projektplan aus dem ASN ausgelesen werden kann. Dieser Vorgang würde ohne Nutzung der RPD-Middleware bei einem Projektplan mit zehn Aktivitäten und zwölf Abhängigkeiten zwischen den Aktivitäten bereits 22 Anfragen erfordern. Ein ähnliches Bild ergibt sich bei der Abfrage der *Eigenschaften des physischen Prototypen* (siehe Abschnitt 8.3). Besitzt der Prototyp zwanzig Eigenschaften, wie beispielsweise seine Abmessungen, das verwendete Material, etc., so können diese mit einer Anfrage an den Retrieval-Agenten abgerufen werden, wodurch eine Reduktion der Kommunikationsschritte von zwanzig auf einen erfolgt.

Bei der quantitativen Beurteilung der Informationsüberwachung (Monitor-Agent) ist der Performancegewinn noch deutlicher. Ohne RPD-Middleware erfolgt die Überwachung des ASN durch zyklisches Abfragen. Je lückenloser die Überwachung sein soll, desto kürzer müssen die Abfrageintervalle der RPD-Anwendung sein. Mit Hilfe des Monitor-Agenten wird die Überwachung nur noch mit einer Anfrage gestartet und mit einer zweiten bei Bedarf beendet. Zusätzlich sendet der Monitor-Agent alle Veränderungen seiner Bedingungsanweisung an die RPD-Anwendung. Dies bedeutet, geht man beispielsweise von einer minütlichen Überwachung der *Kosteninformationen des physischen Prototypen*, wie im Anwendungsbeispiel dargestellt (siehe Abschnitt 8.4), aus, so entstehen ohne RPD-Middleware 60 Anfragen pro Stunde, während mit RPD-Middleware bei durchschnittlich fünf Veränderungen pro Stunde maximal sieben Kommunikationsvorgänge erfolgen.

Die quantitative Bewertung der Koordination kann nicht durch einfaches Abzählen der Kommunikationsschritte erfolgen, da der Koordinations-Agent nicht mit dem ASN kommuniziert. Eine quantitative Bewertung kann hier nur durch eine Abschätzung der Zeitaufwände zwischen Nutzung der RPD-Middleware und Nutzung herkömmlicher Techniken, wie beispielsweise Telefonkonferenzen, erfolgen. Betrachtet man die Vorgehensweise des Koordinations-Agenten, so ist ein Abstimmungstreffen zur Spezifikation des Koordinationsprotokolls notwendig. Während der Koordination sind dann je Koordinationsschritt mindestens zwei bis maximal N+2 Kommunikationsschritte notwendig. N ist dabei die Anzahl der Koordinationspartner, die über den Koordinationsschritt informiert werden müssen. Der Überhang von zwei Kommunikationsschritten entsteht durch die Einleitung und die Bestätigung des Koordinationsschritts. Ohne Einsatz der RPD-Middleware würde für jeden Koordinationsschritt ein Telefongespräch, ein Treffen, etc. benötigt werden. Diese können von wenigen Minuten bis mehrere Stunden dauern, während der Koordina-

tions-Agent für die Durchführung eines Koordinationsschritts weniger als eine Sekunde benötigt.

Aufgrund der hohen Kosten, die durch die gemeinsame Spezifikation des Koordinationsprotokolls entstehen, lohnt sich der Einsatz des Koordinations-Agenten erst dann, wenn die Koordination aufwendig ist und / oder wenn viele Abstimmungstreffen notwendig wären.

In Tabelle 9 sind die Ergebnisse der quantitativen Bewertung zusammengefasst.

RPD-Middlewarekomponente	Aufwand ohne RPD-Middleware	Aufwand mit RPD-Middleware
Informationsbeschaffung und -aufbereitung (Projektplan abfragen)	22 Anfragen	1 Anfrage
Informationsbeschaffung und -aufbereitung (Eigenschaften des physischen Prototypen abfragen)	20 Anfragen	1 Anfrage
Informationsüberwachung	60 Anfragen pro Stunde	2 einmalige Anfragen und 5 Anfragen pro Stunde
Koordination	X mehrminütige Telefonate Y mehrstündige Treffen	1 mehrstündiges Treffen mindestens X+Y Kommunikationsschritte zu je einer Sekunde

Tabelle 9: Qualitative Bewertung der RPD-Middleware

8.7.2 Qualitative Bewertung der RPD-Middleware

Bei der qualitativen Bewertung sind die Vor- und Nachteile der zusätzlichen Funktionen der RPD-Middleware und deren Nutzen zu beurteilen.

Die Vorteile der zusätzlichen Funktionen des Retrieval-Agenten gegenüber einer Standarddatenbank und dem ASN, wie Suchanfragen ohne Kenntnis der Informationsstruktur und Navigation wird besonders deutlich bei der Beschaffung der Eigenschaften des physischen Prototypen durch den Qualitätsmanager im Anwendungsbeispiel (siehe Abschnitt 8.3). Der Qualitätsmanager ist mit Hilfe der RPD-Middleware nun in der Lage durch Nutzung der Navigation Zusatzinformationen, wie z. B. Geometriedaten in Form eines Facettenmodells zu erhalten. Dies war bisher nicht möglich. Derartige Informationen mussten bisher direkt beim Prototypenbauer erfragt werden, wodurch ein Medienbruch und Verlust an Arbeitszeit entstand.

Des Weiteren ermöglicht die unscharfe Suche dem Qualitätsmanager ohne Kenntnis über das Datenmodell Informationen über ähnliche Prototypen abzufragen. Dies hat den Vorteil, dass er in seine Qualitätsbetrachtungen bereits gewonnene Erkenntnisse aus früheren Projekten einfließen lassen kann.

Wie bereits im vorangegangenen Abschnitt erwähnt, hängt der quantitative Vorteil des Monitor-Agenten stark von der Wahl des Überwachungsintervalls ab. Die Aufgabe des zyklischen Überwachens entfällt für die RPD-Anwendung durch die Nutzung des Monitor-Agenten. Dabei vereinfacht sich nicht nur der Arbeitsplan des Projektplaners, sondern es werden auch alle Veränderungen im ASN zuverlässig erkannt. Ohne Einsatz der RPD-Middleware kann es vorkommen, dass kurzfristige Veränderungen unentdeckt bleiben, wenn sie zwischen zwei Prüfvorgängen stattfinden. Dies bedeutet, dass durch die Überwachungskomponente eine direkte und schnelle Rückkopplung von Ereignissen und aufgetretenen Problemen möglich wird, die unerlässlich für die Integration von Experten unterschiedlicher Domänen ist.

Durch die Implementierung des ASN und des Monitor-Agenten gibt es im Bereich der Überwachung von Strukturänderungen ein Problem. Wird das Löschen von Informationselementen, Netze, Konzepte, Attribute, Relationen in der Überwachungsbedingung des Monitor-Agenten verwendet, so können die entsprechenden Teile der Bedingung nur einmalig ihren Wahrheitswert von *falsch* nach *wahr* ändern. Wird ein Informationselement gelöscht und wieder angelegt, so handelt es sich auch dann nicht um dasselbe, wenn alle Eigenschaften des Informationselements mit denselben Werten erneut belegt werden. Der Grund hierfür ist, dass im ASN für jedes neue Informationselement eine neue und eindeutige ID vergeben wird, auf die der Monitor-Agent aufbaut. Dies hat zur Folge, dass wenn ein Informationselement beispielsweise versehentlich gelöscht und sofort neu angelegt wird, die an den Monitor-Agenten gestellte Anfrage, die dieses Informationselement beinhaltet, erneut gestellt werden muss. Dieser Nachteil bezieht sich nur auf die Strukturfunktionen *DeleteAttribute*, *DeleteConcept*, *DeleteNet* und *DeleteRelation* und kann aufgrund der ASN-Realisierung nicht umgangen werden.

Durch die Schaffung einer einheitlichen informationstechnisch unterstützten Koordination ist es nun möglich, Abstimmungen über Fachgebietsgrenzen hinweg schnell, effizient und strukturiert durchzuführen (siehe Abschnitt 8.6). Besonders die flexiblen Gestaltungsmöglichkeiten des Koordinationsprotokolls führen zu einer hohen Akzeptanz, während das textuelle Erstellen des Koordinationsprotokolls mühsam und fehlerträchtig ist. Die Hauptvorteile sind, dass Koordinationsschritte sequenziell abgearbeitet werden, wodurch es möglich wird, dass jeder Koordinationspartner unabhängig von den anderen Koordinationspartnern die Koordination im Rahmen des Koordinationsprotokolls fortführt. Eine gleichzeitige Anwesenheit aller Koordinationspartner wird dadurch vermieden.

Bei der Betrachtung der Basisfunktionalität zeichnet sich die RPD-Middleware durch ihr einfaches Adressierungsschema bedingt durch den Einsatz der Multicast-

Technologie aus. Allerdings wird die Multicast-Technologie derzeit nicht von allen Serviceprovidern angeboten.

Bei der Realisierung der RPD-Middleware wurde darauf geachtet, dass keine direkten Eingriffe in die bestehenden RPD-Anwendung, durch die RPD-Middleware verursacht, notwendig werden. Dies wurde durch die XML-basierte Nachrichtenkommunikation erreicht.

Der RPD-Nutzer ist durch die RPD-Middleware nun in der Lage Informationen einfacher im ASN aufzufinden und weiterverarbeiten zu können. Er erhält die Informationen in wesentlich kürzerer Zeit. Durch den Einsatz der Informationsüberwachung wird der RPD-Nutzer proaktiv über für ihn relevanten Informationen benachrichtigt, wodurch keine Information verloren geht. Die Koordinationsunterstützung gibt dem RPD-Nutzer ein flexibles Instrument an die Hand, mit dem er jeder Zeit sich mit anderen RPD-Nutzern strukturiert nach einem vorher definierten Protokoll abstimmen kann.

Die RPD-Middleware hat durch ihre verbesserten Suchmechanismen und die zusätzlichen Funktionalitäten, wie die Überwachung von Informationen und die Koordination, eine hohe integrative Wirkung. Die abgeschlossenen Welten der einzelnen RPD-Domänen werden aufgebrochen und zusammengeführt. Dabei ist besonders im Bereich der Koordination eine intensive Mitarbeit aller Beteiligten notwendig und wünschenswert.

9 Zusammenfassung

Der stark arbeitsteilig organisierte Produktentstehungsprozess wird bestimmt durch die wachsende Zahl von spezialisierten Experten. Dadurch ist neben der ablaufbedingten Aufteilung von Aufgaben eine Verteilung der Kompetenzen auf Experten zu beobachten. Die Integration von neuen Produktfunktionen bedeutet einen erhöhten Abstimmungsbedarf. Dieser Abstimmungsbedarf bedingt eine erhöhte Kommunikation der Experten, um eine verbesserte Koordination des Produktentstehungsprozesses zu erzielen.

Der RPD-Prozess umfasst alle benötigten Wissensdomänen und deren Experten, die für die erfolgreiche Durchführung von RPD-Projekten notwendig sind. Dies sind vor allem die Bereiche Prototypenbau, Qualitäts- und Kostenmanagement, Projektplanung, Simulation und Bewertung von Prototypeigenschaften, Konstruktionsverfahren, sowie Verfahren zur optimalen Materialauswahl.

Die Integration der RPD-Experten und ihren Anwendungen müssen zwei Arten von Verteiltheit überwinden. Die erste Art ist eine inhaltliche Dimension, bedingt durch die Beteiligung unterschiedlicher Wissensbereiche am RPD-Prozess, wobei das explizite Wissen der RPD-Experten in unterschiedlicher Granularität und in unterschiedlicher Strukturierung vorliegt. Die zweite Art ist eine räumliche Dimension. Aufgrund der Tätigkeiten sind die am RPD-Prozess beteiligten Experten räumlich getrennt. Beide Dimensionen stellen grundlegende Problematiken für die Integration der RPD-Domänen in den RPD-Prozess dar.

Als erster Integrationsschritt wurde als gemeinsame Datenbasis das Aktive Semantische Netz (ASN) im Rahmen des Sonderforschungsbereichs 374 entwickelt. Die unterschiedlichen RPD-Domänen legen ihre Informationen, die für den RPD-Prozess von Bedeutung sind, im ASN ab. Dieses explizite Wissen ist dabei logisch nach den Zusammenhängen zwischen den Wissensdomänen strukturiert. Das ASN stellt den verteilten Entwicklungsteams des RPD eine gemeinsame Grundlage der Datenhaltung zur Verfügung, die alle Wissensbereiche des Produktentstehungsprozesses in einem integrierten Produktmodell repräsentiert und die Voraussetzung für Kommunikations- und Kooperationsmechanismen ist.

Die Aufgaben des ASN bewegen sich auf einer datenorientierten Ebene. Der Zugriff auf das ASN wird im Wesentlichen durch eine objektorientierte Programmierschnittstelle ermöglicht. Bei der Anwendung des ASN hat sich jedoch gezeigt, dass diese nahezu auf das Funktionsspektrum traditioneller Datenbanksysteme beschränkt bleibt, solange für die erweiterten Repräsentationskonstrukte keine adäquaten Anfragestrukturen bereitgestellt werden. Somit entsteht die Notwendigkeit, eine erweiterte Funktionalität zur Integration der interdisziplinären Anwendungen mit dem ASN in Form einer RPD-Middleware aufzubauen.

Ziel dieser Arbeit war die Entwicklung einer agentenbasierten RPD-Middleware, die die funktionale Lücke zwischen dem ASN, als Datenhaltung, und den RPD-Anwendungen, als Werkzeuge des Produktentstehungsprozesses, schließt, um eine Integration der RPD-Experten entlang des RPD-Prozesses zu erreichen. Die Agenten der RPD-Middleware nehmen dabei die Aufgaben der Informationsbeschaffung, -aufbereitung und -überwachung, sowie der Koordination wahr und nutzen die im ASN hinterlegten semantischen Zusammenhänge. Die für die Agenten benötigten funktionalen Methoden wurden im Laufe dieser Arbeit entwickelt.

Hierfür wurde in einem ersten Schritt die Problemstellung, verursacht durch die Randbedingungen der RPD-Anwendungen der einzelnen RPD-Domänen, wie unterschiedliche Informationen, verschiedene Detaillierungsgrade von Informationen, sowie unterschiedliche Repräsentationsformen ebenso berücksichtigt, wie die Randbedingungen des ASN (Kapitel 2).

Auf der Grundlage der Problemstellung wurden in Kapitel 4 Methoden für die RPD-Middleware und deren Bestandteile entwickelt. Es wurden erweiterte Suchmechanismen, wie z. B. die unscharfe Suche für die Informationsbeschaffung und -aufbereitung ebenso entwickelt, wie die Definition einer Überwachungsfunktion in Form einer Bedingungsanweisung für die Informationsüberwachung von Daten und Strukturen des ASN. Zur informationstechnischen Unterstützung der Abstimmungs- und Synchronisationsvorgänge im RPD-Prozess wurde eine Koordinationsmethode definiert. Das Koordinationsprotokoll, die Vorschrift wie die RPD-Experten und ihre Anwendungen abgestimmt und synchronisiert werden, wird über ein Zustandübergangsdiagramm definiert. Dieses wurde in Form eines endlichen Automaten durch den Koordinations-Agenten realisiert.

Mit Hilfe des Standes der Technik (siehe Abschnitt 2.2) und den Methoden wurde in Kapitel 5 die Architektur der RPD-Middleware entwickelt. Die Architektur besteht aus einem Agentenframework mit einer nachrichtenbasierten Kommunikation, die über eine Multicast-Umgebung abgewickelt wird. Durch diese Vorgehensweise entsteht eine geschlossene Kommunikation, die an ihren Schnittstellen einfach aufgebaut ist und damit einen flexiblen Zugang für die RPD-Anwendungen bietet.

In Kapitel 6 wurden die vier Agententypen Retrieval (Informationsbeschaffung), Aggregation (Informationsaufbereitung), Monitor (Informationsüberwachung) und Koordination entwickelt. Die Methoden aus Kapitel 4 wurden verfeinert und in eine prototypische Realisierung (siehe Kapitel 7) überführt. Die spezifischen Kommunikationssprachen, sowie die XML-basierten Nachrichtenformate finden sich im Anhang.

Anhand der Entwicklung einer Luftdüse für den Fahrzeuginnenraum wurde die RPD-Middleware verprobt (siehe Kapitel 8). Dabei wurde deutlich, dass durch den Einsatz der RPD-Middleware die einzelnen RPD-Domänen näher in ihrer gemeinsamen Arbeit zusammenrücken und damit eine zeitliche und qualitative Steigerung

des RPD-Prozesses möglich wurde ohne dabei vertiefende Kenntnisse über die jeweils andere RPD-Domäne besitzen zu müssen. Neben der zusätzlichen Funktionalität, die die RPD-Middleware zur Verfügung stellt, ist die Schnittstelle zum ASN auf ein abstrakteres Niveau angehoben worden, wodurch Detailkenntnisse über die interne Struktur der Informationsablage und -vernetzung obsolet und der Umgang mit dem ASN stark vereinfacht wurde.

10 Ausblick

Wie bei der Benutzung der RPD-Middleware im Abschnitt 8.7 festgestellt wurde, hat sich die textuelle Definition im XML-Format oder in der Anfragesprache des Koordinationsprotokolls in der Anwendung als fehlerträchtig und komplex herausgestellt. Es ist daher zu überlegen, eine Client-Anwendung zu entwickeln, die eine graphische Eingabe des Koordinationsprotokolls in Form des Zustandübergangsdiagramms ermöglicht.

Des Weiteren bietet der Koordinations-Agent keine Historie über den Ablauf des Koordinationsprotokolls. Es ist damit RPD-Anwendungen, die zu einem späteren Zeitpunkt in das Koordinationsprotokoll eingestiegen sind, nicht möglich die vorangegangenen Koordinationsschritte nachzuvollziehen. Zusätzlich wäre es hilfreich, wenn Ausgabe-Nachrichten nicht nur den RPD-Anwendungen zugänglich gemacht würden, sondern auch die Möglichkeit bestehen würde, diese in das ASN zurück zu schreiben bzw. Informationen aus dem ASN als Eingaben in den Koordinations-Agenten zuzulassen. Auf diesem Wege könnten Koordinationsabläufe von außen beeinflusst werden. Es wäre z. B. nicht mehr notwendig, die Kosten des Prototyps *Luftdüse* im ASN mit dem Monitor-Agenten zu überwachen und bei Überschreitung eines Schwellwertes der RPD-Anwendung zu melden, die dann die Koordination initiiert, sondern sinnvoller wäre es, wenn diese Information direkt aus dem ASN an den Koordinations-Agenten durchgereicht würde.

Die Informationsüberwachung durch den Monitor-Agenten ist in der bestehenden Form stark auf statische Informationen des ASN ausgerichtet. Es wäre wünschenswert dynamische Informationen, wie Suchergebnisse des Retrieval- und Aggregations-Agenten in die Überwachungsbedingung ebenso einfließen zu lassen, wie die Erweiterung der mathematischen Funktionen über die vier Grundrechenarten hinaus.

Aufgrund der starken Zunahme an Information, die auch im Umfeld des RPD stattfindet, hat es sich gezeigt, dass es geboten ist, das ASN in mehrere Partitionen aufzuteilen, die die benötigten Informationen für den eigenen RPD-Einsatz in lokalen ASN-Instanzen bereithalten. Dabei wird deutlich, dass nicht jede RPD-Domäne jede Information benötigt, aber im Gegensatz dazu, die Informationen, die von zwei oder mehreren RPD-Domänen genutzt werden, zwischen den verteilten ASN-Instanzen aktuell sein müssen. Die Entscheidung um welche Informationen es sich dabei handelt, ist Aufgabe der RPD-Anwendung und kann nicht durch das ASN vorhergesagt werden. Daher ist nach einem Mechanismus zu suchen, der es der RPD-Anwendung ermöglicht, die benötigten Informationen zu spezifizieren und zu synchronisieren. Auf der Basis der bereits existierenden Monitor- und Koordinations-Agenten ist es möglich einen Synchronisations-Agenten zu entwickeln, der den Monitor-Agenten zur Aufdeckung der Informationsänderungen und den Koordinations-Agenten zur Steuerung der Synchronisation nutzt.

Durch den geplanten Einsatz der RPD-Middleware im industriellen Umfeld werden die quantitativen und qualitativen Verbesserungen erprobt. Dabei ist von großem Interesse zu beobachten, wie sich die Verbesserungen in der Industrie im Vergleich zu den gewonnenen Erkenntnissen unter Laborbedingungen auswirken. Die Verwendung von realen Daten wird entscheidend sein, um die Vor- und Nachteile der Nutzung der RPD-Middleware zu erschließen. Durch die Nutzung eines größeren Datenbestands als unter Laborbedingungen wird eine deutliche Steigerung der Flexibilität durch den Einsatz der RPD-Middleware besonders im Bereich der Informationsbeschaffung und -aufbereitung erwartet.

Abstract

The strongly based on division of labor organized product developing process (RPD-Process) is defined by the increasing number of highly specialized experts. Thereby a distribution of the competencies on experts is recognized among the allocation of tasks. Due to an increased co-ordination need the integration of new product functions is more difficult. This co-ordination need causes increased communication of the experts, in order to obtain an improved co-ordination of the product developing process.

The RPD-Process contains all necessary knowledge domains and their experts, which are necessary for the successful execution of RPD projects. These are above all the areas of constructing of prototypes, quality and cost management, project planning, simulation and evaluation from prototype characteristics, construction procedures, as well as procedures to the optimal choice of materials.

The integration of the RPD-Experts and their applications must overcome two kinds of distribution. The first kind is a contentbased dimension, due to the participation of different knowledge domains in the RPD-Process, whereas the explicit knowledge of the RPD-Experts is present in different granularity and in different structure. The second kind is a spatial dimension. Due to the activities the experts taken part in the RPD-Process are spatially separated. Both dimensions represent fundamental problems for the integration of the RPD-Domains into the RPD-Process.

As the first integration step a common database the active semantic net (ASN) was developed in the context of the Sonderforschungsbereich 374. All for the RPD-Process needed information are stored of the different RPD-Domains in the ASN. This explicit knowledge is structured thereby after the context between the knowledge domains logical. Thereby the ASN provides a common basis of the data management to the distributed development teams of the RPD, which all knowledge areas of the product developing process in an integrated product model represented and which are a requirement for communication and co-operation mechanisms.

The tasks of the ASN are limited to a data-oriented level. That access to the ASN is essentially made possible by an object-oriented programming interface. With the application of the ASN however it turned out that functionality remains limited almost to the function spectrum of traditional database systems, as long as for the extended representation forms no adequate inquiry structures are made available. Therefore it is necessary to develop an extended functionality for the integration of interdisciplinary applications with the ASN in form of a RPD-Middleware.

A goal of this work was the development of an agent-based RPD-Middleware, which closes the functional gap between the ASN and RPD-Applications in order to reach an integration of the RPD-Experts along the RPD-Process. The agents of the

RPD-Middleware realize thereby the tasks of the provision of information, preparation and monitoring, as well as the co-ordination and use the semantic connections deposited in the ASN. The functional methods needed for the agents were developed in the course of this work.

Therefore in a first step the problem definition, caused by the constraint of RPD-Applications of the individual RPD-Domains, like different information, different degrees of detail of information, as well as likewise considers different representation forms, like the constraints of the ASN (chapter 2).

On the basis of the problem definition in chapter 4 methods for the RPD-Middleware and their components were developed. Extended search mechanisms, e.g. the indistinct search for the provision of information and preparation were as well developed, as the definition of a monitoring function in form of an IF-statement for the information monitoring of data and structures of the ASN. For the information-technical support of co-ordination and synchronization procedures during the RPD-Process a co-ordination method was defined. The co-ordination protocol, the rule like the RPD-Experts and their applications to be coordinated and synchronized, are defined over a state-transition-diagram. This is realized in form of a finite state machine by the co-ordination agent.

With the help of the state of the art (see section 2.2) and the methods, the architecture of the RPD-Middleware was developed in chapter 5. The Architecture consists of an agent-framework with message-based communication, which is processed over a multicast environment. From this proceeding results a closed communication, which is simply developed at their interfaces and so that for RPD-Applications offers a flexible access.

In chapter 6 the four agent types retrieval (provision of information), aggregation (information preparation), monitor (information monitoring) and co-ordination were developed. The methods from chapter 4 were refined and transferred into a prototypical realization (see chapter 7). The specific communication languages, as well as the XML-based message formats are shown in the appendix.

On the basis the development of an air duct for the interior of a car the RPD-Middleware was tested (see chapter 8). It became clear that by the deployment of the RPD-Middleware the individual RPD-Domains moved closer together in their common work. Thereby a temporal and qualitative improvement of the RPD-Process became possible without the need to have knowledge of all different RPD-Domains. Apart from the additional functionality, which makes the RPD-Middleware available, the interface to the ASN was raised to an abstract level, whereby detail knowledge of the internal structure of the data management and cross-linking and handling the ASN was unnecessary and highly simplified.

Literatur

Abi 97 Abitebou, S.; Quass, D.; u.a.: **The Loral Query Language for Semistructured Date.** In: International Journal on Digital Libraries (IJDL) (1997) Nr. 1, S. 68–88.

Abl 04 Able: **Agent Building and Learning Environment: IBM.** URL: http://www.alphaWorks.ibm.com/tech/able, 2004 (07.01.2005)

Agh 85 Agha, G. A.; Hewitt, C.: **Concurrent Programming Using Actors.** In: 5th Conference on Foundations of Software Technology and Theoretical Computer Science. Berlin u.a.: Springer Verlag, 1985, S. 10–34.

Agh 99 Agha, G. A.; Jamali, N.: **Concurrent Programming for DAI.** In: Multiagent Systems – A Modern Approach to Distributed Artificial Intelligence / Weiss, G. (Hrsg.). Cambridge, Massachusetts; London, England: The MIT Press, 1999, S. 505–537.

Ald 02 Aldunate, R.; Gonzalez, R.; Nussbaum, M.: **An Agent-Based Middleware for Supporting Spontaneous Collaboration among Co-Located, Mobile, and not necessarily Known People.** In: Workshop on Ad hoc Communications and Collaboration in Ubiquitous Computing Environments, CSCW 2002 in New Orleans, USA. 2002.

App 99 Appelrath, H.-J.; Freese, T.; u.a.: **Konzept eines Multiagentensystems für die verteilte Ablaufplanung.** In: Proceedings des Workshops „Agententechnologie“ auf der KI ’99: Agententechnologie – Multiagentensysteme in der Informationslogistik und wirtschaftswissenschaftliche Perspektiven der Agenten-Konzeptionalisierung, 14./15. September 1999 in Bonn / Timm, I. J.; Knirsch, P.; u.a. (Hrsg.). Bremen: 1999, S. 1–9.

Bae 02 Baeza-Yates, R.; Navarro, G.: **XQL and Proximal Nodes.** In: Journal of the American Society for Information Science and Technology JASIST 53 (2002) Nr. 6, S. 501–514.

Bar 97 Barbuceanu, M.; Fox, M. S.: **Integrating Communicative Action, Conversations and Decision Theory to Coordinate Agents.** In: Proceedings of Autonomous Agents ’97, February 2–5, 1997 in Marina del Rey. ACM Press, 1997, S. 49–58.

Bar 99 Barbuceanu, M.; Teigen, R.: **Higher Level Integration by Multi-Agent Architectures.** In: Handbook of Information System Architectures / Bernus, P. (Hrsg.). Springer Verlag, 1999.

Ber 03 Berglund, A.; Boag, S.; u.a.: **XML Path Language (XPath) 2.0.** URL: http://www.w3.org/TR/2003/WD-xpath20-20031112, Nov. 2003. (07.01.2005)

Ber 96 Bernstein, P.: Middleware: **A Model for Distributed System Services.** In: Communications of the ACM 39 (1996) Nr. 2, S. 86–98.

Ber 04 Bertsche, B.; Lechner, G.: **Zuverlässigkeit im Maschinenbau – Ermittlung von Bauteil- und System-Zuverlässigkeiten.** Berlin: Springer, 2004.

Big 02 Bigus, J. P.; Schlosnagle, D. A.; u.a.: **Able: A toolkit for building multiagent autonomic systems.** In: IBM Systems Journal 41 (2002) Nr. 3, S. 350–371.

Bia 97 Bianchi, R. A. C.; Rillo, A. H. R. C.: **A Purposive Computer vision System: a Multi-Agent Approach.** In: Proceedings of the 2nd Workshop on Cybernetic Vision in São Carlos, Los Alamitos, CA. IEEE Computer Society, 1997, S. 225–230.

Bir 00 Biron, P.V.; Malhotra, A. (Hrsg.): **XML Schema Part 2: Datatypes.** URL: http://www.w3.org/TR/2000/CR-xmlschema-2-20001024/, Oct. 2000. (07.01.2005)

Bla 01 Blake, M. B.: **Rule-Driven Coordination Agents: A Self-Configurable Agent Architecture for Distributed Control.** In: 5th International Symposium on Autonomous Decentralized Systems, ISADS, March 26–28, 2001 in Dallas, Texas. 2001, S. 271–277.

Bla 02 Blake, M. B.: **An Agent-Based Cross-Organizational Workflow Architecture in Support of Web Services.** In: WETICE Proceedings of the 11th IEEE International Workshops on Enabling Technologies: Infrastructure for Collaborative Enterprises. 2002, S. 176–181.

Boa 03 Boag, S.; Chamberlin, D.; u.a.: **XQuery 1.0: An XML Query Language.** URL: http://www.w3.org/TR/2003/WD-xquery-20031112/, Nov 2003. (07.01.2005)

Bon 01 Bonifati, A.; Lee, D.: **Technical Survey of XML Schema and Query Languages.** URL: http://www.cobase.cs.ucla.edu/tech-docs/dongwon/vldbjCom.pdf, 2001. (07.01.2005)

Boo 99 Booch, G.: **Das UML-Benutzerhandbuch.** Addison-Wesley, 1999.

Bra 98 Brauer, W.; Weiß, G.: **Multi-Machine Scheduling – A Multi-Agent Learning Approach.** In: Proceedings of the 3rd International Conference on Multi-Agent Systems. 1998, S. 42–48.

Bra 87 Bratman, M. E.: **Intentions, Plans, and Practical Reason.** Harvard University Press: Cambridge, MA, 1987.

Bra 88 Bratman, M. E.; Israel, D. J.; Pollack, M. E.: **Plans and resource-bounded practical reasoning.** Computational Intelligence, 1988, S. 349–355.

Bra 00 Bray, T.; Paoli, J.; Sperberg-McQueen, C. M. (Hrsg.): **Extensible Markup Language (XML) 1.0.** 2nd Edition. URL: http://www.w3.org/TR/2000/REC-xml-20001006, Oct. 2000. (07.01.2005)

Bra 04 Bray, T.; Paoli, J.; u.a.: **Extensible Markup Language (XML), XML-Standards.** URL: http://www.w3.org/TR/2004/REC-xml-20040204/, Feb. 2004. (07.01.2005)

Bri 00 Brickley, D.; Guha, R. V.: **Resource Description Framework (RDF) Schema Specification 1.0.** W3C Candidate Recommendation, Technical Report. URL: http://www.w3.org/TR/2000/CR-rdf-schema-20000327/, Mar. 2000. (07.01.2005)

Bro 86 Brooks, R. A.: **A robust layered control system for a mobile robot.** In: IEEE Journal of Robotics and Automation, Nr. 2 (1), 1986, S. 14–23.

Bro 91 Brooks, R. A.: **Intelligence Without Reason.** Massachusetts Institute of Technology; Artificial Intelligence Laboratory, A.I. Memo Number 1293, 1991.

Büc 89 Büchi, J. R.: **Finite automata, their algebras and grammars.** New York: Springer, 1989.

Bul 96a Bullinger, H. J.; Wißler, K.; Wörner, K.: **Rapid Product Development.** In: FB/IE 45 (1996) Nr. 2, S. 67–73.

Bul 96b Bullinger, H. J.; Hase, B.; Fischer, D.: **Kundenorientierte Entwicklung.** Stuttgart: IRB Verlag, 1996.

Bul 00 Bullinger, H.-J.; Frielingsdorf, H.; Hauss, I.; Roth, N.; Wagner, F.; Warschat, J.: **Vom Informations- zum Wissensmanagement in der Produktentwicklung.** In: Proceedings des Innovationsforums “Virtuelle Produktentstehung”, 11.–12. Mai 2000 in Berlin, F.-L. Jruase (Hrsg.), 2000.

Bul 04a Bullinger, H.-J.; Warschat, J.; Diederich, M. K.: **Multi-agent-based Middleware to support construction in Rapid Product Development with an active semantic network.** In: Proceedings of the 2nd International Conference on Robotics. October 14–16 2004 in Timisoara & Resita, Rumania, 2004, S. 35–36.

Bul 04b Bullinger, H.-J.; Warschat, J.; Diederich, M. K.: **Multi-agent-based Middleware to support construction in Rapid Product Development with an active semantic network.** In: Robotica & Management (2004), S. 18–24.

Cab 98a Cabri, G.; Leonardi, L.; u.a.: **How to Coordinate Internet Applications based on Mobile Agents.** In: 7th IEEE Workshop on Enabling Technologies: Infrastructures for Collaborative Enterprises in Stanford (CA), USA. 1998.

Cab 98b Cabri, G.; Leonardi, L.; u.a.: **Reactive Tuple Spaces for Mobile Agents Coordination.** In: 2nd International Workshop on Mobile Agents, Stuttgart, Germany. 1998.

Car 89 Carroll, J.: **Theory of finite automata.** Englewood Cliffs, NJ: Prentice-Hall, 1989. ISBN: 0-13-913708-4

Cas 02 Cassack, I.: **Prozessorientiertes Kostenmanagement am Beispiel der prototypgestützten Kostengestaltung für Dienstleistungen.** In: REFA-Nachrichten 55 (2002) Nr. 1, S. 23–26.

Cas 95 Castelfranchi, C.: **Guarantees for autonomy in cognitive agent architecture.** In: Intelligent Agents: Theories, Architectures and Languages (LNAI Volume 890), Wooldridge, M.; Jennings, N. R. (Hrsg.), Springer Verlag, Heidelberg, 1995, S. 56–70.

Cat 00 Cattell, R. G. G.; Barry, D.; u.a.: **The Object Database Standard – ODMG 3.0.** Morgan Kaufmann Publishers Inc., 2000. ISBN: 1-55860-647-5.

Ceb 04 Cebulla, T.: **Entwicklung einer asynchronen Kommunikationsunterstützung für multidisziplinäre Teams.** Jost-Jetter Verlag, Stuttgart, Univ., Diss., 2004.

Cel 95 Celentano, A.; Fugini, M.; Cefriel, S. P.: **Knowledge-Based Document Retrieval in Office Environments: The Kabiria System.** In: ACM Transactions on Information Systems 13 (1995) Nr. 3, S. 237–268. ISSN:1046-8188

Cer 99 Ceri, S.; Comai, S.; u.a.: **XML-GL: a Graphical Language for Querying and Restructuring WWW Data.** In: International World Wide Web Conference (WWW) in Toronto, Canada. 1999.

Cha 00 Chamberlin, D.; Robie, J.; Florescu, D.: **Quilt: An XML Query Language for Heterogeneous Data Sources.** In: International Workshop on the Web and Databases (WebDB) in Dallas, TX. 2000. URL: http://www.almaden.ibm.com/cs/people/chamberlin/quilt_lncs.pdf (07.01.2005)

Che 99 Chen, L. T.: **AgentOS: The Agent-based Distributed Operation System for Mobile Networks.** URL: http://turing.acm.org/crossroads/xrds5-2/agentos.html, 1999. (07.01.2005)

Cla 99 Clark, J. (Hrsg.): **XSL Transformations (XSLT) Version 1.0.** URL: http://www.w3.org/TR/xslt, Nov. 1999. (07.01.2005)

Coh 95 Cohen, P.R.; Levesque, H. J.: **Communicative Actions for Artificial Agents.** In: First International Conference on Multi-Agent Systems (ICMAS) / Lesser, V. (Hrsg.), The MIT Press, 1995, S.65–72.

Cor 88 Corkill, D. D.; Gallagher, Q.; u. a.: **GBB: A generic blackboard development system.** In: Blackboard Systems, Engelmore, R.; Morgan, T. (Hrsg.), Addison Wesly Publishing Company, 1988.

Cov 96 Covington, M. A.: **Toward a New Type of Language for Electronic Commerce.** In: Proceedings of the 29th Annual Hawaii International Conference on System Sciences, vol. 4. 1996, S. 329–336.

Cov 98 Covington, M. A.: **Speech Acts, Electronic Commerce, and KQML.** In: Decision Support Systems 22. Elsevier, 1998, S. 203–211. URL: http://www.ai.uga.edu/~mc/speechac.pdf (07.01.2005)

Cue 99 Cuena, J.; Ossowski, S.: **Distributed Models for Decision Support.** In: Multiagent Systems – A Modern Approach to Distributed Artificial Intelligence / Weiss, G. (Hrsg.). Cambridge, Massachusetts; London, England: The MIT Press, 1999, S. 459–504.

Dal 05 Dalakakis, S.; Diederich, M. K.; Roller, D.; Warschat, J.: **Multiagentensystem zur Wissenskommunikation in der Produktentstehung – Rapid Product Development.** In: Wirtschaftsinformatik 2005 – eEconomy, eGovernment, eSociety / Ferstl, O. K.; Sinz, E. J.; Eckert, S.; Isselhorst T. (Hrsg.). Heidelberg: Physica-Verlag, 2005, S. 1621–1640. ISBN: 3-7908-1574-8.

Dav 99 Davidson, A.; Fuchs, M.; u.a.: **Schema for Object Oriented XML 2.0.** URL: http://www.w3.org/TR/NOTE-SOX, Jul. 1999. (07.01.2005)

Dav 83 Davis, R.; Smith, R. G.: **Negotiation as a Metaphor for Distributed Problem solving.** In: Artificial Intelligence 20 (1983) Nr. 1, S. 63–109.

Day 83 Day, J. D.; Zimmermann, H.: **The OSI Reference Model.** In: Proceedings of the IEEE, vol. 71, 1983, S. 1334–1340.

DeP 00 De Palma, N.; Bellissard, L.; u.a.: **The AAA Agent-based Message Oriented Middleware.** In: C3DS ESPRIT Long Term Research Project, 3rd Year Report, 2000.

Deu 98 Deutsch, A.; Fernandez, M. F.; u.a.: **XML-QL: A Query Language for XML.** In: WWW The Query Language Workshop (QL) in Cambridge, MA. Aug. 1998. URL: http://www.w3.org/TR/1998/NOTE-xml-ql-19980819/ (07.01.2005)

Dew 89 Dewitz, S. K.; Lee, R. M.: **Legal procedures as formal conversations: contracting on a performative network.** In: Proceedings of the 10th International Conference on Information Systems. 1989, S. 53–65.

Die 00 Diederich, M. K.; Leyh, J.: **Dynamic Planning Portal for Rapid Product Development.** In: Proceedings for the Conference on Rapid Prototyping in the Automotive Industries, 33rd International Symposium on Automotive Technology and Automation (33rd ISATA), September 25–27, 2000 in Dublin, Ireland. 2000.

Die 01 Diederich, M. K.; Leyh, J.: **Agent Based Middleware to Coordinate Distributed Development Teams in the Rapid Product Development.** In: Proceedings for the Conference on Product Development, 1st Automotive and Transportation Technology Congress and Exhibition (1st ATTCE) October 1–4, 2001 in Barcelona, Spain. 2001.

Dij 68 Dijkstra, E. W.: **The structure of The Multiprogramming System.** In: Communication of the ACM, 1968, S. 341–346.

Din 00 Ding, Y.; Malaka, R.: **An Agent-based Architecture for Resource-Aware Mobile Computing.** In: Intelligent Interactive Assistance and Mobile Multimedia Computing. Proceedings of the International Workshop IMC2000 / Heuer, A.; Kirste, T. (Hrsg.). Rostock: 2000, S. 60–66.

dIn 98 d'Inverno, M.; Kinny, D.; Luck, M.: **Interaction Protocols in Agentis.** In: International Conference on Multi Agent Systems (ICMAS). IEEE Computer Society / B., W. (Hrsg.). 1998, S. 112–119.

Dou 98 Doug, B.: **ODMG 2.0: A Standard for Object Storage.** Component Strategies, 1998.

Dra 03 Draper, D; Fankhauser, P.; u. a.: **XQuery 1.0 and XPath 2.0 Formal Semantics.** URL: http://www.w3.org/TR/2003/WD-xquery-semantics-20031112/, Nov. 2003. (07.01.2005)

Dur 87 Durfee, E. H.; Lesser, V. R.; Corkill, D. D.: **Coherent Cooperation among Communicating Problem Solvers.** In: IEEE Transactions on Computers C-36 (1987) Nr.11, S. 1275–1291.

Dur 88 Durfee, E. H.: **Coordination of Distributed Problem Solver.** Kluwer Academic Publisher, Boston, 1988.

Eck 00a Eck, O.: **Ein kooperatives Transaktionssystem für CAD-Datenbanken.** Shaker Verlag, Stuttgart, Univ., Diss., 2000.

Eck 00b Eckstein, R.: **XML – kurz und gut.** O'Reilly Verlag, 2000.

Fal 94 Faloutsos, C.; Barber, R.; u.a.: **Efficient and Effective Querying by Image Content.** In: Journal of Intelligent Information Systems 3 (1994) Nr. 3-4, S. 231–262.

Fan 02 Fankhauser, P.; Lehti, P.: **XQuery by the book – an implementation based on rigid formal semantics.** URL: http://www.idealliance.org/papers/xml02/dx_xml02/papers/05-01-04/05-01-04.html, 2002. (07.01.2005)

Fat 98 Fatima, S. S.; Uma, G.: **An Adaptive Organizational Policy for Multi Agent Systems – AASMAN.** In: International Conference on Multi Agent Systems (ICMAS). IEEE Computer Society / B., W. (Hrsg.). 1998, S. 120–127.

Fei 95 Feitelson, D. G.; Rudolph, L. (Hrsg.): **Job scheduling strategies for parallel processing.** Springer-Verlag, 1995. ISBN: 3-540-60153-8

Fer 96 Ferber, J.: **Reactive distributed artificial intelligence.** In: Foundations of Distributed Artificial Intelligence, O'Hare, G. M. P.; Jennings, N. R. (Hrsg.), 1996, S. 287–317.

Fer 98 Ferber, J.; Gutknecht, O.: **A meta-model for the analysis and design of organizations in multi-agent systems.** In: ICMAS Proceedings of the 3rd International Conference on Multi Agent Systems. IEEE Computer Society, 1998, S. 128.

Fer 03a Ferber, R.: **Information Retrieval – Suchmodelle Data Mining – Verfahren für Textsammlungen und das Web.** Heidelberg: dpunkt. Verlag, 2003. URL: http://information-retrieval.de/irb/ir.part_4.html (07.01.2005)

Fer 03b Fernandez, M.; Malhotra, A.: u.a.: **XQuery 1.0 and XPath 2.0 Data Model.** URL: http://www.w3.org/TR/2003/WD-xpath-datamodel-20031112/, Nov. 2003. (07.01.2005)

Fer 99 Fernandez, M.; Simeon, J.; Wadler, P.: **XML Query Languages: Experiences and Exemplars.** URL: http://www.w3.org/1999/09/ql/docs/xquery.html, 1999. (07.01.2005)

Fin 92 Finin, T.; Weber, J.; u.a.: **Specification of the KQML Agent Communication Language plus example agent policies and architectures.** In: The DARPA Knowledge Sharing Initiative, External Interfaces Working Group, 1992.

Fin 93 Finin, T.; Weber, J.; u.a.: **Draft Specification of the KQML Agent Communication Language plus example agent policies and architectures.** Manuscript obtained from http://www.cs.umbc.edu, 1993. (07.01.2005)

Fin 94 Finin, T.; Fritzson, R.; u.a.: **KQML as an Agent Communication Language.** In: Proceedings of the 3rd International Conference on Information and Knowledge management (CIKM '94) in Gaithersburg, Maryland, USA. ACM Press, 1994, S. 456–463.

Fin 97 Finin, T.; Labrou, Y. und Mayfield, J.: **KQML as an Agent Communication Language.** In: Software Agents, Bradshaw, J. M. (Hrsg.), Menlo Park, Calif.; AAAI Press, 1997, S. 291–316.

FIP 02a **FIPA Abstract Architecture Specification.** In: Foundation for Intelligent Physical Agents (3-12-2002) Nr. SC00001L, 2002.

FIP 02b **FIPA ACL Message Structure Specification.** In: Foundation for Intelligent Physical Agents (3-12-2002) Nr. SC00061G, 2002.

FIP 02c **FIPA Communication Act Library Specification.** In: Foundation for Intelligent Physical Agents (12-3-2002) Nr. SC00037J, 2002.

FIP 03 **FIPA-OS: Foundation for Intelligent Physical Agents Open Source from Nortel Networks.** URL: http://fipa-os.sourceforge.net/summary.htm. (07.01.2005)

Fis 99 Fischmeister, S.; Lugmayr, W.: **The Supervisor-Worker Pattern.** In: Proceedings of Pattern Languages of Programs (PLoP '99) in Kubova Hut, Czech Republic. 1999.

Flo 99 Flores-Mendez, R. A.: **Towards a Standardization of Multi-Agent System Frameworks.** In: ACM Crossroads, Special Issue on Intelligent Agents Issue Nr. 5.4 (1999), S. 18–24. URL: http://www.acm.org/crossroads/xrds5-4/multiagent.html (07.01.2005)

Fuh 94 Fuhr, N.; Pfeifer, U.: **Probabilistic Information Retrieval as a Combination of Abstraction, Inductive Learning, and Probabilistic Assumptions.** In: ACM Transactions on Information Systems 12 (1994) Nr.1, S. 92–115.

Fuh 00 Fuhr, N.; Großjohann, K.: **XIRQL An Extension of XQL for Information Retrieval.** In: Proceedings ACM SIGIR 2000 Workshop on XML and IR, June 5, 2000 in Athens, Greece. 2000, S. 11–17.

Fuh 01 Fuhr, N.; Großjohann, K.: **XIRQL: A Query Language for Information Retrieval in XML Documents.** In: SIGIR ’01, September 9–12, 2001 in New Orleans, Lousiana, USA. 2001. S. 172–180.

Fuh 03 Fuhr, N.: **XML Information Retrieval and Information Extraction.** In: Text Mining. Theoretical Aspects and Applications. 2003, S. 21–32.

Fun 99 Funk, P.; Vierke, G.; Bürckert, H.-J.: **Distributed Intermodal Transportation Planning.** In: Proceedings des Workshops „Agententechnologie“ auf der KI ’99: Agententechnologie – Multiagentensysteme in der Informationslogistik und wirtschaftswissenschaftliche Perspektiven der Agenten-Konzeptionalisierung, 14./15. September 1999 in Bonn / Timm, I. J.; Knirsch, P.; u.a. (Hrsg.). Bremen: 1999, S. 59–69.

Gei 95 Geihs, K.: **Client/Server-Systeme: Grundlagen und Architekturen.** Thomson’s Aktuelle Tutorien, Thomson Publishing, 1995.

Gen 87 Genesereth, M. R.; Nilsson, M.: **Logical Foundations of Artificial Intelligence.** Morgan Kaufmann Publishers: San Mateo, CA, 1987.

Gen 92 Genesereth, M. R.; Fikes, R. E.: **Knowledge Interchange Format Version 3.0 Reference Manual.** In: Report Logic (1992) Nr. 1.

Gen 94 Genesereth, M. R.; Ketchpel, S. P.: **Software Agents.** In: Communications of the ACM 37 (1994) Nr. 7, S. 48–53.

Geo 99 Georgeff, M.; Pell, B.; u.a.: **The Belief-Desire-Intention Model of Agency.** In: Proceedings of the 5th International Workshop on Intelligent Agents V: Agent Theories, Architectures, and Languages (ATAL-98), 1555 / Müller, J.; Singh, M. P.; Rao, A. S. (Hrsg.). Heidelberg, Germany: Springer-Verlag, 1999, S. 1–10.

Gia 00 Giampapa, J. A.; Paolucci, M.; Sycara, K.: **Agent Interoperation Across Multiagent System Boundaries.** In: Proceedings of the 4th International Conference on Autonomous Agents / Sierra, C.; Gini, M.; Rosenschein, J. S. (Hrsg.). Barcelona, Catalonia, Spain: ACM Press, 2000, S. 179–186.

Git 01 Gittens, M. A.: **Implementation of an Agent Monitoring System in a JINI Environment with Restricted User Access.** URL: http://www.csee.umbc.edu/cadip/2001Symposium/AMAProjectPaper2001.pdf, 2001. (07.01.2005)

Gol 99 Goldman, R.; McHugh, J.; Widom, J.: **From Semistructured Data to XML: Migrating the Lore Data Model and Query Language.** In: Workshop on the Web and Databases (WebDB '99). 1999, S. 25–30.

Här 99 Härder, T.; Rahm, E.: **Datenbanksysteme – Konzepte und Techniken der Implementierung.** Springer, 1999. ISBN: 3-540-65040-7

Hau 99 Haustein, S.; Lüdecke, S.: **Kombination von Agenten- und Blackboard-Technologien für betriebswirtschaftliche Anwendungen.** In: Proceedings des Workshops „Agententechnologie" auf der KI '99: Agententechnologie – Multiagentensysteme in der Informationslogistik und wirtschaftswissenschaftliche Perspektiven der Agenten-Konzeptionalisierung, 14./15. September 1999 in Bonn / Timm, I. J.; Knirsch, P.; u.a. (Hrsg.). Bremen: 1999, S. 1–7.

Hay 85 Hayes-Roth, B.: **A blackboard architecture for control.** In: Artificial Intelligence 26, 1985, S. 251.

Hea 93 Hearst, M.; Plaunt, C.: **Subtopic structuring for full-length document access.** In: Proceedings of the 16th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval in New York. 1993, S. 59–68.

Her 94 Herrtwich, R. G.; Hommel, G.: **Nebenläufige Programme.** Springer-Lehrbuch, Springer Verlag, 1994.

Hew 03 Hewlett Packard: **Hewlett Packard Message Service.** URL: http://www.hpmiddleware.com/HPISAPI.dll/hpmiddleware/default.jsp, 2003. (07.01.2005)

Huh 99 Huhns, M. N.; Stephens, L. M.: **Multiagent Systems and Societies of Agents.** In: Multiagent Systems – A Modern Approach to Distributed Artificial Intelligence / Weiss, G. (Hrsg.). Cambridge, Massachusetts; London, England: The MIT Press, 1999, S. 79–120.

IEEE 802.5 IEEE 802.5: **Tokenring.**

ISO 22250 ISO / IEC: **Information Technology – Text and Office systems – Regular Language Description for XML (RELAX) – Part 1: RELAX Core.** 2000, DIS 22250-1. URL: http://www.eos.org.eg/web_en/prog/items/w34922.html (07.01.2005)

ISO 14977 ISO / IEC 14977 : 1996(E): **Information technology – Syntactic Metalanguage – Extended BNF.**

Jad 03 **Jade – Das Java Agent Development Framework.** URL: http://sharon.cselt.it/projects/jade/ (07.01.2005)

Jag 93 Jagannathan, V.: **Blackboard Architectures and Applications.** Academic Press, London.

Jel 00 Jelliffe, R.: **The Schematron.** URL: http://xml.ascc.net/resource/schematron/, 2000. (07.01.2005)

Jen 90 Jennings, N. R.: **Coordination Techniques for Distributed Artificial Intelligence.** In: Foundations of Distributed Artificial Intelligence / O'Hare, G. M. P.; Jennings N. R. (Hrsg.). London: Wiley, 1990, S. 187–210.

Jen 95 Jennings, N. R.; Corera, J. M.; Laresgoiti, I.: **Developing Industrial Multi-Agent Systems.** In: First International Conference on Multi-Agent Systems (ICMAS) / Lesser, V. (Hrsg.), The MIT Press, 1995, S.423–430.

Jen 98a Jennings, N. R.; Sycara, K. und Wooldridge, M.: **A Roadmap of Agent Research and Development.** In: Autonomous Agents and Multi-Agent Systems Journal, Jennings, N.R.; Sycara, K.; Georgeff, M. (Hrsg.), Luwer academic Publishers, Boston, Vol. 1, Issue 1, 1998, S. 7–38.

Jen 98b Jennings, N. R.; Sycara, K.; Wooldridge, M.: **A Roadmap of Agent Research and Development.** In: Autonomous Agents and Multi-Agent Systems (1998) Nr. 1, S. 275–306.

Jon 80 Jones, A. K.; Schwarz, P.: **Experience Using Multiprocessor Systems – Status Report.** COMP-SURV 12 (1980) Nr. 2, S. 121–165.

Kam 97 Kamp, G.; Neumann, B.: **Knowledge-based Inference Methods for Modelling Technical Systems.** In: Proceedings of the 30th Hawaiin International Conference on Systems Sciences in Wailea, Hawaii, USA. Computer Society Press, 1997.

Kar 99 Kargl, F.; Illmann, T. ; Weber, M.: **Evaluation of Java Messaging Middleware as a Platform for Software Agent Communication.** In: JIT 99, Java-Informations-Tage 1999 in Düsseldorf, Germany: 1999, S. 161–170.

Kar 02 Karvounarakis, G.; Alexaki, S.; u.a.: **RQL: A Declarative Query Language for RDF.** In: The 11th Intl. World Wide Web Conference (WWW 2002), November 13, 2001 in Honolulu, Hawaii, USA. ACM Press, 2002, S. 592–603.

Kat 99 Katzenbach, A.: **International Engineering Processes of the Future.** In: Wissensmanagement und schnelle Produktenwicklung. Trends-Methoden-Vorgehensweisen / Bullinger, H.-J. (Hrsg.). 1999, S. 47–66.

Khu 98 Khurana, A.: **Towards Holistic Front Ends in New Product Development.** In: Journal of Product Innovation Management 15 (1998), S. 57–74.

KI 01 **Moderne Methoden der Künstlichen Intelligenz.** URL: http://www.ki.informatik.hu-berlin.de/lehre/ss01/MMKIVL.shtml, 2001. (07.01.2005)

KIF 94 **KIF: Knowledge Interchange Format.** URL: http://www.cs.umbc.edu/kse/kif/, 1994. (07.01.2005)

Kir 98 Kirn, S.; Gasser, L.: **Organizational Approaches to Coordination in Multi-Agent Systems.** In: Arbeitsbericht (1998) Nr. 9.

Kla 99 Klarlund, N.; Moller, A.; Schwartzbach, M. I.: **Document Structure Description 1.0.** URL: http://www.brics.dk/DSD/, 1999. (07.01.2005)

Kla 00 Klarlund, N.; Moller, A.; Schwartzbach, M. I.: **DSD: A Schema Language for XML.** In: ACM SIGSOFT Workshop on Formal Methods in Software Practice. Portland, OR: 2000, S. 101–111.

Kno 01 Knoblock, C. A.; Minton, S.; u.a.: **The Ariadne Approach to Web-based Information Integration.** In: International Journal of Cooperative Information Systems 10 (2001) Nr. 1-2, S. 145–169.

Kon 86 Konolige, K.: **A Deduction Model of Belief.** Pitman Publishing: London and Morgan Kaufmann: San Mateo, CA, 1986.

Lab 94 Labrou, Y.; Finin, T.: **A semantics approach for KQML – a general purpose communication language for software agents.** In: Proceedings of the 3rd international conference on Information and knowledge management in Gaithersburg, Maryland, USA. ACM Press, 1994, S. 447–455.

Lab 97 Labrou, Y.; Finin, T.: **A Proposal for a new KQML Specification.** URL: http://www.cs.umbc.edu/kqml/papers/kqml97.pdf, 1997. (07.01.2005)

Lab 99 Labrou, Y.; Finin, T.; Peng, Y.: **The current landscape of Agent Communication Languages.** In: IEEE Intelligent Systems 14 (1999) Nr. 2, S. 45–52.

Lan 80 Langsford, A.; Moffett, J. D.: **Distributed Systems Management.** Addison-Wesley, 1980.

Las 99 Lassila, O.; Swick, R.: **Resource Description Framework (RDF) Model and Syntax Specification.** W3C Recommendation. Technical Report. URL: http://www.w3.org/TR/1999/REC-rdf-syntax-19990222/, Feb. 1999. (07.01.2005)

LeB 00 Le Bodic, G.; Girma, D.; u.a.: **An Agent-Based Middleware for Enhancing Mobile Communications Infrastructure and Provision of Services in Emerging Systems.** In: Proceedings SOMAS Workshop in Milton Keynes, UK. 2000.

Ley 00 Leyh, J.; Diederich, M. K.: **Evolutionary Planning Methodology for the Early Phases of Product Development.** In: Proceedings for the Conference on Rapid Prototyping in the Automotive Industries, 33rd International Symposium on Automotive Technology and Automation (33rd ISATA), September 25–27, 2000 in Dublin, Ireland. 2000.

Ley 01 Leyh, J.; Diederich, M. K.: **Distributed and Incremental Planning in the Rapid Product Development.** In: Proceedings for the Conference on Product Development, 1st Automotive and Transportation Technology Congress and Exhibition (1st ATTCE) October 1–4, 2001 in Barcelona, Spain. 2001.

Lia 00 Liao, T.: **Light-weight reliable Multicast Protocol.** In: INRIA, Rocquencourt, BP 105. Le Chesnay Cedex, France: 2000. URL: http://webcanal.inria.fr/lrmp/ (07.01.2005)

Lie 01 Lieberman, H.; Nardi, B. A.; Wright, D.: **Training Agents to Recognize Text by Example.** In: Autonomous Agents and Multi Agent Systems 4 (2001) Nr. 1-2, S. 79–92.

Liu 01 Liu, M.; Ling, T. W.: **A Rule-based Query Language for HTML.** In: Proceedings of the 7th International conference on database Systems for Advanced Applications (DASFAA ’01), April 18–21, 2001 in Hong Kong, China. IEEE-CS Press, 2001, S. 6–13.

Liu 02 Liu, M.; Ling, T. W.: **Towards Declarative XML Querying.** In: The 3rd International Conference on Web Information Systems Engineering (WISE ’00). 2002, S. 127–138.

Luc 03a **Lucent Technologies:** URL: http://www.galaxquery.org/, 2003. (07.01.2005)

Luc 03b Luck, M.; McBurney, P.; Preist, C.: **Agent Technology: Enabling Next Generation Computing.** AgentLink, 2003.

Mal 03 Malhotra, A.; Melton, J.; Walsh, N.: **XQuery 1.0 and XPath 2.0 Functions and Operators.** URL: http://www.w3.org/TR/2003/WD-xpath-functions-20031112/, Nov. 2003. (07.01.2005)

Mar 99 Martin, D. L.; Cheyer, A. J.; Moran, D. B.: **The Open Agent Architecture: A Framework for Building Distributed Software Systems.** In: Applied Artificial Intelligence 13 (1999) Nr. 1-2, S. 91–128.

Mat 95 Maturana, F. P.; Norrie, D. H.: **A Multi-Agent Coordination Architecture for Distributed Organizational Systems.** In: The Mediator Research Program. 1995.

Mil 00 Miloslav, N.: **Schematron Tutorial.** URL: http://www.zvon.org/HTMLonly/SchematronTutorial/General/contents.html, 2000. (07.01.2005)

Mit 97 Mitchell, T. M.: **Machine Learning.** McGraw-Hill, Science/Engineering/Math, 1997. ISBN: 0070428077

Moe 03 Moeller, A.: **XML-Tutorial: FLWR expressions.** URL: http://www.brics.dk/~amoeller/XML/querying/flwrexp.html, 2003. (07.01.2005)

Moe 00 Moerkotte, G.: **YAXQL: A powerful and web-aware query language supporting query reuse.** In: Fakultät für Mathematik und Informatik der Universität Mannheim, Reihe Informatik (2000) Nr. 16. S. 52–54.

Moo 93 Moore, S. A.: **Saying and doing: uses of a formal language in the conduct of business.** Pennsylvania, Univ., Diss., 1993.

Mül 95 Müller, J. P.; Pischel, M. und Thiel, M.: **Modelling reactive behaviour in vertically layered agent architectures.** In: Intelligent Agents: Theories, Architectures and Languages (LNAI Volume 890), Wooldridge, M.; Jennings, N. R. (Hrsg.), Springer Verlag: Berlin, 1995, S. 261–276.

Mur 00 Murata, M.: **RELAX (Regular Language description for XML).** URL: http://www.xml.gr.jp/relax/, 2000. (07.01.2005)

Myl 96 Mylopoulos, J.; Chaudhri, V.; u.a.: **Building knowledge base management systems.** In: The VLDB Journal 5 (1996), S. 238–263.

Nav 01 Navarro, F.; Jones, K.; u.a.: **An Agent-Based Service Brokering Architecture for Multiservice Next-Generation Networks.** In: FUJITSU Sci.Tech.J. (2001) Nr. 37, S. 97–108.

Nav 97 Navarro, G.; Baeza-Yates, R.: **Proximal nodes: a model to query document databases by content and structure.** In: ACM Transactions on Information Systems 15 (1997) Nr. 4, S. 400–435.

Nii 86 Nii, H. P.: **Blackboard Systems: The Blackboard Model of Problem Solving and the Evolution of Blackboard Architectures.** In: The AI Magazine, 1986, S. 39–53.

Nwa 94 Nwana, H. S.: **Negotiation Strategies: An Overview.** In: BT Laboratories internal report (1994).

Nwa 96a Nwana, H. S.: **Software Agents: An Overview.** In: Knowledge Engineering Review 11 (1996) Nr. 3, S. 205–244.

Nwa 96b Nwana, H. S.; Lee, L.; Jennings, N. R.: **Co-ordination in software agent systems.** In: BT Technol J 14 (1996) Nr. 4, S. 79–88.

OGr 90 O'Grady, P; Leer, K. H.: **A Hybrid Actor and Blackboard Approach to manufacturing Cell Control.** In: Journal of Intelligent and Robotic Systems vol. 3, Kluwer Academic Publishers, The Netherlands, 1990, S. 67–72.

OHa 96 O'Hare, G. M.; Jennings, N. R. (Hrsg.): **Agent factory: An Environment for the Fabrication of Multiagent Systems.** In: Foundations of Distributed Artificial Intelligence. John Wiley & Sons, Inc. 1996, S. 449–484. ISBN: 1-58113-480-0

OMG 03 Object Management Group, Inc.: **OMG Unified Modelling Language Specification.** URL: http://www.omg.org/technology/documents/formal/uml.htm, 2003. (07.01.2005)

Opl 03 Opletal, S.: **Entwicklung und Implementierung eines Retrieval-Agenten für den Einsatz im Rapid Product Development.** Studienarbeit (1895), Universität Stuttgart, 2003.

Ora 03 Oracle Corporation: **Oracle XQuery Prototype.** URL: http://otn.oracle.com/sample_code/tech/xml/xmldb/xquery_readme.html, 2003. (07.01.2005)

Ora 04a Oracle Corporation: **OJXQI - The Oracle Java XQuery API.** URL: http://otn.oracle.com/sample_code/tech/xml/xmldb/jxqi.html, 2004. (07.01.2005)

Ora 04b Oracle Corporation: **Using XQuery from the command line.** URL: http://otn.oracle.com/sample_code/tech/xml/xmldb/xquery_context.html, 2004. (07.01.2005)

Özs 95 Özsu, M.; Blakeley, J.: **Query Processing in Object-Oriented Database System.** In: Modern database systems: the object model, interoperability, and beyond / Kim, W. (Hrsg.). ACM Press, 1995, S. 146–174.

Pad 02a Padgham, L.; Winikoff, M.: **Prometheus: A Methodology for Developing Intelligent Agents.** In: 3rd International Workshop on Agent-Oriented Software Engineering held at Autonomous Agents & Multi-Agent Systems (AAMAS 2002). Faculty of Engineering. Bologna, Italy: 2002, S. 37–38.

Pad 02b Padgham, L.; Winikoff, M.: **Prometheus: A Pragmatic Methodology for Engineering Intelligent Agents.** In: Proceedings of the workshop on Agent-oriented methodologies at OOPSLA / Debenham, J.; Henderson-Sellers, B.; u.a. (Hrsg.). Seattle, USA: 2002, S. 97–108.

Pao 01 Paolucci, M.; Sycara, K.: **An Exploration in MAS Scalability.** URL: http://www-2.cs.cmu.edu/~softagents/papers/paolucci.pdf, 2001. (07.01.2005)

Pap 95 Papadimitriou, C. H.: **Database Metatheory: Asking the Big Queries.** In: Proceedings of the 14th ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems (PODS '95) in New York, NY. ACM Press, 1995, S. 1–10.

Pap 03 Papakonstantinou, Y.; Borkar, V.; u.a.: **XML Queries and Algebra in the Enosys Integration Platform.** In: Data & Knowledge Engineering Volume 44 (2003) Issue 3 Special issue: Data integration over the Web S. 299–322.

Par 94 Park, H.: **An Agent-Based Approach to Concurrent Cable Harness Design.** In: AIEDAM 8 (1994) Nr.1, S. 45–61.

Pin 89 Pin, J. E. (Ed.); **Formal properties of finite automata and applications.** Berlin: Springer, 1989. ISBN: 3-540-51631-X

Rad 98 Radeke, E.; Seifert, L.: **GENIAL: Enabling Intelligent Access to Internal and External Engineering Information.** In: CAD ’98 – Tele-CAD: Produktentwicklung in Netzwerken in Darmstadt, Germany / Anderl, R.; Encarnacao, J.; Rix, J. (Hrsg.). 1998, S. 75–84.

Ree 98 Reed, C.: **Dialogue Frames in Agent Communication.** In: International Conference on Multi Agent Systems (ICMAS) in Paris, France / Werner, B. (Hrsg.). IEEE Computer Society, 1998, S. 246–253.

Ret 03 **RETSINA Agent Architecture.** URL: http://www-2.cs.cmu.edu/~softagents/retsina.html (07.01.2005)

RFC 793 RFC 793: **Transmission Control Protocol.** URL: http://www.ietf.org/rfc/rfc0793.txt?number=793. (07.01.2005)

RFC 1122 RFC 1122: **Requirements for Internet Hosts – Communication Layers.** URL: http://www.ietf.org/rfc/rfc1122.txt?number=1122. (07.01.2005)

RFC 1323 RFC 1323: **TCP Extensions for High Performance.** URL: http://www.ietf.org/rfc/rfc1323.txt?number=1323. (07.01.2005)

Rob 99 Robie, J. (Ed.): **XQL (XML Query Language).** URL: http://www.ibiblio.org/xql/xql-proposal.html, 1999. (07.01.2005)

Rod 00 Roddick, J. F.; Al-Jadir, L.; u.a.: **Evolution and Change in Data Management – Issues and Directions.** In: SIGMOD Record 29 (2000) Nr. 1, S. 21–25. URL: http://dblab.ewha.ac.kr/hsyong/teach/dbgrad/course2000-2/paperlist/roddick.pdf (07.01.2005)

Rol 02a Roller, D.; Mesina, M.; Dalakakis, S.: **An Active Semantic Network for Rapid Prototyping.** In: Proceedings of the 5th Conference "Informatics and Algorithms 2002" in Kosice, Slovakei. 2002, S. 47–55.

Rol 02b Roller, D.; Eck, O.; Dalakakis, S.: **A Knowledge Base for Rapid Product Development.** In: TMCE, Horvath, I.; Peigen, L.; u.a. (Hrsg.), 2002.

Ros 94 Rosenschein, J.; Zlotkin, G.: **Rules of Encounter – Designing Conventions for Automated Negotiation among Computers.** Cambridge: The MIT Press, 1994. ISBN: 0262181592

Rum 99 Rumbaugh, J.; Jacobsen, I.; Booch, G.: **The Unified Modelling Language Reference Manual.** Addison-Wesley, 1999.

Rus 95 Russell, S. J.; Norvik, P.: **Artificial Intelligence: A Modern Approach.** Prentice Hall, Englewood Cliffs, N.J., 1995.

Sav 99 Savnik, I.; Tari, Z.; Mohoric, T.: **QAL: A Query Algebra of Complex Objects.** In: Data & Knowledge Engineering 30 (1999) Nr. 1, S. 57–94.

Sch 98a Schäfer, K.: **Development and Evaluation of Methodologies and Tools Dedicated to the Implementation of the Multi-Agent Paradigm in Shop Floor Control.** Karlsruhe, Univ., Diss., 1998.

Sch 98b Schmidt, A.; Specker, A.; u.a.: **An Agent-based Telecooperation Framework.** In: Lecture Notes in Computer Science, 1370. Darmstadt, Germany: Springer Heidelberg, 1998, S. 122–129.

Sch 98c Schmidt, A.; Specker, A.; u.a.: **Agents, Brokers, Traders, and Services in Cooperative Systems.** In: Proceedings of the 3rd International Conference on the Design of Cooperative Systems, COOP '98. 1998.

Shi 95 Shirazi, B. A.; Hurson, A. R.; Kavin, K. M. (Hrsg.): **Scheduling and load balancing in parallel and distributed systems.** IEEE Computer Society Press, 1995. ISBN: 0818665874

Sho 97 Shoham, Y.: **An Overview on Agentoriented Programming.** Software Agents, Bradshaw, J. M. (Hrsg.), Menlo Park, Calif.; AAAI Press, 1997, S. 271–290.

Smi 80 Smith, R. G.: **Contract Net Protocol: High Level Communication and Control in a Distributed Problem Solver.** In: IEEE Transaction on Computers 29 (1980), S. 1104–1130.

Spa 02 Spath, D.; Sieth, T.; Kuhn, R.: **Vorteile objektorientierter Datenhaltung in der Technologieplanung.** In: Sonderforschungsbereich Rechnerintegrierte Konstruktion und Fertigung von Bauteilen <Karlsruhe>: Rechnerunterstützte Produktentwicklung und -herstellung : auf Basis eines integrierten Produkt- und Produktionsmodells, Grabowski, H. (Hrsg.); ausgewählte Beiträge des Sonderforschungsbereichs 346 der Deutschen Forschungsgemeinschaft (DFG). Aachen: Shaker, 2002, S. 466–470.

Spa 03a Spath, D.; Korge, A.; Scholtz, O.: **Ganzheitliche Produktionssysteme: eine neue Chance für produzierende Unternehmen.** In: Ratio 9 (2003), Nr. 3, S. 9–11.

Spa 03b Spath, D.; Gerlach, S.: **Informationen für Teams in Montage und Produktion: Visualisierung, Kommunikation, Know-how-Dokumentation.** In: wt Werkstattstechnik online 93 (2003), 9, S. 624–626.

SQL 93 Date, C. J.; Darwen, H.: **A Guide to SQL Standard.** 3rd Edition, Addison-Wesley, 1993. ISBN: 0201964260

Sti 98 Stiefbold, O.: **Konzeption eines reaktionsschnellen Planungssystems für Logistikketten auf Basis von Software-Agenten.** Karlsruhe, Univ, Diss. 1998.

Suc 01 Suciu, D.: **On Database Theory and XML.** In: SIGMOD Record September 2001 30 (2001) Nr. 3, S. 39–45.

Suk 99 Sukthankar, R.; Stockton, R. G.: **ARGUS: An Automated Multi-Agent Visitor Identification System.** In: American Association for Artificial Intelligence in Orlando, Florida, USA. American Association for Artificial Intelligence, 1999, S. 208–213.

Sun 00 Sun Microsystems Inc: **Java Shared Data Toolkit.** URL: http://java.sun.com/products/java-media/jsdt/, 2000. (07.01.2005)

Sun 03 Sun Microsystems Inc: **JXTA.** URL: http://services.jxta.org/servlets/ProjectHome/, 2003. (07.01.2005)

Swi 01 **SwiftMQ.** URL: http://www.swiftmq.com/products/index.html, 2001. (07.01.2005)

Syc 01 Sycara, K.; Paolucci, M.; u.a.: **The RETSINA MAS Infrastructure.** In: Tech. Report CMU-RI-TR-01-05, Robotics Institute, Carnegie Mellon University. Netherland: Kluwer Academic Publishers, 2001, S. 29–48.

Tan 94 Tanenbaum, A. S.: **Moderne Betriebssysteme.** Hanser, 1994.

Tan 96 Tanenbaum, A. S.: **Computer Networks – Third Edition.** New Jersey, USA: Prentice-Hall, 1996. URL http://www.opnet.com/services/university/tanenbaum.html (07.01.2005)

Tan 02 Tanenbaum, A. S.; van Steen, M.: **Distributed Systems: Principles and Paradigms.** Prentice-Hall, 2002.

Tho 04 Thompson, H. S.; Beech, D.; u.a. (Hrsg.): **XML Schema Part 1: Structures.** URL: http://www.w3.org/TR/xmlschema-1/, Oct. 2004.

Tim 98 Timm, I. J.: **Multi-Agentensysteme zur Unterstützung ökologischer Transportlogistik.** In: Umweltinformatik '98 – Vernetzte Strukturen in Informatik, Umwelt und Wirtschaft – 12. Internationales Symposium "Informatik für den Umweltschutz" der Gesellschaft für Informatik (GI), Band I / Haasis, H.-D.; Ranze, K.C. (Hrsg.). 1998, S. 293–303.

Tim 99 Timm, I. J.; Knirsch, P.: **Ökologische Optimierung in der verteilten Tourenplanung durch Multi-Agentensysteme.** In: Proceedings des Workshops "Agententechnologie" auf der KI '99: Agententechnologie – Multiagentensysteme in der Informationslogistik und wirtschaftswissenschaftliche Perspektiven der Agenten-Konzeptionalisierung, 14./15. September 1999 in Bonn / Timm, I. J.; Knirsch, P.; u.a. (Hrsg.). Bremen: 1999, S. 73–86.

Ube 02 **UberMQ.** URL: http://www.ubermq.com/, 2002. (07.01.2005)

Van 00 Van Dyke Parunak, H.: **A Practitioner's Review of Industrial Agent Applications.** In: Autonomous Agents and Multi Agent Systems, Klausen Academic Publishers 3 (2000) Nr. 4, S. 389–407.

Vid 03 Vidal, J. M.; Durfee, E. H.: **Predicting the Expected Behavior of Agents that Learn About Agents: The CLRI Framework.** In: Autonomous Agents and Multiagents Systems 6 (2003), Nr. 1 S. 77–107.

War 00 Warschat, J.; Prieto, J.; u. a.: **Ein integriertes Konzept für die evolutionär-iterative Produktenwicklung.** In: Proceedings of the International Conference for R&D Management 1999 in Stuttgart / Bürgel, H. D. (Hrsg.). Stuttgart, 2000.

War 02 Warschat, J.; Cebulla, T.; u.a.: **Systemumgebung einer multidisziplinär wissensbasierten Produktentwicklung.** In: Modelle, Werkzeuge und Infrastrukturen zur Unterstützung von Entwicklungsprozessen, 20.–22. März 2002 in Aachen / Nagl, M.; Westfechtel, B. (Hrsg.). Aachen: Wiley-VCH, 2002, S. 293–307. ISBN 3-527-27769-2.

Wel 95 Wellmann, M. P.: **A Computational Market Model for Distributed Configuration Design.** In: AI EDAM, 1995, S.125–133.

Wes 94 Westkämper, E.; Laucht, O.: **Dezentralität als Basisprinzip zeitgemäßer Unternehmensorganisation – Teil 1: Gestaltungsregeln und Strukturen.** In: wt 84 (1994) Nr. 5, S. 421–425.

Wes 02 Westkämper, E.; **Stuttgart Model of Virtual Enterprise.** In: Proceedings of the 35th CIRP International Seminar on Manufacturing Systems, Seoul, Korea, 2002, S. 25–31.

Woo 95 Wooldridge, M.; Jennings, N. R.: **Intelligent Agents: Theory and Practice.** In: The Knowledge Engineering Review 10 (1995) Nr. 2, S.114–152.

Woo 99 Wooldridge, M.: **Intelligent Agents.** In: Multiagent Systems – A Modern Approach to Distributed Artificial Intelligence / Weiss, G. (Hrsg.). Cambridge, Massachusetts; London, England: The MIT Press, 1999, S. 27–78.

Zap 00 Zapf, M.; Herrmann, K.; Geihs, K.: **Kommunikationsmechanismen im Agentensystem AMETAS.** URL: http://wwwspies.informatik.tu-muenchen.de/gifgbs/Versand0010/konferenz/gi/programm/MichaelZapf.pdf, 2000. (07.01.2005)

Zic 01 Zice, S. Dr.; Zhengping, L.; u.a.: **Agent-based Logistics Coordination and Collaboration.** In: SIMTech Technical Report (AT/01/011/LCI), 2001.

Anhang

A Multiagentensysteme

Die Funktionalität von Multiagentensystemen ist in mehreren Forschungsprojekten getestet worden. Eine Projektübersicht der bis zum Jahr 1996 ausgeführten Projekte befindet sich in O'Hare [OHa 96]. Ein zusammenfassender Überblick über industrielle Anwendungen von Agenten ist in Van Dyke Parunak [Van 00] gegeben, wie z. B. über das Holonic Manufacturing System Projekt (HMS). Im HMS besteht das grundlegende Element aus einem Kontrollagenten und einer physischen Maschine. Dadurch wird eine autonome mechatronische Einheit gebildet, die mit anderen Einheiten kooperiert, um auf höheren Ebenen Einheiten zu bilden.

Interessante Ansätze bereits entwickelter Multiagentensysteme liefern die Projekte *Agent Building and Learning Environment* (ABLE) [Abl 04] insbesondere im Bereich der Lernstrategien und *Java Agent Development Framework* (JADE) [Jad 03] im Bereich des Runtime-Systemservices. In RETSINA [Ret 03] wird eine Peer-To-Peer-Kommunikation eingesetzt und das Projekt FIPA-OS [FIP 03] zeichnet sich insbesondere durch Konfigurationsansätze aus.

Agent Building and Learning Environment (ABLE)

Ein richtungweisendes Projekt ist ABLE von IBM, das stark durch den Bereich der Computational Intelligence [Abl 04] [Big 02] beeinflusst wurde. ABLE besteht aus einem Java-Framework, einer Komponenten-Library und aus einem Werkzeugsatz, der durch die Nutzung von Techniken aus dem Maschinellen Lernen und Reasoning das Herstellen und Benutzen von hybriden und intelligenten Agenten ermöglicht. Es werden nicht nur mehrfaches Reasoning und Lernmethoden eingesetzt, sondern auch Optimierungs-Algorithmen, die ähnlich aufgebaut sind wie JavaBeans, und AbleBeans genannt werden. Solche Beans z. B. Data-Beans, Learning-Beans, Rule-Beans sowie Function-specific AbleAgents, bieten eine flexible Agenten-Plattform. Die Kommunikation zwischen den Agenten ist ereignisgesteuert und es existiert zudem eine Regel-Sprache für den Ablauf der Kommunikation. Zusätzliche Features, wie z. B. die FIPA-Konformität und Editoren mit graphischem User-Interface, steigern die Anwenderfreundlichkeit des Frameworks.

Java Agent Development Framework (JADE)

Auch das Projekt JADE baut auf den von der FIPA entwickelten Spezifikationen auf [Jad 03]. Die gesamte Plattform bietet Runtime-Systemservices für Agenten an. Komponenten, wie das *Agent Management System* und der *Agent Communication Channel* regeln den Transport von ACL-Nachrichten. Die internen Interaktionen finden durch *light-weight* Kommunikation auf der Ebene der Java Objekte statt. Ana-

log zu einem Applikationsserver, der den Container von Objekten repräsentiert, ist der Agenten-Container innerhalb der Agenten-Plattform organisiert. Dabei wird von einem Server-Objekt der gesamte Lebenszyklus vom Entstehen bis zum Entfernen aller Agenten koordiniert. Das Autonomie-Merkmal der Agenten wird über ein Modell dargestellt, das sich auf eine Verhaltensabstraktion für alle möglichen Agentenzustände stützt. Im Normalfall werden verschiedene, vorbereitete Verhaltensmuster, abhängig von dem Agentenbefehl, abgearbeitet. Es ist auch ein graphisches User-Interface für die Remote-Agentenverwaltung vorhanden.

Reusable Environment for Task-Structured Intelligent Networked Agents (RETSINA)

Im Projekt *Reusable Environment for Task-Structured Intelligent Networked Agents* (RETSINA) wird ein Framework für Multiagentensysteme [Ret 03], [Syc 01], [Pao 01] aufgebaut. Die Agenten werden hier gruppiert und kommunizieren über Peer-to-Peer Kommunikationswege. Dabei wird es vermieden, eine zentrale Kontrollkomponente innerhalb des Agentensystems zu installieren, die das Verwalten übernimmt. Vielmehr werden verteilte Infrastrukturservices implementiert, die die Interaktionen zwischen den Agenten erleichtern. In Giampapa [Gia 00] wird ein Verfahren beschrieben, wie an das RETSINA-Agentensystem andere Agentensysteme angeschlossen werden können.

FIPA-OS

FIPA-OS ist ein mit den FIPA Spezifikationen übereinstimmendes Open Source Projekt [FIP 03]. Durch das komponentenorientierte Agentenwerkzeug werden die zwingend erforderlichen FIPA-gerechten Komponenten bereitgestellt. Der Entwurf eines Multiagentensystems startet mit der Spezifikation der Architektur und der Komponenten sowie mit der Serviceschnittstelle und der Implementierung.

A.1 Weitere Agentensysteme und ihre Anwendungsgebiete

Weiterführende Entwicklungen der Agententechnologie kommen zum größten Teil aus den Bereichen der Künstlichen Intelligenz, wie z. B. aus dem Teilgebiet des *Maschinellen Lernens*. Obwohl Maschinelles Lernen als Fächer übergreifender wissenschaftlicher Bereich sich erst behaupten muss, würden in dem Fall einer interdisziplinären Akzeptanz, dessen Auswirkungen von grundlegender Bedeutung sein [Mit 97]. In dem Projekt ARIADNE [Kno 01] wurden mehrere Techniken entwickelt, die das Lernen durch das Erkennen des Inhalts einer angefragten Information ermöglichen.

Liebermann [Lie 01] stellt eine Methode zur Verfügung, die es ermöglicht, Agenten zu trainieren, die auf der Basis von Beispieltexten Textblöcke erkennen können. Die für den Parser benötigten Regeln werden durch einen iterativen Prozess erstellt, der am Anfang heuristische Methoden für das Parsen von Beispielen nutzt und

dann eine Menge von Hypothesen aufstellt. Am Ende werden die Ergebnisse von den Anwendern beurteilt.

In Brauer [Bra 98] wird anhand einer verteilten Struktur von Maschinen gelernt, die Arbeitspläne nach Kosten und Zeiteffektivität abarbeiten. Dabei lernen die Maschinen durch jede Iteration und passen ihren Arbeitsplan an.

In Ferber [Fer 98] wird ein konzeptionelles Modell eines Agentensystems (AALAADIN) beschrieben. In dem Modell wird ein Gruppenkonzept aufgebaut. Die Agenten werden dabei einzelnen oder mehreren Gruppen zugeteilt, die Regeln enthalten, die der Agent ausführen muss. Die Regeln stellen die Funktionalität des Agenten dar, der somit regelbasiert aufgebaut ist. Als Koordinationsprotokoll wird das Contract Net Protocol verwendet.

In Vidal [Vid 03] wird ein Weg aufgezeigt, der es erlaubt in jedem einzelnen rekursiven Lernschritt die Funktionalität der Agenten zu steigern. Dabei liegt der Fokus nicht auf der statischen Modellierung des Verhaltens der Agenten im Multiagentensystem, es wird vielmehr ein Verfahren entworfen, das es dem Agenten erlaubt sein Verhalten während seiner Ausführung durch Lernen aus Fehlern anzupassen.

In Padgham [Pad 02a], [Pad 02b] wird das Projekt Prometheus vorgestellt. In diesem wird eine Vorgehensweise definiert, wie ein Multiagentensystem aufgebaut werden kann. Dieser Ansatz wird anhand eines fiktiven Agentensystems zur Unterstützung von Einkaufsvorgängen inklusive der finanziellen Abwicklung über Banken erläutert.

Eine Art Middleware für den Internetnutzer wurde in Schmidt [Sch 98b] und [Sch 98c] entwickelt. Dabei wird eine dreischichtige Architektur aufgebaut, die auf der untersten Ebene die Services wie beispielsweise Workflow-, Team-, Travel-Service anbietet. Die nächst höhere Schicht besteht aus einem oder mehreren Verzeichnisdiensten bzw. Brokern, die Kommunikations- und Adressdaten verwalten und damit den Zugriff auf die Services sicherstellen. Die oberste Ebene, die Agentenebene beinhaltet einen Agentencluster, der den Benutzer bei der Nutzung der angebotenen Dienstleistungen unterstützt. Die Agenten sind dabei lernfähig und versuchen die Vorlieben der Benutzer durch das Protokollieren der Vorgehensweisen der Benutzer zu erkennen und bei weiteren Dienstleistungsanfragen zu berücksichtigen.

In Sukthankar [Suk 99] wird das Multiagentensystem *Argus* vorgestellt, mit dessen Hilfe Gesichtererkennung durchgeführt wird. Dabei existieren im ARGUS-System Detector- und Recognizer-Agenten für die Bilderkennung, sowie Delegator-Agenten für die Informationsverteilung und Notifier-Agenten, die die Schnittstelle zu den Anwendungen darstellen.

Die in Aldunate [Ald 02] vorgestellte agentenbasierte Middleware unterstützt die spontane Zusammenarbeit von verteilten Menschen, die sich nicht zwingend ken-

nen müssen. Um eine Kommunikation zwischen den Teilnehmern durchführen zu können, wird jeder Teilnehmer mit zwei Agenten ausgestattet. Der eine Agent repräsentiert das Profil des Nutzers wodurch dieser charakterisiert wird. Dabei werden neben seinen Bedürfnissen auch seine Möglichkeiten spezifiziert. Der zweite Agent ist ein technischer Agent, der die technischen Randbedingungen, wie die Übertragung einer Information von einem Teilnehmer zu einem anderen Teilnehmer übernimmt.

Ein Workflow-System für Web-Services, unterstützt durch die Agententechnologie, wurde in Blake [Bla 02] entwickelt. Die Architektur stellt Basis Agententypen zur Verfügung, wie einen Site-Manager-Agent, der die angebotenen Dienstleistungen verwaltet, und einen Global-Workflow-Manager-Agent, der den Ablauf der Dienstleistungsbenutzung koordiniert. Der Global-Workflow-Manager-Agent wird durch den auf das Aufgabengebiet angepassten Workflow-Manager-Agent, der wiederum Role-Manager-Agents verwaltet, die durch Konfiguration aus dem Site-Manager-Agent hervorgegangen sind, unterstützt. Die Role-Manager-Agents steuern dann die angebotenen Dienstleistungen.

Für den Bereich der Logistik wurde in Zice [Zic 01] eine agentenbasierte Koordination und Zusammenarbeit entwickelt. Dabei werden die Einkaufs-, Verkaufs- und Lager- sowie Transportvorgänge durch den Einsatz von Agenten unterstützt.

A.2 Mobile Agententechnologien

Im Projekt MARS [Cab 98a], [Cab 98b] wird eine Architektur entwickelt, um mobile Agenten im Internet betreiben zu können. Hierzu wird eine Serverinstanz in Java implementiert, die an jedem Knoten im Internet vorhanden sein muss, auf der die mobilen Agenten zum Einsatz kommen sollen. Ein mögliches Einsatzgebiet ist eCommerce. Wobei der Einkäufer-Agent zu den Anbietern migriert, um dort Angebotsanfragen in Empfang zu nehmen und entsprechend zu bewerten.

Chen [Che 99] definiert die Randbedingung für ein Operation-System für mobile Agenten, das AgentOS. Ziel des AgentOS ist es mobile User im Internet bei Ihrer Arbeit derart zu unterstützen, dass sie beispielsweise überall auf der Welt dieselbe Sicht auf ihre Daten und Anwendungen haben, unabhängig davon ob das eigene Notebook oder ein fremdes Internetterminal benutzt wird.

In Fischmeister [Fis 99] wird ein Verfahren vorgestellt, das es erlaubt mit Hilfe einer zentralen Instanz, in diesem Fall Supervisor genannt, mobile Agenten zu koordinieren. Dabei wird der Grundgedanke des Client/Server – Ansatzes weiterverfolgt. Die Supervisor Instanz nimmt die Anfragen an und verteilt die Anfragen auf verschiedene mobile Agenten, die dann die Probleme lösen.

Ein weiteres Agentensystem, das den Einsatz mobiler Agenten erlaubt, ist AMETAS [Zap 00]. AMETAS stellt ein Framework zur Verfügung, in dem die selbst entwickelten mobilen Agenten eingebettet werden. Das Hauptkriterium von

AMETAS liegt auf der Sicherheit in der Agentenkommunikation bei mobilen Agenten. Hierfür wurde ein Authentifizierungsmechanismus entwickelt um die mobilen Agenten zu identifizieren und die Ressourcen zu reglementieren.

Die in Le Bodic [LeB 00] beschriebene agentenbasierte Middleware hat die Aufgabe Quality-of-service Eigenschaften in einem drahtlosen Netz anzubieten und zu überwachen. Dabei besitzen die mobilen Teilnehmer einen Agenten, der über den Marktmechanismus einen Vertrag mit den stationären Agenten auf Seiten des Netzanbieters aushandelt. Die ausgehandelte Bandbreite wird durch die stationären Agenten zur Verfügung gestellt und überwacht. Dabei wird auch das Roaming der mobilen Teilnehmer berücksichtigt.

Zur Überwachung von mobilen Agenten wurde in Gittens [Git 01] ein Überwachungsagent Agent-Monitoring-Agent (AMA) entwickelt. Der AMA kann in einer JINI-Umgebung eingesetzt werden. Die sich im Agentensystem befindlichen Agenten melden sich beim AMA mit ihren Adressinformation wie beispielsweise die Portnummer an. Dadurch kann der AMA den Weg eines mobilen Agenten durch das Netz verfolgen.

Im Rahmen des Projekts EMBASSI [Din 00] wurde eine agentenbasierte Architektur entwickelt, die es ermöglicht mobile Dienstleistungen zu benutzen unabhängig vom eingesetzten Gerät. Die Architektur berücksichtigt zwei Ebenen, zum einen die Applikationsebene, die aus den dienstspezifischen Agenten besteht, und der Systemebene, die Infrastrukturleistungen wie das Überwachen und Kontrollieren von Ressourcen und das Planen von Multiagenten-Aktionen bereitstellt.

B Programmiertechniken und Darstellungshilfsmittel

Dieser Abschnitt dient als kurze Einführung in die Bedeutung der verwendeten Diagrammarten und Programmiertechniken. Zuerst wird das statische Klassendiagramm und das dynamische Sequenzdiagramm der objektorientierten Programmierung vorgestellt. Durch das Zustandübergangsdiagramm werden die einzelnen Kommunikationsprotokolle für die Kommunikation mit der RPD-Middleware spezifiziert. Dieser Diagrammtyp bildet auch die Grundlage für den Koordinations-Agenten. Die erweiterte Backus-Naur-Form (EBNF) dient zur Beschreibung der Anfragesprachen der Agenten und die XML-Notationen bilden die Grundlagen für die bei der Kommunikation eingesetzten Nachrichten.

B.1 Klassendiagramm

Das Klassendiagramm [Boo 99], [OMG 03], [Rum 99] dient zur Beschreibung der Programmlogik eines objektorientierten Programms. Hier werden die Klassen, deren Attribute (Variablen) und deren Methoden (Funktionen) beschrieben. Die Klassen werden über gerichtete Relationen mit unterschiedlicher Bedeutung untereinander verbunden um deren Zusammenhänge darzustellen. Die folgende Symbolik (Bild 46) wird dabei verwendet:

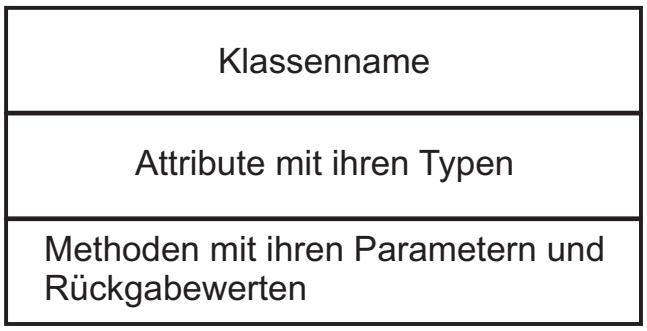


Bild 46: Symbol einer Klasse innerhalb des Klassendiagramms

Die gerichteten Relationen mit ihrer Bedeutung sind in Tabelle 10 zusammengefasst. Die Zahlenwerte an den Enden der Symbole geben dabei jeweils die Kardinalität an, d. h. die Anzahl der Instanzen der Klasse, die jeweils am anderen Ende des Relationssymbols aufgeführt sein müssen.

Relationsname	Relationssymbol	Bedeutung
Abhängigkeit		Die Abhängigkeit zeigt eine semantische Beziehung zwischen den beiden Klassen an.
Assoziation		Zwei Objekte stehen miteinander in Beziehung.

Relationsname	Relationssymbol	Bedeutung
Aggregation		Die Klasse, die auf der Seite der Raute ist, stellt das Aggregat dar. Bei der Klasse auf der anderen Seite handelt es sich um eine Klasse, die die Bestandteile des Aggregats repräsentiert. Die Aggregation ist eine Spezialisierung der Assoziation.
Komposition		Komposition ist die strengere Form der Aggregation. Im Gegensatz zur Aggregation sind die instanziierten Bestandteile in der Komposition fest mit dem Aggregat verbunden. Weder das Aggregat noch dessen Bestandteile können getrennt voneinander
Generalisierung		Die Klasse auf der Seite des Dreiecks ist die Generalisierung einer anderen Klasse. Sie ist damit die Grundlage für die andere Klasse. Über das Konstrukt Generalisierung werden Vererbungshierarchien aufgebaut.

Tabelle 10: Relationstypen des Klassendiagramms

B.2 Sequenzdiagramm

Im Gegensatz zum Klassendiagramm werden mit Hilfe der Sequenzdiagramme [Boo 99] die Algorithmen in objektorientierten Programmen modelliert. Die Sequenzdiagramme kommen aber auch bei der Modellierung von Nachrichtenflüssen zwischen Prozessen zum Einsatz. Dabei wird im Gegensatz zum Klassendiagramm nicht mehr nur der statische Zusammenhang der Klassen betrachtet, sondern der Informationsfluss zwischen realen Instanzen.

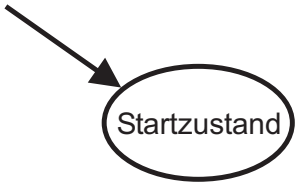
Die Instanzen von Klassen bzw. Prozessen werden dabei durch Quadrate symbolisiert. Vom oberen Ende des Diagramms zum unteren Ende verläuft die Zeitachse, wodurch die zeitliche Abfolge festgelegt ist. Die Methodenaufrufe bzw. die versendeten Nachrichten sind durch Pfeile zwischen den Instanzen symbolisiert. Wird eine Methode derselben Instanz aufgerufen oder derselben Instanz eine Nachricht geschickt, so ist das durch eine Schleife gekennzeichnet. Um Methodenaufrufe von Nachrichten unterscheiden zu können, werden Nachrichten durch das Schlüsselwort *Nachricht* ausgezeichnet. Mit Hilfe des Rautesymbols wird angedeutet, dass je nach Systembedingung alternative Nachrichten bzw. Methodenaufrufe stattfinden können. Die Balken auf der Zeitachse je Instanz symbolisieren die Verarbeitungszeit durch die Instanz bzw. den Prozess (siehe Tabelle 11).

Symbol	Bedeutung
	Instanz bzw. Prozess
	Zeitachse je Instanz
	Beliebig lange Zeitspanne ohne Aktion
	Zerstörung der Instanz bzw. Beendigung des Prozesses.
	Verarbeitungszeitraum
	Methodenaufruf bzw. Versand einer Nachricht
	Alternativer Methodenaufruf bzw. Versand alternativer Nachrichten

Tabelle 11: Symbole des Sequenzdiagramms

B.3 Zustandübergangsdiagramm

Das Zustandübergangsdiagramm dient zur Beschreibung von Protokollen. Dabei repräsentieren die Zustände den jeweiligen Zustand eines Protokolls und der Zustandswechsel das Fortschreiten innerhalb des Protokolls. Die Auswahl des richtigen Zustandübergangs wird bestimmt durch die Eingabe, während der Zustandswechsel und damit die Aktivität des dargestellten Protokolls durch die Ausgabe repräsentiert werden. In Tabelle 12 sind die Symbole und ihre Bedeutung zusammengefasst. Zusätzlich zum klassischen Zustandübergangsdiagramm ist die Eingabe vom Sender, der die Eingabe tätigt, und die Ausgabe vom Empfänger, der die Ausgabe empfangen soll, abhängig. Die Sender und Empfänger sind beim Symbol *Zustandswechsel* in Klammern vermerkt. Das Schlüsselwort *All* bedeutet, dass die Eingabe vom Sender unabhängig ist bzw. die Ausgabe an alle möglichen Empfänger gehen soll.

Symbol	Bedeutung
	beliebiger Zustand
	Startzustand

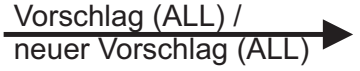
Symbol	Bedeutung
	Endzustand
	Zustandswechsel

Tabelle 12: Symbole des Zustandübergangsdiagramms

B.4 Erweiterte Backus-Naur-Form (EBNF)

Die erweiterte Backus-Naur-Form (EBNF) [ISO 14977] dient zur Spezifikation der Syntax einer Sprache. Sie besteht aus Termen, die von links nach rechts aufgelöst werden. Ein Term ist dabei eine Vorschrift, die mit einem Nicht-Terminal-Symbol beginnt und das durch die rechte Seite des Terms beschrieben wird, wobei die rechte Seite des Terms wiederum Nicht-Terminal-Symbole enthalten kann, die dann ebenfalls wieder rekursiv aufgelöst werden müssen. Des Weiteren kann die rechte Seite des Terms Symbole für die Wiederholung bzw. Auswahl von Ausdrücken enthalten. Die genaue Bedeutung der Symbole sind in Tabelle 13 aufgeführt.

Symbol	Bedeutung
Nicht-Terminal-Symbol	Platzhalter für einen Ausdruck.
„Terminal-Symbol“	Begriff, der in der Sprache verwendet wird.
Nicht-Terminal-Symbol = Ausdruck;	Term, wobei Ausdruck ein beliebiger Ausdruck bestehend aus Nicht-Terminal-Symbolen, Terminal-Symbolen und anderen Sprachelementen darstellt. Der Term wird mit *;* abgeschlossen.
(Terminal-Symbol | Nicht-Terminal-Symbol)	Beliebiger Ausdruck. Hier eine Auswahl zwischen einem Terminal-Symbol und einem Nicht-Terminal-Symbol.
(Ausdruck1 | Ausdruck2 | …)	Einfachauswahl zwischen beliebig vielen Ausdrücken.
[Ausdruck]	Wiederholung eines Ausdrucks. Der Ausdruck wird nicht oder einmal verwendet.
{ Ausdruck }	Wiederholung eines Ausdrucks. Der Ausdruck wird nicht oder beliebig oft verwendet.
Ausdruck1 – Ausdruck2	Ausdruck2 darf nicht in Ausdruck1 enthalten sein.
(* Text *)	Kommentare werden durch *(** und ***)* begrenzt.

Tabelle 13: Symbole der EBNF

B.5 XML-Notation

Die Extensible Markup Language (XML)-Notation [Bra 04], [Eck 00b] ist eine Repräsentationssprache und wird zur Strukturierung von Informationen genutzt. Dabei können Hierarchien aufgebaut werden. Das XML-Dokument enthält die eigentliche Information. Die Vorschrift wie das XML-Dokument aufgebaut sein soll wird durch eine Repräsentationsbeschreibungssprache, der XML-DTD, spezifiziert. In Tabelle 14 sind die wichtigsten Strukturelemente, die in dieser Arbeit verwendet werden, in Form einer EBNF [ISO 14977] aufgezeigt. Die vollständige Spezifikation findet sich im Standard [Bra 04].

Comment	=	(* Kommentare werden zwischen <*!*-- und --> eingeschlossen. *);
S	=	(* Leerzeichen *);
Name	=	(* beliebiger Text bestehend aus Buchstaben, Ziffern, _ und *:*. Der Text muss mit einem Buchstaben beginnen *);
Eq	=	S? “=“ S?;
CharData	=	(* Ist eine beliebige Zeichenkette *);
Document	=	Prolog Element Misc*;
Prolog	=	(* Im Prolog wird der Dokumententyp, die Version und weitere Basis-Informationen über das XML-Dokument hinterlegt *);
Element	=	(EmptyElemTag | STag Content ETag);
EmptyElemTag	=	(* Hierbei handelt es sich um eine Kennzeichnung die kein weiteres Element oder eine Information enthält *);
STag	=	“<“ Name (S Attribute)* S? “>“;
ETag	=	“</“ Name S? “>“;
Attribute	=	Name Eq AttValue;
AttValue	=	(* Inhalt des Attributes *);
Content	=	CharData? ((Element | Reference | CDSect | PI | Comment) CharData?)*;

Tabelle 14: EBNF der XML-Notation

B.6 XML-DTD

Mit Hilfe der Data Type Definition (DTD) für XML [Bra 04] wird die Form des XML-Dokuments spezifiziert. Die DTD dient zur Verifikation des XML-Dokuments und als Vorlage, welche Information das XML-Dokument enthalten muss und wie diese strukturiert ist. Sie stellt damit eine Metaebene dar und sollte für jeden Typ XML-Dokument angegeben sein. Für die im Anhang definierten Nachrichtenformate

wurde jeweils eine DTD erstellt. Die verwendeten grundlegenden Sprachelemente der DTD ist in EBNF Darstellung [ISO 14977] in Tabelle 15 zusammengefasst. Der vollständige Standard findet sich in Bray [Bra 04].

Comment	=	(* Kommentare werden zwischen <!-- und --> eingeschlossen. *);
S	=	(* Leerzeichen *);
Name	=	(* beliebiger Text bestehend aus Buchstaben, Ziffern, _ und :. Der Text muss mit einem Buchstaben beginnen *);
AttValue	=	(* Inhalt des Attributs *);
Elementdecl	=	“<!ELEMENT“ S Name S Contentspec S? “>“;
Contentspec	=	(“EMPTY” | “ANY” | Mixed | Children);
Children	=	(Choice | Seq) (“?” | “*“ | “+“)?;
Cp	=	(Name | Choice | Seq) (“?“ | “*“ | “+“)?;
Choice	=	“(“ S? Cp (S? “|“ S? Cp)+ S? “)“;
Seq	=	“(“ S? Cp (S? “,“ S? Cp)* S? “)“;
Mixed	=	(“(“ S? “#PCDATA“ (S? “|“ S? Name)* S? “)*“ | “(“ S? “#PCDATA“ S? “)“);
AttlistDecl	=	“<!ATTLIST“ S Name AttDef* S? “>”;
AttDef	=	S Name S AttType S DefaultDecl;
AttType	=	(StringType | TokenizedType | EnumeratedType);
StringType	=	“CDATA”;
TokenizedType	=	(“ID” | “IDREF” | “IDREFS” | “ENTITY” | “ENTITIES” | “NMTOKEN” | “NMTOKENS”);
DefaultDecl	=	(“#REQUIRED” | “#IMPLIED” | ((“#FIXED” S)? AttValue));

Tabelle 15: EBNF der XML-DTD

C Definition der Anfragesprachen der RPD-Agenten

C.1 Anfragesprache des Retrieval-Agenten

In Tabelle 16 ist die Syntax der ASN-QL für den Retrieval-Agenten in erweiterter Backus Naur Form (EBNF) [ISO 14977] angegeben. Der Einstieg erfolgt über das Schlüsselwort (Non-Terminal) *ASN-QL*. Mit Hilfe der *QFunction* kann das Suchergebnis quantifiziert bzw. qualifiziert werden. Die *SemanticFunction* beschreibt die Suchanfrage. Mit *Restriction* kann der Suchraum begrenzt werden, während mit *Navigation* die Ergebnisse genauer betrachtet werden können. Die Erweiterungen für die scharfen Methoden finden sich in Tabelle 17 und für die unscharfen Methoden in Tabelle 18.

ASN-QL	=	[QFunction] SemanticFunction [Restriction] [Navigation];
SemanticFunction	=	(Semantic “(“ ConceptDescr “,” Range “)” | Semantic “(“ SemanticFunction “,” Range “)” | Semantic “(“ ConceptDescr, ConceptDescr “,” Range “)” | Semantic “(“ ConceptDescr, SemanticFunction “,” Range “)”);
ConceptDescrSharp	=	“ Concept “ (“ID” id | (“TYPE” Type | “NAME” Name | ”TYPE” Type “NAME” Name) { “ATTR” Name, { “,” Name}});
ConceptDescr	=	(ConceptDescrSharp | ConceptDescrUnsharp);
Semantic	=	(“bestehtaus“ | “istaehnlich“ | “hatBeziehungzu“ | “findeKonzept“ (* akzeptiert nur ein Argument *) | “istabgeleitet“);
QFunction	=	(“Welche“ | “Wieviele“ | “Wie oft“ | “Womit“);
Restriction	=	AttributeDescr Operator Value { (OR | AND) Restriction};
AttributeDescr	=	String;
Value	=	(String | Digit);
Operator	=	(“=“ | “<>” | “<” | “<=” | “>” | “>=”);
Navigation	=	(“+” | “-“);
Range	=	{Digit}-;
Digit	=	(“0“ |“1“| “2“| “3“| “4“ |“5“ | “6“ |“7“ |“8“ |“9“);

Tabelle 16: EBNF der Anfragesprache ASN-QL

In Tabelle 17 ist der Teil der Anfragesprache für die scharfen Methoden in erweiterter Backus Naur Form (EBNF) [ISO 14977] dargestellt und ist dabei eine Fortführung der in Tabelle 16 dargestellten Anfragesprache. Es wird die *SemanticFunction* um die Anfrageelemente der scharfen Methode erweitert.

SharpMethods	=	(“get-concept” “(“ ConceptDescr { “,” ConceptDescr } “)” | “isAssociatedWith” Parameter | “isAggregatedOf” Parameter | “isComposedOf” Parameter | “isGeneralizationOf” Parameter);
Parameter	=	(“(“ ConceptDescr, Range “)” | “(“ ConceptDescr, ConceptDescr, Range “)”);

Tabelle 17: EBNF für die scharfe Suche

In Tabelle 18 ist der Teil der Anfragesprache für die unscharfen Methoden in erweiterter Backus Naur Form (EBNF) [ISO 14977] dargestellt, die eine Fortführung der in Tabelle 16 dargestellten Anfragesprache ist. Es wird die *SemanticFunction* um die Anfrageelemente der unscharfen Methoden erweitert. Neben den unten beschriebenen Methoden erlaubt der Begriff *Keyword* die freie Suche nach Informationen ohne die Struktur der Daten zu kennen. Mit dem *RatingLimit* wird der Ergebnisraum auf die relevanten Ergebnisse beschränkt.

UnsharpMethods	=	(“get-concept-unsharp” “(“ ConceptDescrUnsharp, RatingLimit “)” | “isRelatedWith” “(“ ConceptDescr, Range, RatingLimit “)” | “isRelatedWith” “(“ ConceptDescr, ConceptDescr, Range, RatingLimit “)” | “consistsOf” “(“ ConceptDescr, Range, RatingLimit “)” | “consistsOf” “(“ ConceptDescr, ConceptDescr, Range, RatingLimit “)” | “isSimilarTo” “(“ ConceptDescr , Range, RangeLimit “)”);
ConceptDescrUnsharp	=	(“ConceptDescrUnsharp” [“TYPE” Type] [“NAME” Name] { “ATTR” Name , { “,” Name }} | “Keyword” Keyword {, Keyword} | “ RegularExpression” RegularExpression { , RegularExpression });
RegularExpression	=	(X | Y | XY | X”|”Y | “(“ X”) | X”*” | X” ?” | X”+”) ;

X	=	String;
Y	=	String;
Keyword	=	{Char}-;

Tabelle 18: EBNF für die unscharfe Suche

C.2 Anfragesprache des Aggregations-Agenten

In Bild 47 ist die Anfragesprache und die Sprachdefinition für die Ergebnisse des Aggregations-Agenten in Form einer XML-DTD dargestellt.

```
<!ELEMENT Aggregation (Element)*>
<!ATTLIST Aggregation
        name CDATA #REQUIRED
        type CDATA #REQUIRED
        context CDATA #IMPLIED>

<!ELEMENT Element (Query|Result)*>
<!ATTLIST Element
        name CDATA #REQUIRED
        type CDATA #REQUIRED>

<!ELEMENT Query (QuerySource)*>
<!ATTLIST Query
        id CDATA #REQUIRED
        name CDATA #IMPLIED
        resultType CDATA #REQUIRED>

<!ELEMENT QuerySource (#PCDATA)>
<!ATTLIST QuerySource
        queryType CDATA #REQUIRED>

<!ELEMENT Result (#PCDATA | concept)*>
<!ATTLIST Result
        queryId CDATA #REQUIRED>

<!ELEMENT attribute (#PCDATA)>
<!ATTLIST attribute
        name CDATA #REQUIRED
        type CDATA #REQUIRED
        id CDATA #IMPLIED>
```

```
<!ELEMENT concept (searchinfo, attributes, relations)>
<!ATTLIST concept
        name CDATA #REQUIRED
        type CDATA #REQUIRED
        id CDATA #IMPLIED>

<!ELEMENT relation (#PCDATA)>
<!ATTLIST relation
        name CDATA #REQUIRED
        multiplicity CDATA #REQUIRED
        direction CDATA #REQUIRED
        type CDATA #REQUIRED>

<!ELEMENT attributes (attribute)*>

<!ELEMENT relations (relation)*>

<!ELEMENT searchinfo>
<!ATTLIST searchinfo
        rating CDATA #REQUIRED
        retrievedby CDATA #REQUIRED
        foundrelatedto CDATA #REQUIRED
        direction CDATA #REQUIRED>
```

Bild 47: Syntax für die Aggregationsanfrage und das Aggregationsergebnis

C.3 Anfragesprache des Monitor-Agenten

In Tabelle 19 ist die Syntax der Anfragesprache zur Spezifikation der Überwachungsbedingung in erweiterter Backus Naur Form (EBNF) [ISO 14977] dargestellt.

Der Einstieg in die Sprachdefinition erfolgt über das Schlüsselwort (Non-Terminal) *Request*. Über das Non-Terminal *Expr* werden die Überwachungsbedingung bzw. die Teilbedingungen spezifiziert. Hierbei ist es möglich entweder eine Strukturfunktion mit den dafür vorgesehenen Parametern auszuwählen oder zwei Werte auf ihre Ordnung zueinander zu vergleichen oder zwei Teilbedingungen aussagenlogisch miteinander zu verknüpfen. Beim Vergleich zweier String-Werte wird die alphanumerische Ordnung herangezogen. Beim Vergleich zweier Zahlenwerte finden neben den Ganzen- und Realen-Zahlen auch die Ergebnisse von Zugriffs- und mathematischer Funktionen Verwendung.

Request	=	Expr;
Value	=	“Integer(“<Integerwert>”)”;
Value	=	“Real(“<Realwert>”)”;

Value	=	“Value” ValueAttribute;
Value	=	“(“ AccessFunc ValueAttribute “)”;
Value	=	“(“ MathFunc Value Value “)”;
ValueString	=	“String(“ (<String> | “Value” ValueAttribute) ”)”;
ValueAttribute	=	“(“ <Netzname> “,” <Konzeptname> “,” <Attributname> “)”;
ValueConcept	=	“(“ <Netzname> “,“ <Konzeptname> “)“;
ValueNet	=	“(“ <Netzname> “)”;
ValueRelation	=	“(“ <Netzname> “,“ <Konzeptname> “,“ <Relationname> “)“;
AccessFunc	=	(“CountRead” | “CountWrite” | “Version”);
MathFunc	=	(“Add“ | “Sub“ | “Mul“ | “Div“);
StructureFunc1	=	(“ChangeVersion“ | “ChangeAttributeValue“ | “DeleteAttribute“);
StructureFunc2	=	“DeleteConcept“;
StructureFunc3	=	“DeleteNet“;
StructureFunc4	=	“DeleteRelation“;
StructureFunc5	=	“IncludeConcept“;
Operator1	=	(“<” | “>” | “=” | “<>” | “<=” | “>=”);
Operator2	=	(“AND“ | “OR“ | “NAND“ | “NOR“ | “XOR“);
Expr	=	StructureFunc1 ValueAttribute;
Expr	=	StructureFunc2 ValueConcept;
Expr	=	StructureFunc3 ValueNet;
Expr	=	StructureFunc4 ValueRelation;
Expr	=	StrcutureFunc5 ValueRelation ValueConcept;
Expr	=	“NOT“ “(“ Expr “)”;
Expr	=	Value Operator1 Value;
Expr	=	ValueString Operator1 ValueString;
Expr	=	“(“ Expr “)“ Operator2 “(“ Expr “)“;

Tabelle 19: EBNF der Monitor-Anfrage

C.4 Anfragesprache des Koordinations-Agenten

In Tabelle 20 ist in erweiterter Backus Naur Form (EBNF) [ISO 14977] die Sprache zur Beschreibung des Zustandübergangsdiagramm für den Koordinations-Agenten angegeben.

State Transition Diagram (STD) stellt hierbei den Startpunkt der Sprachdefinition dar. Es werden zuerst alle Zustände vom Startzustand über die Zwischenzustände, die als Normalzustände bezeichnet werden, hin zu den Endzuständen aufgelistet. Der entsprechende Typ des Zustandes wird durch die Schüsselworte *Start*, *Normal*, *Ende* angegeben. Anschließend werden die Zustandsübergänge aufgeführt. Die Zustandsübergänge bestehen aus Ihrem Anfangszustand gefolgt durch den Zielzustand, sowie der Ein- und Ausgabenachricht. Die Ein- und Ausgabenachrichten sind eingeklammert, um auch die Darstellung von Leerzeichen in den Nachrichten zu ermöglichen. Die Sender- bzw. Empfängerlisten folgen den Ein- und Ausgabenachrichten. Bei den Sender- und Empfängerlisten handelt es sich um offene Listen. Durch das Schlüsselwort *ALL* wird ausgesagt, dass alle Sender bzw. Empfänger gemeint sind.

STD	=	Startstate Normalstate {Normalstate} Endstate {Endstate} Transition {Transition};
Transition	=	Statename Statename Input Output;
Statename	=	“(“ <Name> “)“;
Startstate	=	Statename “Start“;
Normalstate	=	Statename “Normal“;
Endstate	=	Statename “Ende“;
Input	=	“(“ <Input> “)” Sender;
Output	=	“(“ <Output> ”)“ Recipient;
Sender	=	(<senderid> {“,” <senderid>} | “ALL“);
Recipient	=	(<recipientid> {“,“ <recipientid>} | “ALL“);

Tabelle 20: EBNF des Zustandübergangsdiagramms

D Algorithmen und Datenstrukturen

In diesem Abschnitt werden die wichtigsten Algorithmen und Datenstrukturen des Retrieval-, Monitor- und Koordinations-Agenten aus Kapitel 6 beschrieben. Die Form der Beschreibung variiert je nach Agententyp.

D.1 Retrieval-Agent

Erläuterungen zu den im Folgenden für den Retrieval-Agenten dargestellten Algorithmen sind als Kommentare direkt in die Algorithmen eingefügt.

D.1.1 Algorithmen für die scharfe Suche

Algorithmus von get-concept(ConceptDescr[] CDA):

/* Diese Funktion sucht alle im ASN existierenden Konzepte, deren Attribute und Relationen, die mit der Beschreibung (*ConceptDescr*), wie sie in der Anfrage formuliert ist, übereinstimmen. */

```
Concept[] ConceptResult
Concept[] ConceptResultName
Concept[] ConceptResultType

// Auswahl des Konzepts, wenn die ID spezifiziert wurde
for i := 1 to CDA[].max
   if CDA[i].id ! = Empty
      ConceptResult[].add(findConceptByPrimaryKey(CDA[i].id) )
      CDA[].delete(i)

// Auswahl des Konzepts, wenn der Name oder der Typ spezifiziert wurde
for i := 1 to CDA[].max
   if (CDA[i].type = Empty or CDA[i].name = Empty)

      // Auswahl des Konzepts, wenn der Name spezifiziert wurde
      if CDA[i].type = Empty
         Concept[] ConceptByName = findConceptByName(CDA[i].name)
         for each ConceptByName
            if CDA[i].attributes ∈ ConceptByName
               ConceptResult[].add(ConceptByName)
               CDA[].delete(i)
   else

      // Auswahl des Konzepts, wenn der Typ spezifiziert wurde
      Concept[] ConceptByType = findConceptByType(CDA[i].type)
      for each ConceptByType
```

```
        if CDA[i].attributes ∈ ConceptByType
            ConceptResult[].add(ConceptByType)
            CDA[].delete(i)

// Auswahl des Konzepts, wenn der Name und der Typ spezifiziert wurde
for i := 1 to CDA[].max
    if (CDA[i].type ! = Empty and CDA[i].name) ! = Empty
        ConceptResultName[].add(findConceptByName(CDA[i].name))
        ConceptResultType[].add(findConceptByType(CDA[i].type))
        Concept[] ConceptResultTemp = ConceptResultName[] ∩
            ConceptResultType[])
        for each ConceptResultTemp
            if (exists CDA[i].attributes)
                if (CDA[i].attributes ∈ ConceptResultTemp)
                    ConceptResult[].add(ConceptResultTemp)
                    CDA[].delete(i)
                else
                    ConceptResult[].add(ConceptResultTemp)
                    CDA[].delete(i)
return (ConceptResult[])
```

Algorithmus von is-associated-with(ConceptDescr[] CDA, int Range):

/* Mit dieser Funktion werden alle ausschließlich assoziierten Konzepte gesucht. Für die Suche werden als Parameter ein oder mehrere Startkonzepte und die Größe der zu durchsuchenden Umgebung (Range) herangezogen. */

```
Concept[] ConceptResult
Concept[] ConceptVisited
Concept[] StartConcept
Concept ActiveConcept
int ActiveConceptRange

// Startkonzepte bestimmen
if CDA = sharp
    StartConcept[] = get-concept(CDA)
else
    StartConcept[] = get-concept-unsharp(CDA)

// Suche ausführen für alle Startkonzepte, die die Suchanfrage erfüllen
for each Concept ∈ StartConcept[]
    Int StartRange = 0
    ConceptStack Stack = [StartConcept, StartRange]
```

```
    // Suche durchführen
    while (Stack not Empty)

        // oberstes Konzept des Stacks zur weiteren Suche auswählen, wenn es
        // innerhalb der Range liegt
        repeat
            ActiveConcept = Stack.pop
            ActiveConceptRange = Stack.pop
        until ActiveConceptRange < Range

        // Alle Relationen des ausgewählten Konzepts untersuchen
        RelationsActiveConcept[] = getAllRelation(ActiveConcept)
        for i :=1 to RelationsActiveConcept[].max

            // Nur Assoziationen weiterverfolgen
            if RelationsActiveConcept[i].getType = “Assoziation“

                // Wenn das ausgewählte Konzept noch nicht besucht wurde, dann das
                // aktuelle Konzept der Ergebnismenge hinzufügen und alle abgehenden
                // Konzepte der Suchmenge hinzufügen
                if (RelationsActiveConcept[i].getName() ∉ ConceptVisited[]) and
                    (Range < (Range ( ActiveConcept, StartConcept) + 1))
                    Stack.push(Range(ActiveConcept, StartConcept))
                    Stack.push(RelationsActiveConcept[i].getName())
                    ConceptResult[].add(ActiveConcept)
                    ConceptVisited[].add(ActiveConcept)
return (ConceptResult[])
```

Algorithmus von is-associated-with(ConceptDescr[] CDA, ConceptDescr[] CDB, int Range):

/* Mit dieser Funktion werden alle ausschließlich assoziierten Konzepte gesucht. Für die Suche werden als Parameter ein oder mehrere Start- und Zielkonzepte benötigt. Ebenfalls wird die Suche auf eine gewisse Umgebung (Range) beschränkt. Das Ergebnis ist die Menge von Zielkonzepten für die ein assoziierter Pfad zwischen dem betrachteten Start- und Zielkonzept existiert. */

```
Concept[] ConceptResult
Concept[] ConceptVisited
Concept[] StartConcept
Concept[] DestinationConcept
Concept ActiveConcept
int ActiveConceptRange
```

```
// Startkonzepte bestimmen
if CDA = sharp
   StartConcept[] = get-concept(CDA)
else
   StartConcept[] = get-concept-unsharp(CDA)

// Zielkonzepte bestimmen
if CDB = sharp
   DestinationConcept[] = get-concept(CDB)
else
   DestinationConcept[] = get-concept-unsharp(CDB)

// Suche ausführen für alle Startkonzepte, die die Suchanfrage erfüllen
for each Concept ∈ Startconcept[]
   Int StartRange = 0
   ConceptStack Stack = [StartConcept, StartRange]

   // Suche durchführen
   while (Stack not empty)

      // oberstes Konzept des Stacks zur weiteren Suche auswählen, wenn es
      // innerhalb der Range liegt
      repeat
         ActiveConcept = Stack.pop
         ActiveConceptRange = Stack.pop
      until ActiveConceptRange < Range

      // Alle Relationen des ausgewählten Konzepts untersuchen
      RelationsActiveConcept[] = getAllRelation(ActiveConcept)
      for i := 1 to RelationsActiveConcept[].max

         // Nur Assoziationen weiterverfolgen
         if RelationsActiveConcept[i].getType = “Assoziation”

            // Wenn das ausgewählte Konzept noch nicht besucht wurde, dann alle
            // abgehenden Konzepte der Suchmenge hinzufügen
            if (RelationsActiveConcept[i].getName() ∉ ConceptVisited[]) and
               (Range < (Range ( ActiveConcept, StartConcept) + 1))
               Stack.push(Range(ActiveConcept, StartConcept))
               Stack.push(RelationsActiveConcept[i].getName())

               // Konzepte der Ergebnismenge hinzufügen, wenn sie Bestandteil der
               // Zielmenge sind
               for each Concept ∈ DestinationConcept[]
                  if ActiveConcept = Concept
                     ConceptResult[].add(ActiveConcept)
```

```
                    ConceptVisited[].add(ActiveConcept)
return (ConceptResult[])
```

Algorithmus von is-aggregated-of(ConceptDescr[] CDA, int Range):

/* Mit dieser Funktion werden alle ausschließlich aggregierten Konzepte gesucht. Für die Suche werden als Parameter ein oder mehrere Startkonzepte und die Größe der zu durchsuchenden Umgebung (Range) herangezogen. */

```
Concept[] ConceptResult
Concept[] ConceptVisited
Concept[] StartConcept
Concept ActiveConcept
int ActiveConceptRange

// Startkonzepte bestimmen
if CDA = sharp
   StartConcept[] = get-concept(CDA)
else
   StartConcept[] = get-concept-unsharp(CDA)

// Suche ausführen für alle Startkonzepte, die die Suchanfrage erfüllen
for each Concept ∈ Startconcept[]
   Int StartRange = 0
   ConceptStack Stack = [StartConcept, StartRange]

   // Suche durchführen
   while (Stack not Empty)

      // oberstes Konzept des Stacks zur weiteren Suche auswählen, wenn es
      // innerhalb der Range liegt
      repeat
         ActiveConcept = Stack.pop
         ActiveConceptRange = Stack.pop
      until ActiveConceptRange < Range

      // Alle Relationen des ausgewählten Konzepts untersuchen
      RelationsActiveConcept[] = getAllRelation(ActiveConcept)
      for i := 1 to RelationsActiveConcept[].max

         // Nur Aggregationen weiterverfolgen
         if RelationsActiveConcept[i].getType = “Aggregation”

            // Wenn das ausgewählte Konzept noch nicht besucht wurde, dann das
            // aktuelle Konzept der Ergebnismenge hinzufügen und alle abgehenden
            // Konzepte der Suchmenge hinzufügen
```

```
            if (RelationsActiveConcept[i].getName() ∉ ConceptVisited[]) and
               (Range < (Range ( ActiveConcept, StartConcept) + 1))
               Stack.push(Range(ActiveConcept, StartConcept))
               Stack.push(RelationsActiveConcept[i].getName())
               ConceptResult[].add(ActiveConcept)
               ConceptVisited[].add(ActiveConcept)
return (ConceptResult[])
```

Algorithmus von is-aggregated-of(ConceptDescr[] CDA, ConceptDescr[] CDB, int Range):

/* Mit dieser Funktion werden alle ausschließlich aggregierten Konzepte gesucht. Für die Suche werden als Parameter ein oder mehrere Start- und Zielkonzepte benötigt. Ebenfalls wird die Suche auf eine gewisse Umgebung (Range) beschränkt. Das Ergebnis ist die Menge von Zielkonzepten für die ein assoziierter Pfad zwischen dem betrachteten Start- und Zielkonzept existiert, wobei diese über die Beziehung Aggregation miteinander verbunden sind. */

```
Concept[] ConceptResult
Concept[] ConceptVisited
Concept[] StartConcept
Concept[] DestinationConcept
Concept ActiveConcept
int ActiveConceptRange

// Startkonzepte bestimmen
if CDA = sharp
   StartConcept[] = get-concept(CDA)
else
   StartConcept[] = get-concept-unsharp(CDA)

// Zielkonzepte bestimmen
if CDB = sharp
   DestinationConcept[] = get-concept(CDB)
else
   DestinationConcept[] = get-concept-unsharp(CDB)

// Suche ausführen für alle Startkonzepte, die die Suchanfrage erfüllen
for each Concept ∈ StartConcept[]
   Int StartRange = 0
   ConceptStack Stack = [StartConcept, StartRange]

   // Suche durchführen
   while (Stack not empty)
```

```
    // oberstes Konzept des Stacks zur weiteren Suche auswählen, wenn es
    // innerhalb der Range liegt
    repeat
       ActiveConcept = Stack.pop
       ActiveConceptRange = Stack.pop
    until ActiveConceptRange < Range

    // Alle Relationen des ausgewählten Konzepts untersuchen
    RelationsActiveConcept[] = getAllRelation(ActiveConcept)
    for i := 1 to RelationsActiveConcept[].max

       // Nur Aggregationen weiterverfolgen
       if RelationsActiveConcept[i].getType = “Aggregation”

          // Wenn das ausgewählte Konzept noch nicht besucht wurde, dann alle
          // abgehenden Konzepte der Suchmenge hinzufügen
          if (RelationsActiveConcept[i].getName() ∉ ConceptVisited[]) and
             (Range < (Range ( ActiveConcept, StartConcept) + 1))
             Stack.push(Range(ActiveConcept, StartConcept))
             Stack.push(RelationsActiveConcept[i].getName())

             // Konzepte der Ergebnismenge hinzufügen, wenn sie Bestandteil der
             // Zielmenge sind
             for each Concept ∈ DestinationConcept[]
                if ActiveConcept = Concept
                   ConceptResult[].add(ActiveConcept)
                   ConceptVisited[].add(ActiveConcept)
return (ConceptResult[])
```

Algorithmus von is-composed-of(ConceptDescr[] CDA, int Range):

/* Mit dieser Funktion werden alle Konzepte gesucht zwischen denen eine Kompositionsbeziehung besteht. Für die Suche werden als Parameter ein oder mehrere Startkonzepte und die Größe der zu durchsuchenden Umgebung (Range) herangezogen. */

```
Concept[] ConceptResult
Concept[] ConceptVisited
Concept[] StartConcept
Concept ActiveConcept
int ActiveConceptRange

// Startkonzepte bestimmen
if CDA = sharp
   StartConcept[] = get-concept(CDA)
```

```
else
   StartConcept[] = get-concept-unsharp(CDA)

// Suche ausführen für alle Startkonzepte, die die Suchanfrage erfüllen
for each Concept ∈ StartConcept[]
   Int StartRange = 0
   ConceptStack Stack = [StartConcept, StartRange]

   // Suche durchführen
   while (Stack not Empty)

      // oberstes Konzept des Stacks zur weiteren Suche auswählen, wenn es
      // innerhalb der Range liegt
      repeat
         ActiveConcept = Stack.pop
         ActiveConceptRange = Stack.pop
      until ActiveConceptRange < Range

      // Alle Relationen des ausgewählten Konzepts untersuchen
      RelationsActiveConcept[] = getAllRelation(ActiveConcept)
      for i := 1 to RelationsActiveConcept[].max

         // Nur Kompositionen weiterverfolgen
         if RelationsActiveConcept[i].getType = “Komposition”

            // Wenn das ausgewählte Konzept noch nicht besucht wurde, dann das
            // aktuelle Konzept der Ergebnismenge hinzufügen und alle abgehenden
            // Konzepte der Suchmenge hinzufügen
            if (RelationsActiveConcept[i].getName() ∉ ConceptVisited[]) and
               (Range < (Range ( ActiveConcept, StartConcept) + 1))
               Stack.push(Range(ActiveConcept, StartConcept))
               Stack.push(RelationsActiveConcept[i].getName())
               ConceptResult[].add(ActiveConcept)
               ConceptVisited[].add(ActiveConcept)
return (ConceptResult[])
```

Algorithmus von is-composed-of(ConceptDescr[] CDA, ConceptDescr[] CDB, int Range):

/* Mit dieser Funktion werden alle Konzepte gesucht zwischen denen eine Kompositionsbeziehung besteht. Für die Suche werden als Parameter ein oder mehrere Start- und Zielkonzepte benötigt. Ebenfalls wird die Suche auf eine gewisse Umgebung (Range) beschränkt. Das Ergebnis ist die Menge von Zielkonzepten für die ein assoziierter Pfad zwischen dem betrachteten Start- und Zielkonzept existiert, wobei diese über die Beziehung Komposition miteinander verbunden sind. */

```
Concept[] ConceptResult
Concept[] ConceptVisited
Concept[] StartConcept
Concept[] DestinationConcept
Concept ActiveConcept
int ActiveConceptRange

// Startkonzepte bestimmen
if CDA = sharp
   StartConcept[] = get-concept(CDA)
else
   StartConcept[] = get-concept-unsharp(CDA)
if CDB = sharp

// Zielkonzepte bestimmen
   DestinationConcept[] = get-concept(CDB)
else
   DestinationConcept[] = get-concept-unsharp(CDB)

// Suche ausführen für alle Startkonzepte, die die Suchanfrage erfüllen
for each Concept ∈ StartConcept[]
   Int StartRange = 0
   ConceptStack Stack = [StartConcept, StartRange]

   // Suche durchführen
   while (Stack not empty)

      // oberstes Konzept des Stacks zur weiteren Suche auswählen, wenn es
      // innerhalb der Range liegt
      repeat
         ActiveConcept = Stack.pop
         ActiveConceptRange = Stack.pop
      until ActiveConceptRange < Range

      // Alle Relationen des ausgewählten Konzepts untersuchen
      RelationsActiveConcept[] = getAllRelation(ActiveConcept)
      for i := 1 to RelationsActiveConcept[].max

         // Nur Kompositionen weiterverfolgen
         if RelationsActiveConcept[i].getType = “Komposition”

            // Wenn das ausgewählte Konzept noch nicht besucht wurde, dann alle
            // abgehenden Konzepte der Suchmenge hinzufügen
            if (RelationsActiveConcept[i].getName() ∉ ConceptVisited[]) and
               (Range < (Range ( ActiveConcept, StartConcept) + 1))
```

```
            Stack.push(Range(ActiveConcept, StartConcept))
            Stack.push(RelationsActiveConcept[i].getName())

            // Konzepte der Ergebnismenge hinzufügen, wenn sie Bestandteil der
            // Zielmenge sind
            for each Concept ∈ DestinationConcept[]
               if ActiveConcept = Concept
                  ConceptResult[].add(ActiveConcept)
                  ConceptVisited[].add(ActiveConcept)
return (ConceptResult[])
```

Algorithmus von is-generalization-of(ConceptDescr[] CDA, int Range):

/* Mit dieser Funktion werden alle Konzepte gesucht zwischen denen eine Generalisierungsbeziehung besteht. Für die Suche werden als Parameter ein oder mehrere Startkonzepte und die Größe der zu durchsuchenden Umgebung (Range) herangezogen. */

```
Concept[] ConceptResult
Concept[] ConceptVisited
Concept[] StartConcept
Concept ActiveConcept
int ActiveConceptRange

// Startkonzepte bestimmen
if CDA = sharp
   StartConcept[] = get-concept(CDA)
else
   StartConcept[] = get-concept-unsharp(CDA)

// Suche ausführen für alle Startkonzepte, die die Suchanfrage erfüllen
for each Concept ∈ StartConcept[]
   Int StartRange = 0
   ConceptStack Stack = [StartConcept, StartRange]

   // Suche durchführen
   while (Stack not Empty)

      // oberstes Konzept des Stacks zur weiteren Suche auswählen, wenn es
      // innerhalb der Range liegt
      repeat
         ActiveConcept = Stack.pop
         ActiveConceptRange = Stack.pop
      until ActiveConceptRange < Range
```

```
      // Alle Relationen des ausgewählten Konzepts untersuchen
      RelationsActiveConcept[] = getAllRelation(ActiveConcept)
      for i := 1 to RelationsActiveConcept[].max

         // Nur Generalisierungen weiterverfolgen
         if RelationsActiveConcept[i].getType = “Generalisierung”

            // Wenn das ausgewählte Konzept noch nicht besucht wurde, dann das
            // aktuelle Konzept der Ergebnismenge hinzufügen und alle abgehenden
            // Konzepte der Suchmenge hinzufügen
            if (RelationsActiveConcept[i].getName() ∉ ConceptVisited[]) and
               (Range < (Range ( ActiveConcept, StartConcept) + 1))
               Stack.push(Range(ActiveConcept, StartConcept))
               Stack.push(RelationsActiveConcept[i].getName())
               ConceptResult[].add(ActiveConcept)
               ConceptVisited[].add(ActiveConcept)
return (ConceptResult[])
```

Algorithmus von is-generalization-of(ConceptDescr[] CDA, ConceptDescr[] CDB, int Range):

/* Mit dieser Funktion werden alle Konzepte gesucht zwischen denen eine Generalisierungsbeziehung besteht. Für die Suche werden als Parameter ein oder mehrere Start- und Zielkonzepte benötigt. Ebenfalls wird die Suche auf eine gewisse Umgebung (Range) beschränkt. Das Ergebnis ist die Menge von Zielkonzepten für die ein assoziierter Pfad zwischen dem betrachteten Start- und Zielkonzept existiert, wobei diese über die Beziehung Generalisierung miteinander verbunden sind. */

```
Concept[] ConceptResult
Concept[] ConceptVisited
Concept[] StartConcept
Concept[] DestinationConcept
Concept ActiveConcept
int ActiveConceptRange

// Startkonzepte bestimmen
if CDA = sharp
   StartConcept[] = get-concept(CDA)
else
   StartConcept[] = get-concept-unsharp(CDA)

// Zielkonzepte bestimmen
if CDB = sharp
   DestinationConcept[] = get-concept(CDB)
```

```
else
   DestinationConcept[] = get-concept-unsharp(CDB)

// Suche ausführen für alle Startkonzepte, die die Suchanfrage erfüllen
for each Concept ∈ StartConcept[]
   Int StartRange = 0
   ConceptStack Stack = [StartConcept, StartRange]

   // Suche durchführen
   while (Stack not empty)

      // oberstes Konzept des Stacks zur weiteren Suche auswählen, wenn es
      // innerhalb der Range liegt
      repeat
         ActiveConcept = Stack.pop
         ActiveConceptRange = Stack.pop
      until ActiveConceptRange < Range

      // Alle Relationen des ausgewählten Konzepts untersuchen
      RelationsActiveConcept[] = getAllRelation(ActiveConcept)
      for i := 1 to RelationsActiveConcept[].max

         // Nur Generalisierungen weiterverfolgen
         if RelationsActiveConcept[i].getType = “Generalisierung”

            // Wenn das ausgewählte Konzept noch nicht besucht wurde, dann alle
            // abgehenden Konzepte der Suchmenge hinzufügen
            if (RelationsActiveConcept[i].getName() ∉ ConceptVisited[]) and
               (Range < (Range ( ActiveConcept, StartConcept) + 1))
               Stack.push(Range(ActiveConcept, StartConcept))
               Stack.push(RelationsActiveConcept[i].getName())

               // Konzepte der Ergebnismenge hinzufügen, wenn sie Bestandteil der
               // Zielmenge sind
               for each Concept ∈ DestinationConcept[]
                  if ActiveConcept = Concept
                     ConceptResult[].add(ActiveConcept)
                     ConceptVisited[].add(ActiveConcept)
return (ConceptResult[])
```

D.1.2 Algorithmen für die unscharfe Suche

Algorithmus von get-concept-unsharp(ConceptDescrUnsharp[], int RatingLimit):

```
/* Diese Funktion dient zum Auffinden aller Konzepte, deren Attribute teilweise mit
denen der Anfrage übereinstimmen. */

Concept[] ConceptResult

for i := 1 to ConceptDescrUnsharp[].max

   // Unscharfe Methode “keyword” ausführen
   if ConceptDescrUnsharp[i].contains(“Keyword”)

      // Alle Konzepte und alle Keywords bestimmen
      Concept[] AllConcepts = getAllConcept()
      Keywords[] Keywords = deliveredKeywords
      for x := 1 to AllConcepts[].max
         for y := 1 to Keywords[].max

            // Keyword mit dem Konzeptnamen vergleichen
            if AllConcepts[x].getName().contains(Keyword[y])
               ConceptResult[].add(AllConcepts[x])
               ConceptResult[].rating(AllConcepts[x]) =
               ConceptResult[].rating(AllConcepts[x]) + 1

            // Keyword mit dem Konzepttyp vergleichen
            if AllConcepts[x].getType().contains(Keyword[y])
               ConceptResult[].add(AllConcepts[x])
               ConceptResult[].rating(AllConcepts[x]) =
               ConceptResult[].rating(AllConcepts[x]) + 1

            // Keyword mit den Attributen des Konzepts vergleichen
            Attribute[]ConceptAttributes = getAllAttribute(x)
            for z := 1 to ConceptAttributes[].max
               if ConceptAttributes[z].contains(Keyword[y])
                  ConceptResult[].add(AllConcepts[x])
                  ConceptResult[].rating(AllConcepts[x]) =
                  ConceptResult[].rating(AllConcepts[x]) + 1

            // Keyword mit den vom ausgewählten Konzept abgehenden Relationen
            // vergleichen
            Relation[]ConceptRelations = getAllRelations(x)
               for z := 1 to ConceptRelations[].max
               if ConceptRelations[z].contains(Keyword[y])
```

```
                ConceptResult[].add(AllConcepts[x])
                ConceptResult[].rating(AllConcepts[x]) =
                ConceptResult[].rating(AllConcepts[x]) + 1

   // Restliche unscharfe Methoden anwenden
   else
      Concept[] AllConcepts = getAllConcept()
      for x := 1 to AllConcepts[].max
         for y := 1 to ConceptDescrUnsharp[].max

            // Unscharfe Beschreibung mit dem Konzeptnamen vergleichen
            if AllConcepts[x].getName().contains(ConceptDescrUnsharp[y].name)
               ConceptResult[].add(AllConcepts[x])
               ConceptResult[].rating(AllConcepts[x]) =
               ConceptResult[].rating(AllConcepts[x]) + 1

            // Unscharfe Beschreibung mit dem Konzepttyp vergleichen
            if AllConcepts[x].getType().contains(ConceptDescrUnsharp[y].type)
               ConceptResult[].add(AllConcepts[x])
               ConceptResult[].rating(AllConcepts[x]) =
               ConceptResult[].rating(AllConcepts[x]) + 1

            // Unscharfe Beschreibung mit den Attributen des Konzepts vergleichen
            for each (ConceptDescrUnsharp[].attributes X AllConcepts[].attributes)
               if ConceptDescrUnsharp [].attributes matches AllConcepts[].attribute
                  ConceptResult[].add(AllConcepts[x])
                  ConceptResult[].rating(AllConcepts[x]) =
                  ConceptResult[].rating(AllConcepts[x]) + 1

// Alle Ergebniskonzepte, die das Rating nicht erfüllen aus der Ergebnismenge
// entfernen
for i := 1 to ConceptResult[].max
   if ConceptResult[i].rating < RatingLimit
      ConceptResult[].delete(i)
return (ConceptResult[])
```

Algorithmus von is-related-with(ConceptDescr[] CDA, int Range, int RatingLimit):

/* Diese Funktion sucht alle Konzepte, die zueinander in Beziehung stehen, d. h. die miteinander über eine oder mehrere Relationen verbunden sind. Über die Vorgabe einer Umgebung (Range) wird der Suchraum begrenzt. Die gefundenen Relationen werden gewichtet und damit die semantische Nähe, also der inhaltliche Verwandtschaftsgrad, bestimmt. Das Ergebnis ist damit im besten Fall eine Liste von Konzepten geordnet nach der semantischen Nähe. */

```
Concept[] ConceptResult
Concept[] ConceptVisited
Concept[] StartConcept
Concept ActiveConcept
Concept LastConcept
int ActiveConceptRange

// Startkonzepte bestimmen
if CDA = sharp
   StartConcept[] = get-concept(CDA)
else
   StartConcept[] = get-concept-unscharp(CDA)

// Suche ausführen für alle Startkonzepte, die die Suchanfrage erfüllen
for each Concept ∈ StartConcept[]
   Int StartRange = 0
   ConceptStack Stack = [StartConcept, StartRange]

   // Suche durchführen
   while (Stack not empty)

      // oberstes Konzept des Stacks zur weiteren Suche auswählen, wenn es
      // innerhalb der Range liegt
      repeat
         ActiveConcept = Stack.pop
         LastConcept = Stack.pop
         ActiveConceptRange = Stack.pop
      until ActiveConceptRange < Range

      // Alle Relationen des ausgewählten Konzepts untersuchen
      RelationsActiveConcept[] = getAllRelation(ActiveConcept)
      for i := 1 to RelationsActiveConcept[].max

         // Nur Assoziationen weiterverfolgen
         if RelationsActiveConcept[i].getT ype = “Assoziation“

            // Wenn das ausgewählte Konzept noch nicht besucht wurde, dann das
            // aktuelle Konzept der Ergebnismenge hinzufügen und alle abgehenden
            // Konzepte der Suchmenge hinzufügen
            if RelationsActiveConcept[i].getName() ∉ ConceptVisited[]
               Stack.push(Range(ActiveConcept, StartConcept))
               Stack.push(ActiveConcept)
               Stack.push(RelationsActiveConcept[i].getName())

               // Konzeptverknüpfung der Ratingliste hinzufügen, um die Gewichtung
               // bestimmen zu können
```

```
            RatingList[].add(ActiveConcept,RatingList[LastConcept] +
               “Assoziation“)
            ConceptVisited[].add(ActiveConcept)

        // Nur Generalisierungen weiterverfolgen
        if RelationsActiveConcept[i].getType = “Generalisierung“

            // Wenn das ausgewählte Konzept noch nicht besucht wurde, dann das
            // aktuelle Konzept der Ergebnismenge hinzufügen und alle abgehenden
            // Konzepte der Suchmenge hinzufügen
            if RelationsActiveConcept[i].getName() ∉ ConceptVisited[]
               Stack.push(Range(ActiveConcept, StartConcept))
               Stack.push(ActiveConcept)
               Stack.push(RelationsActiveConcept[i].getName())

               // Konzeptverknüpfung der Ratingliste hinzufügen, um die Gewichtung
               // bestimmen zu können
               RatingList[].add(ActiveConcept,RatingList[LastConcept] +
                  “Generalisierung”)
               ConceptVisited[].add(ActiveConcept)

        // Nur Kompositionen weiterverfolgen
        if RelationsActiveConcept[i].getType = “Komposition“

            // Wenn das ausgewählte Konzept noch nicht besucht wurde, dann das
            // aktuelle Konzept der Ergebnismenge hinzufügen und alle abgehenden
            // Konzepte der Suchmenge hinzufügen
            if RelationsActiveConcept[i].getName() ∉ ConceptVisited[]
               Stack.push(Range(ActiveConcept, StartConcept))
               Stack.push(ActiveConcept)
               Stack.push(RelationsActiveConcept[i].getName())

               // Konzeptverknüpfung der Ratingliste hinzufügen, um die Gewichtung
               // bestimmen zu können
               RatingList[].add(ActiveConcept,RatingList[LastConcept] +
                  “Komposition“)
               ConceptVisited[].add(ActiveConcept)

        // Nur Aggregationen weiterverfolgen
        if RelationsActiveConcept[i].getType = “Aggregation“

            // Wenn das ausgewählte Konzept noch nicht besucht wurde, dann das
            // aktuelle Konzept der Ergebnismenge hinzufügen und alle abgehenden
            // Konzepte der Suchmenge hinzufügen
            if RelationsActiveConcept[i].getName() ∉ ConceptVisited[]
               Stack.push(Range(ActiveConcept, StartConcept))
```

```
            Stack.push(ActiveConcept)
            Stack.push(RelationsActiveConcept[i].getName())

            // Konzeptverknüpfung der Ratingliste hinzufügen, um die Gewichtung
            // bestimmen zu können
            RatingList[].add(ActiveConcept,RatingList[LastConcept] +
            "Aggregation")
            ConceptVisited[].add(ActiveConcept)

// Alle Ergebniskonzepte, die das Rating nicht erfüllen aus der Ergebnismenge
// entfernen
Concept[] RatedConcepts = rating(StartConcept,RatingList[],RatingMode)
for i := 1 to RatedConcepts[].max
   if RatedConcepts[i].rating < RatingLimit
      RatedConcepts[].delete(i)
return (RatedConcepts[])
```

Algorithmus von is-related-with(ConceptDescr[] CDA, ConceptDescr[] CDB, int Range, int RatingLimit):

/* Diese Funktion sucht alle Konzepte, die zueinander in Beziehung stehen, d. h. die miteinander über eine oder mehrere Relationen verbunden sind. Über die Vorgabe einer Umgebung (Range) wird der Suchraum begrenzt. Die gefundenen Relationen werden gewichtet und damit die semantische Nähe, also der inhaltliche Verwandtschaftsgrad, bestimmt. Das Ergebnis ist damit im besten Fall eine Liste von Konzepten geordnet nach der semantischen Nähe. Durch die Definition eines Zielkonzepts neben dem Startkonzept wird der Suchraum auf alle Relationswege zwischen den beiden Konzepten begrenzt. */

```
Concept[] ConceptResult
Concept[] ConceptVisited
Concept[] StartConcept
Concept[] DestinationConcept
Concept ActiveConcept
Concept LastConcept
int ActiveConceptRange

// Startkonzepte bestimmen
if CDA = sharp
   StartConcept[] = get-concept(CDA)
else
   StartConcept[] = get-concept-unsharp(CDA)

// Zielkonzepte bestimmen
if CDB = sharp
```

```
   DestinationConcept[] = get-concept(CDB)
else
   DestinationConcept[] = get-concept-unsharp(CDB)

// Suche ausführen für alle Startkonzepte, die die Suchanfrage erfüllen
for each Concept ∈ StartConcept[]
   Int StartRange = 0
   ConceptStack Stack = [StartConcept, StartRange]

   // Suche durchführen
   while (Stack not empty)

      // oberstes Konzept des Stacks zur weiteren Suche auswählen, wenn es
      // innerhalb der Range liegt
      repeat
         ActiveConcept = Stack.pop
         LastConcept = Stack.pop
         ActiveConceptRange = Stack.pop
      until ActiveConceptRange < Range

      // Alle Relationen des ausgewählten Konzepts untersuchen
      RelationsActiveConcept[] = getAllRelation(ActiveConcept)
      for i := 1 to RelationsActiveConcept[].max

         // Nur Assoziationen weiterverfolgen
         if RelationsActiveConcept[i].getType = “Assoziation“

            // Wenn das ausgewählte Konzept noch nicht besucht wurde und in der
            // Ergebnismenge liegt, dann das aktuelle Konzept der Ergebnismenge
            // hinzufügen und alle abgehenden Konzepte der Suchmenge hinzufügen
            if (RelationsActiveConcept[i].getName() ∉ ConceptVisited[]) and
               (RelationsActiveConcept[i].getName() ∈ DestinationConcept[])
               Stack.push(Range(ActiveConcept, StartConcept))
               Stack.push(ActiveConcept)
               Stack.push(RelationsActiveConcept[i].getName())

               // Konzeptverknüpfung der Ratingliste hinzufügen, um die Gewichtung
               // bestimmen zu können
               RatingList[].add(ActiveConcept,RatingList[LastConcept] +
                  “Assoziation“)
               ConceptVisited[].add(ActiveConcept)

         // Nur Generalisierungen weiterverfolgen
         if RelationsActiveConcept[i].getType = “Generalisierung“
```

```
    // Wenn das ausgewählte Konzept noch nicht besucht wurde und in der
    // Ergebnismenge liegt, dann das aktuelle Konzept der Ergebnismenge
    // hinzufügen und alle abgehenden Konzepte der Suchmenge hinzufügen
    if (RelationsActiveConcept[i].getName() ∉ ConceptVisited[]) and
       (RelationsActiveConcept[i].getName() ∈ DestinationConcept[])
       Stack.push(Range(ActiveConcept, StartConcept))
       Stack.push(ActiveConcept)
       Stack.push(RelationsActiveConcept[i].getName())

       // Konzeptverknüpfung der Ratingliste hinzufügen, um die Gewichtung
       // bestimmen zu können
       RatingList[].add(ActiveConcept,RatingList[LastConcept] +
          "Generalisierung")
       ConceptVisited[].add(ActiveConcept)

// Nur Kompositionen weiterverfolgen
if RelationsActiveConcept[i].getType = "Komposition"

    // Wenn das ausgewählte Konzept noch nicht besucht wurde und in der
    // Ergebnismenge liegt, dann das aktuelle Konzept der Ergebnismenge
    // hinzufügen und alle abgehenden Konzepte der Suchmenge hinzufügen
    if (RelationsActiveConcept[i].getName() ∉ ConceptVisited[]) and
       (RelationsActiveConcept[i].getName() ∈ DestinationConcept[])
       Stack.push(Range(ActiveConcept, StartConcept))
       Stack.push(ActiveConcept)
       Stack.push(RelationsActiveConcept[i].getName())

       // Konzeptverknüpfung der Ratingliste hinzufügen, um die Gewichtung
       // bestimmen zu können
       RatingList[].add(ActiveConcept,RatingList[LastConcept] +
          "Komposition")
       ConceptVisited[].add(ActiveConcept)

// Nur Aggregationen weiterverfolgen
if RelationsActiveConcept[i].getType = "Aggregation"

    // Wenn das ausgewählte Konzept noch nicht besucht wurde und in der
    // Ergebnismenge liegt, dann das aktuelle Konzept der Ergebnismenge
    // hinzufügen und alle abgehenden Konzepte der Suchmenge hinzufügen
    if (RelationsActiveConcept[i].getName() ∉ ConceptVisited[]) and
       (RelationsActiveConcept[i].getName() ∈ DestinationConcept[])
       Stack.push(Range(ActiveConcept, StartConcept))
       Stack.push(ActiveConcept)
       Stack.push(RelationsActiveConcept[i].getName())
```

```
            // Konzeptverknüpfung der Ratingliste hinzufügen, um die Gewichtung
            // bestimmen zu können
            RatingList[].add(ActiveConcept,RatingList[LastConcept] +
               "Aggregation")
            ConceptVisited[].add(ActiveConcept)

// Alle Ergebniskonzepte, die das Rating nicht erfüllen, aus der Ergebnismenge
// entfernen
Concept[] RatedConcepts = rating(StartConcept,RatingList[],RatingMode)
for i := 1 to RatedConcepts[].max
   if RatedConcepts[i].rating < RatingLimit
      RatedConcepts[].delete(i)
return (RatedConcepts[])
```

Algorithmus von rating(Concept StartConcept, int RatingList, int RatingMode):

/* Diese Funktion ordnet die gefundenen Konzepte der Funktion is-related-with nach deren Gewichtung */

```
// Bewertung nach Inhaltsvergleich
for each (StartConcept.attributes × (Concept.attributes ∈ RatingList[]))
   if StartConcept.attribute matches Concept.attribute
      RatingList[Concept].content =
      RatingList[Concept].content + 1

// 1. Sortierung nach Gewichtung
Sort – Ascending by RatingList[].content

// 2. Sortierung nach Anzahl der Relationen
Group by similar
Sort – Ascending Groups by |Relationscount|

// 3. Sortierung nach RatingMode
Group in Subgroup by similar
Sort – Ascending Subgroup by Semantics according to RatingMode
return (RatingList[])
```

Algorithmus von consists-of(ConceptDescr[] CDA, int Range, int RatingLimit):

/* Diese Funktion fasst die Methoden der scharfen Suche is-associated-with, is-aggregated-of und is-composed-of zusammen, wodurch alle Konzepte gefunden werden, die in irgendeiner direkten Verbindung zueinander stehen. */

```
Concept[] ConceptResult
Concept[] ConceptVisited
Concept[] StartConcept
Concept ActiveConcept
int ActiveConceptRange

// Startkonzepte bestimmen
if CDA = sharp
   StartConcept[] = get-concept(CDA)
else
   StartConcept[] = get-concept-unsharp(CDA)

// Suche ausführen für alle Startkonzepte, die die Suchanfrage erfüllen
for each Concept ∈ StartConcept[]
   Int StartRange = 0
   ConceptStack Stack = [StartConcept, StartRange]

   // Suche durchführen
   while (Stack not Empty)

      // oberstes Konzept des Stacks zur weiteren Suche auswählen, wenn es
      // innerhalb der Range liegt
      repeat
         ActiveConcept = Stack.pop
         ActiveConceptRange = Stack.pop
      until ActiveConceptRange < Range

      // Alle Relationen des ausgewählten Konzepts untersuchen
      RelationsActiveConcept[] = getAllRelation(ActiveConcept)
      for i := 1 to RelationsActiveConcept[].max

         // Nur Assoziationen, Aggregationen und Kompositionen weiterverfolgen
         if (RelationsActiveConcept[i].getType = “Assoziation” or
            RelationsActiveConcept[i].getType = “Aggregation” or
            RelationsActiveConcept[i].getType = “Komposition)

            // Wenn das ausgewählte Konzept noch nicht besucht wurde, dann das
            // aktuelle Konzept der Ergebnismenge hinzufügen und alle abgehenden
            // Konzepte der Suchmenge hinzufügen
            if RelationsActiveConcept[i].getName() ∉ ConceptVisited[]
               Stack.push(Range(ActiveConcept, StartConcept))
               Stack.push(RelationsActiveConcept[i].getName())
               ConceptResult[].add(ActiveConcept)
               ConceptVisited[].add(ActiveConcept)
return (ConceptResult[])
```

Algorithmus von consists-of(ConceptDescr[] CDA, ConceptDescr[] CDB, int Range, int RatingLimit):

/* Diese Funktion fasst die Methoden der scharfen Suche is-associated-with, is-aggregated-of und is-composed-of zusammen, wodurch alle Konzepte gefunden werden, die in irgendeiner direkten Verbindung zueinander stehen. Durch die Definition eines oder mehreren Zielkonzepten neben den Startkonzepten wird der Suchraum auf alle Relationswege zwischen den beiden Konzepten begrenzt. */

```
Concept[] ConceptResult
Concept[] ConceptVisited
Concept[] StartConcept
Concept[] DestinationConcept

// Startkonzepte bestimmen
if CDA = sharp
   StartConcept[] = get-concept(CDA)
else
   StartConcept[] = get-concept-unsharp(CDA)

// Zielkonzepte bestimmen
if CDB = sharp
   DestinationConcept[] = get-concept(CDB)
else
   DestinationConcept[] = get-concept-unsharp(CDB)

// Suche ausführen für alle Startkonzepte, die die Suchanfrage erfüllen
for each Concept ∈ StartConcept[]
   Int StartRange = 0
   ConceptStack Stack = [StartConcept, StartRange]

   // Suche durchführen
   while (Stack not empty)

      // oberstes Konzept des Stacks zur weiteren Suche auswählen, wenn es
      // innerhalb der Range liegt
      repeat
         ActiveConcept = Stack.pop
         ActiveConceptRange = Stack.pop
      until ActiveConceptRange < Range

      // Alle Relationen des ausgewählten Konzepts untersuchen
      RelationsActiveConcept[] = getAllRelation(ActiveConcept)
      for i := 1 to RelationsActiveConcept[].max
```

```
        // Nur Assoziationen, Aggregationen und Kompositionen weiterverfolgen
        if (RelationsActiveConcept[i].getType = “Assoziation” or
              RelationsActiveConcept[i].getType = “Aggregation” or
              RelationsActiveConcept[i].getType = “Komposition”)

           // Wenn das ausgewählte Konzept noch nicht besucht wurde und
           // Bestandteil der Ergebnismenge ist, dann das aktuelle Konzept der
           // Ergebnismenge hinzufügen und alle abgehenden Konzepte der
           // Suchmenge hinzufügen
           if (RelationsActiveConcept[i].getName() ∉ ConceptVisited[]) and
              (RelationsActiveConcept[i].getName() ∈ DestinationConcept[])
              Stack.push(Range(ActiveConcept, StartConcept))
              Stack.push(RelationsActiveConcept[i].getName())
              for each Concept ∈ DestinationConcept[]
                 if ActiveConcept = Concept
                    ConceptResult[].add(ActiveConcept)
                    ConceptVisited[].add(ActiveConcept)
return (ConceptResult[])
```

Algorithmus von is-similar-to(ConceptDescr[] CDA, int Range, int RatingLimit):

/* Diese Funktion sucht zu einem übergebenen Konzept ähnliche Konzepte. Im ersten Schritt werden alle Konzepte anhand ihrer semantischen Beziehung zum übergebenen Konzept gesucht. Anschließend werden die gefundenen Konzepte und das zu vergleichende Konzept in einem Vektorraum sortiert nach der Anzahl ihrer übereinstimmenden Attribute eingetragen. Die Gewichtung der Attribute ergibt sich aus dem Wert der Attribute, falls dieser quantifizierbar ist. Mit dem Cosinus-Maß [Fer 03a] wird die Ähnlichkeit für alle Konzepte bestimmt. Dabei wird ein n-dimensionaler Raum aufgespannt, wobei n die Anzahl der Attribute der zu vergleichenden Konzepte ist. Über die Werte der Attribute jedes Konzepts ist ein Vektor im Raum verbunden. Je kleiner der Winkel zwischen zwei Vektoren ist, desto ähnlicher sind die Konzepte, die durch diese Vektoren repräsentiert sind. */

```
Concept[][] ConceptResult
Concept StartConcept
Concept[] RatedConcepts

// Startkonzepte bestimmen
if CDA = sharp
   StartConcept[] = get-concept(CDA)
else
   StartConcept[] = get-concept-unsharp(CDA)
```

```
// Ergebnismenge bestimmen mit Hilfe der anderen Funktionen
ConceptResult[].add(StartConcept)
ConceptResult[].add(is-associated-with(CDA, Range)
ConceptResult[].add(is-aggregated-of(CDA, Range)
ConceptResult[].add(is-composed-of(CDA, Range)
ConceptResult[].add(is-generalization-of(CDA, Range)

// Attributmatrix für die Cosinus-Maß-Berechnung aufbauen.
int[][] ConceptAttributeCorrelationsMatrix
for i := 1 to ConceptResult[].max
   ConceptAttributCorrelationsMatrix[i][].add(getAllAttributes(ConceptResult(i)))

   // Berechnung der einzelnen Winkel der Konzeptvektoren durchführen
   RatedConcepts = calculateConsinusValue(ConceptAttributeCorrelationsMatrix)

// Alle Ergebniskonzepte, die das Rating nicht erfüllen aus der Ergebnismenge
// entfernen
for i := 1 to RatedConcepts[].max
   if ConceptResult[i].rating < RatingLimit
   ConceptResult[].delete[i]
return (ConceptResult[])
```

D.2 Monitor-Agent

In Bild 48 ist das Klassendiagramm für die Überwachungsbedingung, das die Grundlage der Problemlösungskomponente des Monitor-Agenten ist, dargestellt. Die Anfragesprache aus Anhang C.3 spiegelt exakt die Klassenstruktur der Implementierung wieder. An zentraler Stelle steht die Klasse *Expr*, sie ist die Basisklasse, von der aus die Überwachungsbedingung aufgebaut wird. An sie ist entweder eine der Klassen *StructureFunc1* bis *StructureFunc5*, die die Strukturfunktionen repräsentieren, die Not-Funktion (*expr*), eine der Klasse *Value* und *ValueString* für die Value-Funktionen mit dem *Operator1* oder der aussagenlogische *Operator2* mit den zwei Teilausdrücken *expr1* und *expr2* gebunden. Die Strukturfunktionen besitzen ebenso wie die Value-Funktionen und die Access-Funktionen als Parameter Informationseinheiten aus dem ASN, die durch die entsprechenden Klassen *ValueAttribute*, *ValueConcept*, *ValueNet* und *ValueRelation* repräsentiert werden.

Die Attribute *structurefunc* der Strukturfunktionen, *valuetype* der Value-Funktionen, sowie *accessfunc* und *mathfunc* der Klassen *AccessFunc* und *MathFunc* geben Auskunft über die Art der Funktion und damit auch wie die Funktion innerhalb der Überwachungsbedingung auszuwerten ist. So kann z. B. das Attribut *structurefunc* der Klasse *StructureFunc1* die Werte *changeversion*, *changeattributevalue* und *deleteattribute* annehmen.

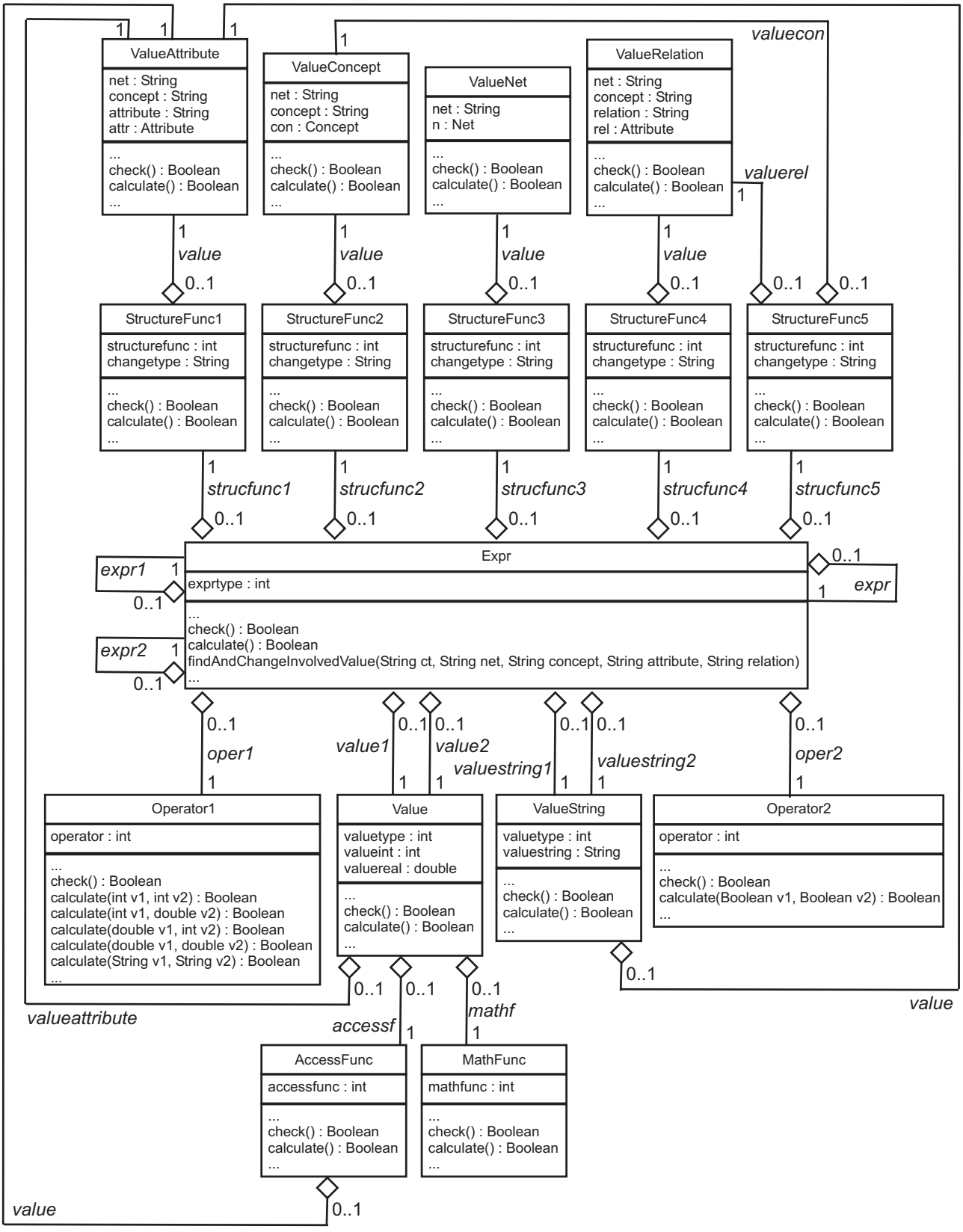


Bild 48: Klassendiagramm des Monitor-Agenten

Jede Klasse besitzt eine Methode *check* und eine Methode *calculate*. Ausgehend von der Klasse *expr* verifiziert rekursiv die Methode *check* die Syntax der Überwachungsbedingung, während durch die Methode *calculate* rekursiv die Bedingung bei jeder Änderung neu berechnet wird.

Die Methoden *check* der Klassen *ValueAttribute*, *ValueConcept*, *ValueNet* und *ValueRelation* überprüfen neben der Syntax auch ob das entsprechende Informationselement im ASN vorhanden ist. Die Methode *check* wird einmal beim Empfang der Monitoranfrage ausgewertet. Dies bedeutet, beinhaltet z. B. die Überwachungsbedingung den Test, ob ein bestimmtes Attribut gelöscht wird, so ist die Bedingung semantisch nur dann korrekt, wenn das Attribut beim Start des Monitor-Agenten vorhanden war. Des Weiteren wird der Teil der Überwachungsbedingung, der das Attribut testet, nach dem Löschen des Attributs nie wieder falsch, da es nicht möglich ist, das physisch selbe Attribut erneut anzulegen (siehe Abschnitt 8.7.2).

Die unterschiedlichen Ausprägungen der Methode *calculate* in der Klasse *Operator1* kommen durch die unterschiedlichen Typen von Parametern in der Klasse *Value* zustande. So werden Integer-Werte mit Integer- bzw. Real-Werten genauso auf ihre mathematische Ordnung, während String-Werte auf ihre alphanumerische Ordnung verglichen werden.

Für die Klassen *StructureFunc1* bis *StructureFunc5*, welche die Strukturfunktionen symbolisieren, ist es, um die Veränderung im ASN richtig deuten zu können, notwendig den vorherigen Wert des Informationselements gespeichert zu haben. Diese Speicherung und den Vergleich von neuem mit gespeichertem Wert nimmt die Methode *findAndChangeInvolvedValue* vor.

D.3 Koordinations-Agent

In Bild 49 ist das Klassendiagramm des endlichen Automaten, der die Grundlage des Problemlösungsmoduls des Koordinations-Agenten ist, abgebildet. Der endliche Automat besteht zum einen aus der Klasse *State Transition Diagram (STD)* die den Rahmen des endlichen Automaten darstellt und zum anderen aus den Klassen *State* und *Transition*. Die Klasse *State* repräsentiert dabei einen Zustand des endlichen Automaten und die Klasse *Transition* den Zustandsübergang.

Eine Instanz der Klasse *STD* muss dabei aus genau einem Startzustand (*Startstate*), mindestens einem Übergangszustand (*Normalstate*) und mindestens einem Endzustand (*Endstate*) bestehen. Des Weiteren muss ein Weg zwischen dem Startzustand und mindestens einem Endzustand vorhanden sein. Die erste Bedingung, das Vorhandensein der einzelnen Zustände, wird durch die Methode *check* der Klasse *STD* beim Empfang des Koordinationsprotokolls überprüft. Die Methode *check* der Klasse *State* und *Transition* überprüft die Vollständigkeit der Angaben über einen Zustand oder einen Zustandsübergang. Die zweite Bedingung, einen möglichen Weg zwischen Start- und Endzustand zu finden, wird durch die Methode

findPath der Klasse *STD* verifiziert. Die Methode *findPath* untersucht dabei rekursiv vom Startzustand ausgehend alle möglichen Wege bis zu einem Endzustand.

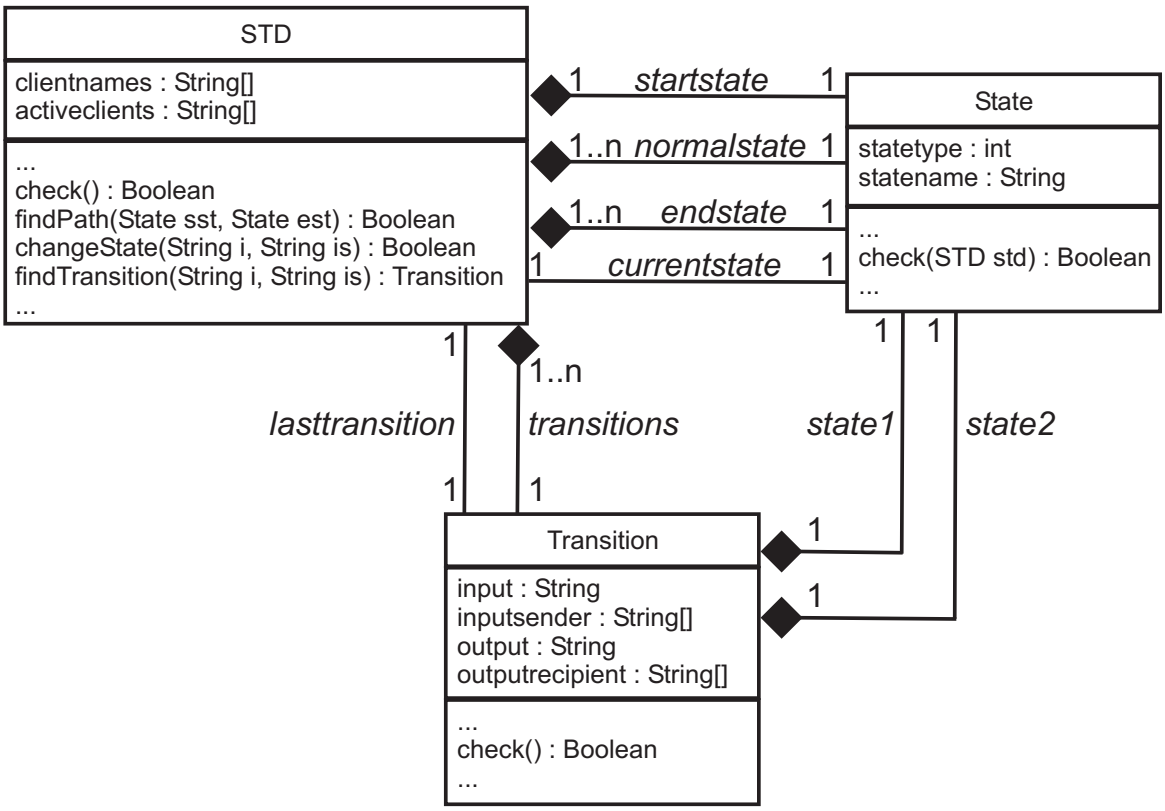


Bild 49: Klassendiagramm des Koordinations-Agenten

Mit der Methode *changeState* der Klasse *STD* wird ein Zustandsübergang eingeleitet. Dafür erhält die Methode als Parameter *i* die Eingabenachricht und als *is* den Sendernamen. Anschließend wählt sie mit Hilfe der Methode *findTransition* den richtigen Zustandsübergang aus.

E Nachrichtenformate

Im Folgenden sind die zwischen RPD-Anwendung und RPD-Middleware ausgetauschten Nachrichten als XML-DTDs angegeben. Für jede Nachricht ist sowohl beschrieben zu welchem Zeitpunkt innerhalb der Kommunikation sie zum Einsatz kommt, als auch auf welche Nachricht sie die Antwort ist bzw. welche Antwort-Nachrichten möglich sind.

E.1 Retrieval-Agent

Retrieval-Request:

Mit dieser Nachricht fordert die RPD-Anwendung bei der RPD-Middleware einen neuen Retrieval-Agenten an. Diese Nachricht wird vom neuen Retrieval-Agenten mit der Nachricht *Retrieval-Acknowledge* quittiert.

```
<!ELEMENT retrieval-request (sender, destination)>
<!ELEMENT sender (#PCDATA)>
<!ELEMENT destination (#PCDATA)>
```

Retrieval-Acknowledge:

Diese Nachricht ist die Antwort auf die Nachricht *Retrieval-Request*. Sie informiert die RPD-Anwendung über die Instanziierung des Retrieval-Agenten.

```
<!ELEMENT retrieval-ack (sender, destination)>
<!ELEMENT sender (#PCDATA)>
<!ELEMENT destination (#PCDATA)>
```

Retrieval-Order:

Diese Nachricht wird von der RPD-Anwendung an den Retrieval-Agenten geschickt. Sie beinhaltet als Information die Suchanfrage für den Retrieval-Agenten (siehe Anhang C.1). Die Nachricht wird entweder mit der Nachricht *Retrieval-Order-Acknowledge* quittiert, wenn die Suchanfrage syntaktisch korrekt war. Ist dies nicht der Fall, wird dies der RPD-Anwendung durch die Nachricht *Retrieval-Order-Notacknowledge* mitgeteilt.

```
<!ELEMENT retrieval-order (sender, destination, retrieval)>
<!ELEMENT sender (#PCDATA)>
<!ELEMENT destination (#PCDATA)>
<!ELEMENT retrieval (#PCDATA)>
```

Retrieval-Order-Acknowledge:

Diese Nachricht bestätigt der RPD-Anwendung die korrekte Anfrage. Sie ist die Antwort auf die Nachricht *Retrieval-Order*.

```
<!ELEMENT retrieval-order-ack (sender, destination)>
<!ELEMENT sender (#PCDATA)>
<!ELEMENT destination (#PCDATA)>
```

Retrieval-Order-Notacknowledge:

Diese Nachricht zeigt der RPD-Anwendung an, dass die Suchanfrage syntaktisch nicht korrekt ist. Sie ist die Antwort auf die Nachricht *Retrieval-Order*.

```
<!ELEMENT retrieval-order-nack (sender, destination)>
<!ELEMENT sender (#PCDATA)>
<!ELEMENT destination (#PCDATA)>
```

Retrieval-Result:

Mit dieser Nachricht wird der RPD-Anwendung das Ergebnis des Retrieval-Agenten mitgeteilt. Die Nachricht ist damit die Antwort auf die Nachricht *Retrieval-Order*.

```
<!ELEMENT retrieval-result (sender, destination, result)>
<!ELEMENT sender (#PCDATA)>
<!ELEMENT destination (#PCDATA)>
<!ELEMENT result (#PCDATA | concept)*>
<!ATTLIST result
   queryId CDATA #REQUIRED>
<!ELEMENT attribute (#PCDATA)>
<!ATTLIST attribute
   name CDATA #REQUIRED
   type CDATA #REQUIRED
   id CDATA #IMPLIED>
<!ELEMENT concept (searchinfo, attributes, relations)>
<!ATTLIST concept
   name CDATA #REQUIRED
   type CDATA #REQUIRED
   id CDATA #IMPLIED>
<!ELEMENT relation (#PCDATA)>
<!ATTLIST relation
   name CDATA #REQUIRED
   multiplicity CDATA #REQUIRED
   direction CDATA #REQUIRED
   type CDATA #REQUIRED>
```

```
<!ELEMENT attributes (attribute)*>
<!ELEMENT relations (relation)*>
<!ELEMENT searchinfo>
<!ATTLIST searchinfo
   rating CDATA #REQUIRED
   retrievedby CDATA #REQUIRED
   foundrelatedto CDATA #REQUIRED
   direction CDATA #REQUIRED>
```

Retrieval-Stop:

Mit dieser Nachricht kann eine RPD-Anwendung den Retrieval-Agenten beenden.

```
<!ELEMENT retrieval-stop (sender, destination)>
<!ELEMENT sender (#PCDATA)>
<!ELEMENT destination (#PCDATA)>
```

E.2 Aggregations-Agent

Aggregation-Request:

Mit dieser Nachricht fordert die RPD-Anwendung bei der RPD-Middleware einen neuen Aggregations-Agenten an. Diese Nachricht wird vom neuen Aggregations-Agenten mit der Nachricht *Aggregation-Acknowledge* quittiert.

```
<!ELEMENT aggregation-request (sender, destination)>
<!ELEMENT sender (#PCDATA)>
<!ELEMENT destination (#PCDATA)>
```

Aggregation-Acknowledge:

Diese Nachricht ist die Antwort auf die Nachricht *Aggregation-Request*. Sie informiert die RPD-Anwendung über die Instanziierung des Aggregations-Agenten.

```
<!ELEMENT aggregation-ack (sender, destination)>
<!ELEMENT sender (#PCDATA)>
<!ELEMENT destination (#PCDATA)>
```

Aggregation-Order:

Diese Nachricht wird von der RPD-Anwendung an den Aggregations-Agenten geschickt. Sie beinhaltet als Information die Aggregationsanfrage für den Aggregations-Agenten, die alle zu aggregierenden Anfragen an die Retrieval-Agenten einschließen (siehe Anhang C.2). Die Nachricht wird entweder mit der Nachricht *Aggregation-Order-Acknowledge* quittiert, wenn die Aggregationsanfrage syntaktisch

korrekt war. Andernfalls wird dies der RPD-Anwendung durch die Nachricht *Aggregation-Order-Notacknowledge* mitgeteilt.

```
<!ELEMENT aggregation-order (sender, destination, aggregation)>
<!ELEMENT sender (#PCDATA)>
<!ELEMENT destination (#PCDATA)>
<!ELEMENT aggregation (element)*>
<!ATTLIST aggregation
   name CDATA #REQUIRED
   type CDATA #REQUIRED
   context CDATA #IMPLIED>
<!ELEMENT element (query)*>
<!ATTLIST element
   name CDATA #REQUIRED
   type CDATA #REQUIRED>
<!ELEMENT query (querySource)*>
<!ATTLIST query
   id CDATA #REQUIRED
   name CDATA #IMPLIED
   resultType CDATA #REQUIRED>
<!ELEMENT querySource (#PCDATA)>
<!ATTLIST querySource
   queryType CDATA #REQUIRED>
```

Aggregation-Order-Acknowledge:

Diese Nachricht bestätigt der RPD-Anwendung die korrekte Anfrage. Sie ist die Antwort auf die Nachricht *Aggregation-Order*.

```
<!ELEMENT aggregation-order-ack (sender, destination)>
<!ELEMENT sender (#PCDATA)>
<!ELEMENT destination (#PCDATA)>
```

Aggregation-Order-Notacknowledge:

Diese Nachricht zeigt der RPD-Anwendung an, dass die Aggregationsanfrage syntaktisch nicht korrekt ist. Sie ist die Antwort auf die Nachricht *Aggregation-Order*.

```
<!ELEMENT aggregation-order-nack (sender, destination)>
<!ELEMENT sender (#PCDATA)>
<!ELEMENT destination (#PCDATA)>
```

Aggregation-Result:

Mit dieser Nachricht wird der RPD-Anwendung das Ergebnis des Aggregations-Agenten mitgeteilt. Die Nachricht ist damit die Antwort auf die Nachricht *Aggregation-Order*.

```
<!ELEMENT aggregation-result (sender, destination, aggregation)>
<!ELEMENT sender (#PCDATA)>
<!ELEMENT destination (#PCDATA)>
<!ELEMENT aggregation (element)*>
<!ATTLIST aggregation
   name CDATA #REQUIRED
   type CDATA #REQUIRED
   context CDATA #IMPLIED>
<!ELEMENT element (result)*>
<!ATTLIST element
   name CDATA #REQUIRED
   type CDATA #REQUIRED>
<!ELEMENT result (#PCDATA | concept)*>
<!ATTLIST result
   queryId CDATA #REQUIRED>
<!ELEMENT attribute (#PCDATA)>
<!ATTLIST attribute
   name CDATA #REQUIRED
   type CDATA #REQUIRED
   id CDATA #IMPLIED>
<!ELEMENT concept (searchinfo, attributes, relations)>
<!ATTLIST concept
   name CDATA #REQUIRED
   type CDATA #REQUIRED
   id CDATA #IMPLIED>
<!ELEMENT relation (#PCDATA)>
<!ATTLIST relation
   name CDATA #REQUIRED
   multiplicity CDATA #REQUIRED
   direction CDATA #REQUIRED
   type CDATA #REQUIRED>
<!ELEMENT attributes (attribute)*>
<!ELEMENT relations (relation)*>
<!ELEMENT searchinfo>
<!ATTLIST searchinfo
   rating CDATA #REQUIRED
   retrievedby CDATA #REQUIRED
   foundrelatedto CDATA #REQUIRED
   direction CDATA #REQUIRED>
```

Aggregation-Stop:

Mit dieser Nachricht kann eine RPD-Anwendung den Aggregations-Agenten beenden.

<!ELEMENT aggregation-stop (sender, destination)>
<!ELEMENT sender (#PCDATA)>
<!ELEMENT destination (#PCDATA)>

E.3 Monitor-Agent

Monitor-Request:

Mit dieser Nachricht fordert die RPD-Anwendung bei der RPD-Middleware einen neuen Monitor-Agenten an. Diese Nachricht wird vom neuen Monitor-Agenten mit der Nachricht *Monitor-Acknowledge* quittiert.

<!ELEMENT monitor-request (sender, destination)>
<!ELEMENT sender (#PCDATA)>
<!ELEMENT destination (#PCDATA)>

Monitor-Acknowledge:

Diese Nachricht ist die Antwort auf die Nachricht *Monitor-Request*. Sie teilt der RPD-Anwendung das Vorhandensein des Monitor-Agenten mit.

<!ELEMENT monitor-ack (sender, destination)>
<!ELEMENT sender (#PCDATA)>
<!ELEMENT destination (#PCDATA)>

Monitor-Order:

Diese Nachricht wird von der RPD-Anwendung an den Monitor-Agenten geschickt (siehe Anhang C.3). Sie beinhaltet als Information die Überwachungsbedingung für den Monitor-Agenten. Die Nachricht wird entweder mit der Nachricht *Monitor-Order-Acknowledge* quittiert, wenn die Anfrage korrekt war, d. h. wenn die zu überwachenden Elemente im ASN vorhanden sind. Ist dies nicht der Fall, wird die RPD-Anwendung durch die Nachricht *Monitor-Order-Notacknowledge* informiert.

<!ELEMENT monitor-order (sender, destination, expr)>
<!ELEMENT sender (#PCDATA)>
<!ELEMENT destination (#PCDATA)>
<!ELEMENT expr (operator, parameter, parameter)>
<!ELEMENT operator (#PCDATA)>
<!ELEMENT parameter (net, concept, attribute)>

<!ELEMENT parameter (net, concept, relation)>
<!ELEMENT parameter (net, concept)>
<!ELEMENT parameter (net)>
<!ELEMENT parameter (expr)>
<!ELEMENT parameter (integer)>
<!ELEMENT parameter (real)>
<!ELEMENT parameter (string)>
<!ELEMENT parameter (function)>
<!ELEMENT net (#PCDATA)>
<!ELEMENT concept (#PCDATA)>
<!ELEMENT attribute (#PCDATA)>
<!ELEMENT integer (#PCDATA)>
<!ELEMENT real (#PCDATA)>
<!ELEMENT string (#PCDATA)>
<!ELEMENT function (typ, funcparameter, funcparameter?)>
<!ELEMENT funcparameter (net, concept, attribute)>
<!ELEMENT funcparameter (integer)>
<!ELEMENT funcparameter (real)>
<!ELEMENT funcparameter (function)>
<!ELEMENT typ (#PCDATA)>

Monitor-Order-Acknowledge:

Diese Nachricht bestätigt der RPD-Anwendung die korrekte Anfrage. Sie ist die Antwort auf die Nachricht *Monitor-Order*.

<!ELEMENT monitor-order-ack (sender, destination)>
<!ELEMENT sender (#PCDATA)>
<!ELEMENT destination (#PCDATA)>

Monitor-Order-Notacknowledge:

Diese Nachricht zeigt der RPD-Anwendung an, dass die Überwachungsbedingung nicht korrekt ist. Sie ist die Antwort auf die Nachricht *Monitor-Order*.

<!ELEMENT monitor-order-nack (sender, destination)>
<!ELEMENT sender (#PCDATA)>
<!ELEMENT destination (#PCDATA)>

Monitor-Condition-True:

Die Nachricht *Monitor-Condition-True* wird vom Monitor-Agenten an die RPD-Anwendung geschickt, wenn die Überwachungsbedingung ihren Zustand von *falsch* nach *wahr* geändert hat.

<!ELEMENT monitor-condition-true (sender, destination)>
<!ELEMENT sender (#PCDATA)>
<!ELEMENT destination (#PCDATA)>

Monitor-Condition-False:

Die Nachricht *Monitor-Condition-False* wird vom Monitor-Agenten an die RPD-Anwendung geschickt, wenn die Überwachungsbedingung ihren Zustand von *wahr* nach *falsch* geändert hat.

<!ELEMENT monitor-condition-false (sender, destination)>
<!ELEMENT sender (#PCDATA)>
<!ELEMENT destination (#PCDATA)>

Monitor-Stop:

Mit dieser Nachricht kann eine RPD-Anwendung jederzeit den Monitor-Agenten beenden.

<!ELEMENT monitor-stop (sender, destination)>
<!ELEMENT sender (#PCDATA)>
<!ELEMENT destination (#PCDATA)>

ASN-Down:

Diese Nachricht zeigt der RPD-Anwendung an, dass das ASN derzeit nicht verfügbar ist und daher der Monitor-Agent auch nicht seine Arbeit aufnehmen bzw. fortsetzen kann.

<!ELEMENT monitor-asn-down (sender, destination)>
<!ELEMENT sender (#PCDATA)>
<!ELEMENT destination (#PCDATA)>

ASN-Change:

Mit dieser Nachricht informiert das ASN den Monitor-Agenten über die aufgetretene Veränderung.

<!ELEMENT monitor-asn-change (sender, destination, changetype, netname, conceptname, relationname, attributename)>
<!ELEMENT sender (#PCDATA)>
<!ELEMENT destination (#PCDATA)>
<!ELEMENT changetype (#PCDATA)>
<!ELEMENT netname (#PCDATA)>
<!ELEMENT conceptname (#PCDATA)>

<!ELEMENT relationname (#PCDATA)>
<!ELEMENT attributename (#PCDATA)>

E.4 Koordinations-Agent

Initiate-Coordination-Agent:

Mit dieser Nachricht wird ein neuer Koordinations-Agent erzeugt. Diese Nachricht wird vom Master-Agenten empfangen, worauf ein Koordinations-Agent instanziiert und diesem diese Nachricht zur weiteren Bearbeitung zugesandt wird.

<!ELEMENT initiate-coordination-agent (sender, destination, name, fsmname)>
<!ELEMENT sender (#PCDATA)>
<!ELEMENT destination (#PCDATA)>
<!ELEMENT name (#PCDATA)>
<!ELEMENT fsmname (#PCDATA)>

Initiate-Coordination-Agent-Acknowledge:

Die Nachricht ist die Antwort des neuen Koordinations-Agenten an die anfordernde RPD-Anwendung. Sie ist die direkte Quittierung der Nachricht *Initiate-Coordination-Agent*.

<!ELEMENT initiate-coordination-agent-ack (sender, destination, name, fsmname)>
<!ELEMENT sender (#PCDATA)>
<!ELEMENT destination (#PCDATA)>
<!ELEMENT name (#PCDATA)>
<!ELEMENT fsmname (#PCDATA)>

Request-Coordination-Agent:

Die Nachricht wird von den RPD-Anwendungen, die an der Koordination beteiligt sind, aber nicht das Koordinationsprotokoll festgelegt haben, an die RPD-Middleware geschickt. Damit wird dem Koordinations-Agenten angezeigt, dass diese RPD-Anwendung an der Koordination teilnimmt.

<!ELEMENT request-coordination-agent (sender, destination, name, fsmname)>
<!ELEMENT sender (#PCDATA)>
<!ELEMENT destination (#PCDATA)>
<!ELEMENT name (#PCDATA)>
<!ELEMENT fsmname (#PCDATA)>

Request-Coordination-Agent-Acknowledge:

Diese Nachricht wird als Antwort auf die Nachricht *Request-Coordination-Agent* an die RPD-Anwendung übermittelt. Sie dient als Bestätigung für die anfragende RPD-Anwendung, womit dieser angezeigt wird, dass der entsprechende Koordinations-Agent existiert. Dies ist insofern von Bedeutung, da der Koordinations-Agent erst dann *Request-Coordination-Agent*-Nachrichten empfangen kann, wenn er durch eine RPD-Anwendung mit Hilfe der Nachricht *Initiate-Coordination-Agent* instanziiert wurde.

<!ELEMENT request-coordination-agent-ack (sender, destination, name, fsmname)>
<!ELEMENT sender (#PCDATA)>
<!ELEMENT destination (#PCDATA)>
<!ELEMENT name (#PCDATA)>
<!ELEMENT fsmname (#PCDATA)>

FSM-Definition:

Mit dieser Nachricht wird das Zustandübergangsdiagramm des endlichen Automaten in der in Anhang C.4 beschriebenen Form durch die RPD-Anwendung definiert und dem Koordinations-Agenten übermittelt.

<!ELEMENT fsm-definition (sender, destination, std)>
<!ELEMENT sender (#PCDATA)>
<!ELEMENT destination (#PCDATA)>
<!ELEMENT std (startstate, normalstate?, endstate, endstate?, transition, transition?)>
<!ELEMENT startstate (#PCDATA)>
<!ELEMENT normalstate (#PCDATA)>
<!ELEMENT endstate (#PCDATA)>
<!ELEMENT transition (state1, state2, input, output)>
<!ELEMENT state1 (#PCDATA)>
<!ELEMENT state2 (#PCDATA)>
<!ELEMENT input (info, sender)>
<!ELEMENT output (info, recipient)>
<!ELEMENT info (#PCDATA)>
<!ELEMENT sender (#PCDATA)>
<!ELEMENT recipient (#PCDATA)>

FSM-Definition-Acknowledge:

Mit dieser Nachricht wird der RPD-Anwendung der Empfang des Zustandübergangsdiagramms quittiert. Es ist die direkte Antwort auf die Nachricht *FSM-Definition*.

<!ELEMENT fsm-definition-ack (sender, destination)>
<!ELEMENT sender (#PCDATA)>
<!ELEMENT destination (#PCDATA)>

FSM-Definition-Notacknowledge:

Mit dieser Nachricht wird der RPD-Anwendung mitgeteilt, dass das Zustandübergangsdiagramm syntaktisch und / oder semantisch nicht korrekt ist. Es ist die direkte Antwort auf die Nachricht *FSM-Definition*.

<!ELEMENT fsm-definition-nack (sender, destination)>
<!ELEMENT sender (#PCDATA)>
<!ELEMENT destination (#PCDATA)>

Request-FSM:

Diese Nachricht dient zur Abfrage des Zustandübergangsdiagramms. Mit Hilfe dieser Nachrichten können die Koordinationspartner, die nicht den Automaten definiert haben, diesen zur weiteren internen Bearbeitung und Auswertung vom Koordinations-Agenten erfragen.

<!ELEMENT request-fsm (sender, destination)>
<!ELEMENT sender (#PCDATA)>
<!ELEMENT destination (#PCDATA)>

FSM:

Diese Nachricht enthält das Zustandübergangsdiagramm und ist die Antwort des Koordinations-Agenten auf die Nachricht *Request-FSM*. Auch hier wird auf die in Anhang C.4 angegebene Sprachdefinition zurückgegriffen.

<!ELEMENT fsm (sender, destination, std)>
<!ELEMENT sender (#PCDATA)>
<!ELEMENT destination (#PCDATA)>
<!ELEMENT std (startstate, normalstate?, endstate, endstate?, transition,
transition?)>
<!ELEMENT startstate (#PCDATA)>
<!ELEMENT normalstate (#PCDATA)>
<!ELEMENT endstate (#PCDATA)>
<!ELEMENT transition (state1, state2, input, output)>
<!ELEMENT state1 (#PCDATA)>
<!ELEMENT state2 (#PCDATA)>
<!ELEMENT input (info, sender)>
<!ELEMENT output (info, recipient)>
<!ELEMENT info (#PCDATA)>

<!ELEMENT sender (#PCDATA)>
<!ELEMENT recipient (#PCDATA)>

Input:

Diese Nachricht erzwingt einen Zustandsübergang. Ist im aktuellen Zustand die Eingabe unzulässig, beispielsweise weil der Sender keine Eingabenachricht mit diesem Text verschicken darf, so erhält der Sender dies als Fehler (*Wrong-Input*) mitgeteilt. Über den erfolgreichen Zustandswechsel wird der Sender durch die Nachricht *Input-Acknowledge* informiert.

<!ELEMENT input (sender, destination, info)>
<!ELEMENT sender (#PCDATA)>
<!ELEMENT destination (#PCDATA)>
<!ELEMENT info (#PCDATA)>

Input-Acknowledge:

Diese Nachricht wird an den Sender einer Eingabenachricht als Empfangsbestätigung verschickt. Das ist wichtig, da der Sender nicht zwingend in der Liste der zu empfangenden Ausgabenachrichten enthalten ist und daher nicht unbedingt eine Quittierung in Form einer *Output*-Nachricht erhält.

<!ELEMENT input-ack (sender, destination, info)>
<!ELEMENT sender (#PCDATA)>
<!ELEMENT destination (#PCDATA)>
<!ELEMENT info (#PCDATA)>

Wrong-Input:

Diese Nachricht wird verschickt, wenn die Eingabe im aktuellen Zustand unzulässig ist. Dies ist dann der Fall, wenn entweder kein Zustandsübergang existiert, der denselben Eingabetext besitzt wie die Inputnachricht oder wenn der Sender nicht in der Liste der möglichen Sender des Zustandübergangs definiert ist.

<!ELEMENT wrong-input (sender, destination, info, state)>
<!ELEMENT sender (#PCDATA)>
<!ELEMENT destination (#PCDATA)>
<!ELEMENT info (#PCDATA)>
<!ELEMENT state (#PCDATA)>

Output:

Diese Nachricht wird beim Zustandsübergang erzwungen. Sie enthält die Ausgabenachricht und den neuen Zustandsnamen. Die Nachricht wird nur an die Koordinationspartner verschickt, die in der Liste der Empfänger vermerkt sind.

<!ELEMENT output (sender, destination, info, state)>
<!ELEMENT sender (#PCDATA)>
<!ELEMENT destination (#PCDATA)>
<!ELEMENT info (#PCDATA)>
<!ELEMENT state (#PCDATA)>

End-State:

Die Nachricht *End-State* wird verschickt, wenn ein Endzustand des Koordinationsprotokolls erreicht wurde.

<!ELEMENT end-state (sender, destination, info, state)>
<!ELEMENT sender (#PCDATA)>
<!ELEMENT destination (#PCDATA)>
<!ELEMENT info (#PCDATA)>
<!ELEMENT state (#PCDATA)>

Request-State:

Mit dieser Nachricht kann die RPD-Anwendung den aktuellen Zustand beim Koordinations-Agenten abfragen und erhält als Antwort die *Current-State*-Nachricht.

<!ELEMENT request (sender, destination)>
<!ELEMENT sender (#PCDATA)>
<!ELEMENT destination (#PCDATA)>

Current-State:

Diese Nachricht übermittelt den angefragten, aktuellen Zustand an die RPD-Anwendung und ist die Antwort auf die *Request-State*-Nachricht.

<!ELEMENT current-state (sender, destination, state)>
<!ELEMENT sender (#PCDATA)>
<!ELEMENT destination (#PCDATA)>
<!ELEMENT state (#PCDATA)>

F Nutzung der RPD-Middleware anhand des Beispiels: Entwicklung einer Luftdüse

F.1 Retrieval-Agent

F.1.1 Retrievalanfrage und -ergebnis (XML-Notation)

Für das Anwendungsbeispiel aus Abschnitt 8.1 ist im Folgenden exemplarisch die Anfrage und das Ergebnis des Retrieval-Agenten in der XML-Notation dargestellt.

Anfrage an den Retrieval-Agenten

```
<retrieval-order>
   <sender>Projektplaner/LocalAgent/PP</sender>
   <destination>RetrievalTeam/RetrievalAgent/35</destination>
   <retrieval>
      welche bestehtaus(name = Entwicklung, type = Mitarbeiter, 1)
      funktion = Teamleiter -
   </retrieval>
</retrieval-order>
```

Ergebnis des Retrieval-Agenten

```
<retrieval-result>
   <sender> RetrievalTeamInfo/RetrievalAgent/35</sender>
   <destination>Projektplaner/LocalAgent/PP</destination>
   <result queryId="1">
      <concept name="Warschat" type="Mitarbeiter" id="145">
         <searchinfo rating="0" retrievedby="bestehtaus"
         foundrelatedto="Entwicklung" direction="up" />
         <attributes>
            <attribute name="nachname">Warschat</attribute>
            <attribute name="vorname">Joachim</attribute>
            <attribute name="teamrolle"></attribute>
            <attribute name="funktion">Teamleiter</attribute>
            <attribute name="personalkosten">leer</attribute>
         </attributes>
         <relations>
            <relation name="geschaeftsadresse" multiplicity="one"
               direction="Mitarbeiter" type="Aggregation">
               Geschaeftsadresse Warschat</relation>
            <relation name="teamzugehoerigkeit" multiplicity="one"
               direction="Team" type="Aggregation">Entwicklung</relation>
            <relation name="abtzugehoerigkeit" multiplicity="one"
```

```
                direction="ungerichtet" type="Assoziation">IAT</relation>
            </relations>
        </concept>
    </result>
</retrieval-result>
```

F.1.2 Retrievalanfrage und -ergebnis (graphischer Client)

In Bild 50 ist die Anfrage an den Retrieval-Agenten unter Zuhilfenahme des graphischen Clients für den Retrieval-Agenten abgebildet. Im Texteingabefeld (1) des Clients wird die Anfrage in der ASN-QL formuliert. Im Eingabefeld (2) kann die UAI des Retrieval-Agenten angegeben werden, dadurch können auch mehrere Anfragen nacheinander an ein und denselben Retrieval-Agenten gestellt werden. Durch Klicken auf den Button Send (3) wird ein Request an die RPD-Middleware gestellt und der Retrieval-Agent instanziiert. Über die Menüleiste (4) wird der Client gesteuert.

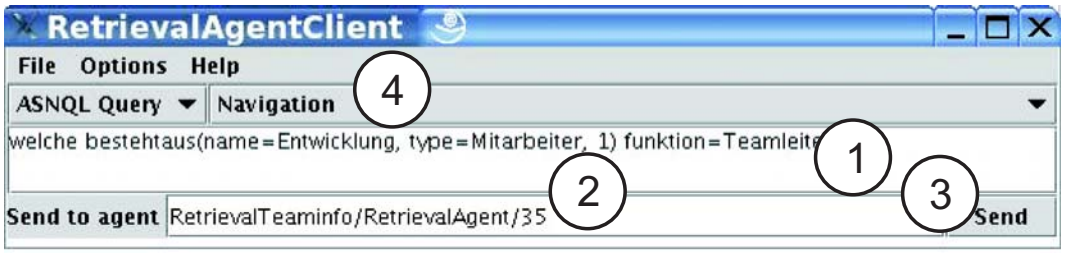


Bild 50: Screenshot: Anfrage an den Retrieval-Agent

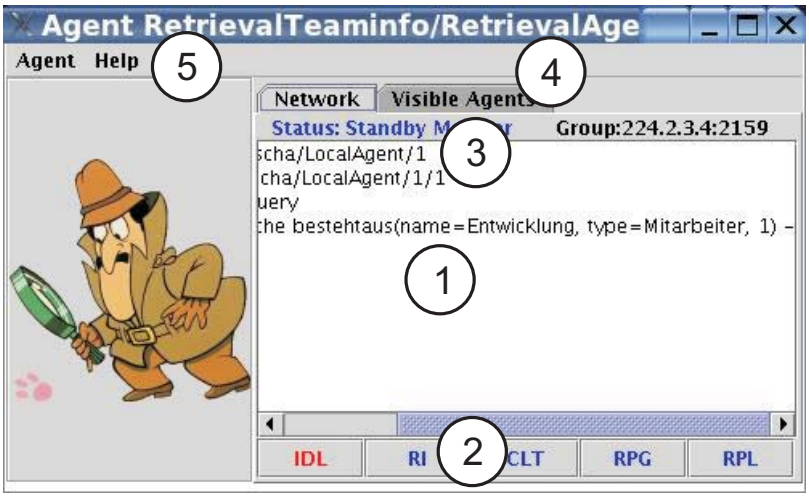


Bild 51: Screenshot: Retrieval-Agent

Der Screenshot Bild 51 zeigt das Informationsfenster des Retrieval-Agenten. Im Textfeld (1) ist die Anfrage des Clients zu sehen. Die Anzeigeelemente (2) zeigen den aktuellen Zustand des Agenten, den er innerhalb des Master-Agenten-Prozesses inne hat (siehe Abschnitt 5.4). In der Statusleiste (3) ist ersichtlich, dass der Agent im Moment Standby-Master ist und als Multicast-Adresse die IP

224.2.3.4 mit dem Port 2159 verwendet wird. Auf der zweiten Tab-Seite (4) sind die aktuell im Agentensystem befindlichen Agenten verzeichnet. Über die Menüleiste (5) kann der Agent bei Bedarf ausgeblendet oder zwangsweise beendet werden.

Das zurück gelieferte Ergebnis ist in Bild 52 in der XML-Notation (1) nach Anhang E.1 dargestellt.

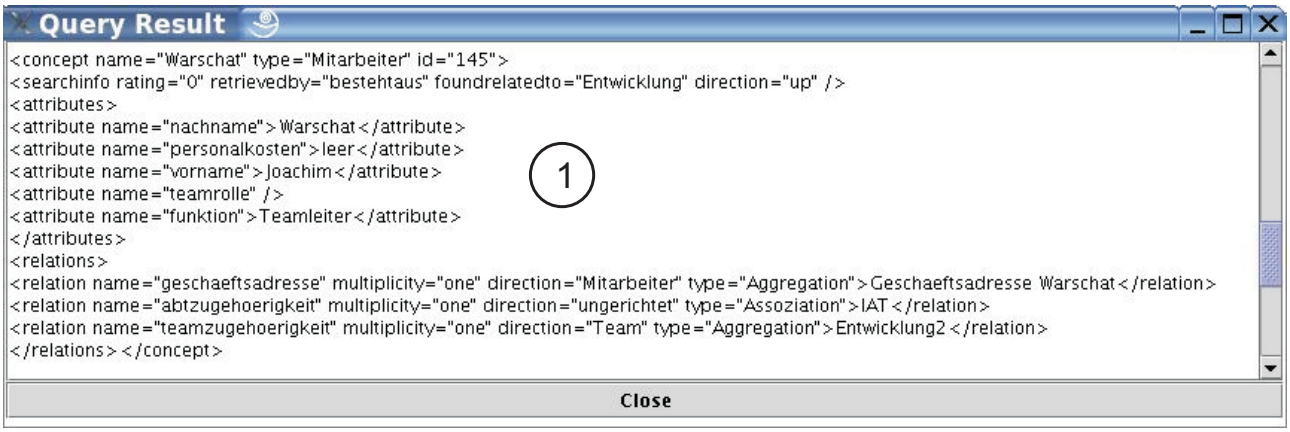


Bild 52: Screenshot: Ergebnis des Retrieval-Agenten

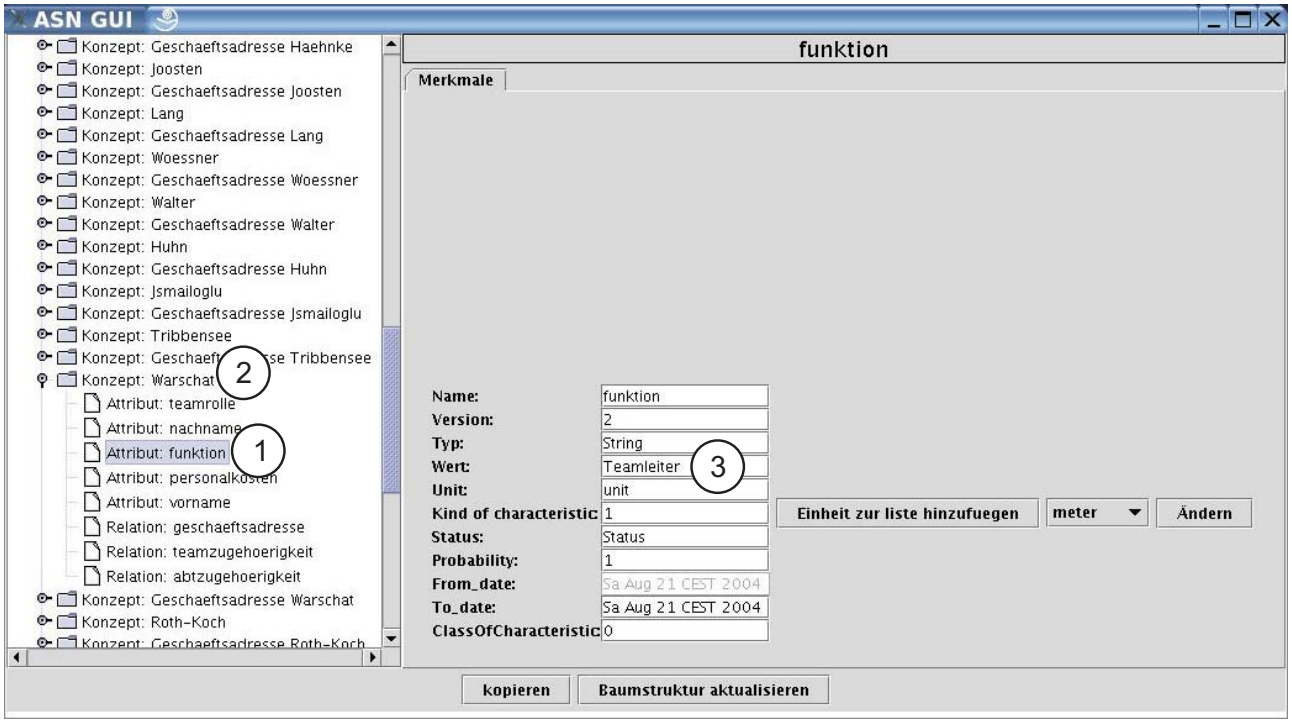


Bild 53: Screenshot: ASN-Browser (Retrieval-Agent) (1)

Zur Verdeutlichung der gewonnenen Informationen durch den Retrieval-Agenten sind in Bild 53 und Bild 54 die Inhalte mit Hilfe des ASN-Browsers dargestellt. Bild 53 zeigt das Attribut *funktion* (1) des Konzepts *Warschat (2)* mit dem Wert

Teamleiter (3). Während in Bild 54 die semantische Verknüpfung *teamzugehoerigkeit* (1) von dem Mitarbeiter *Warschat* (2) zum Team *Entwicklung* (3) abgebildet ist.

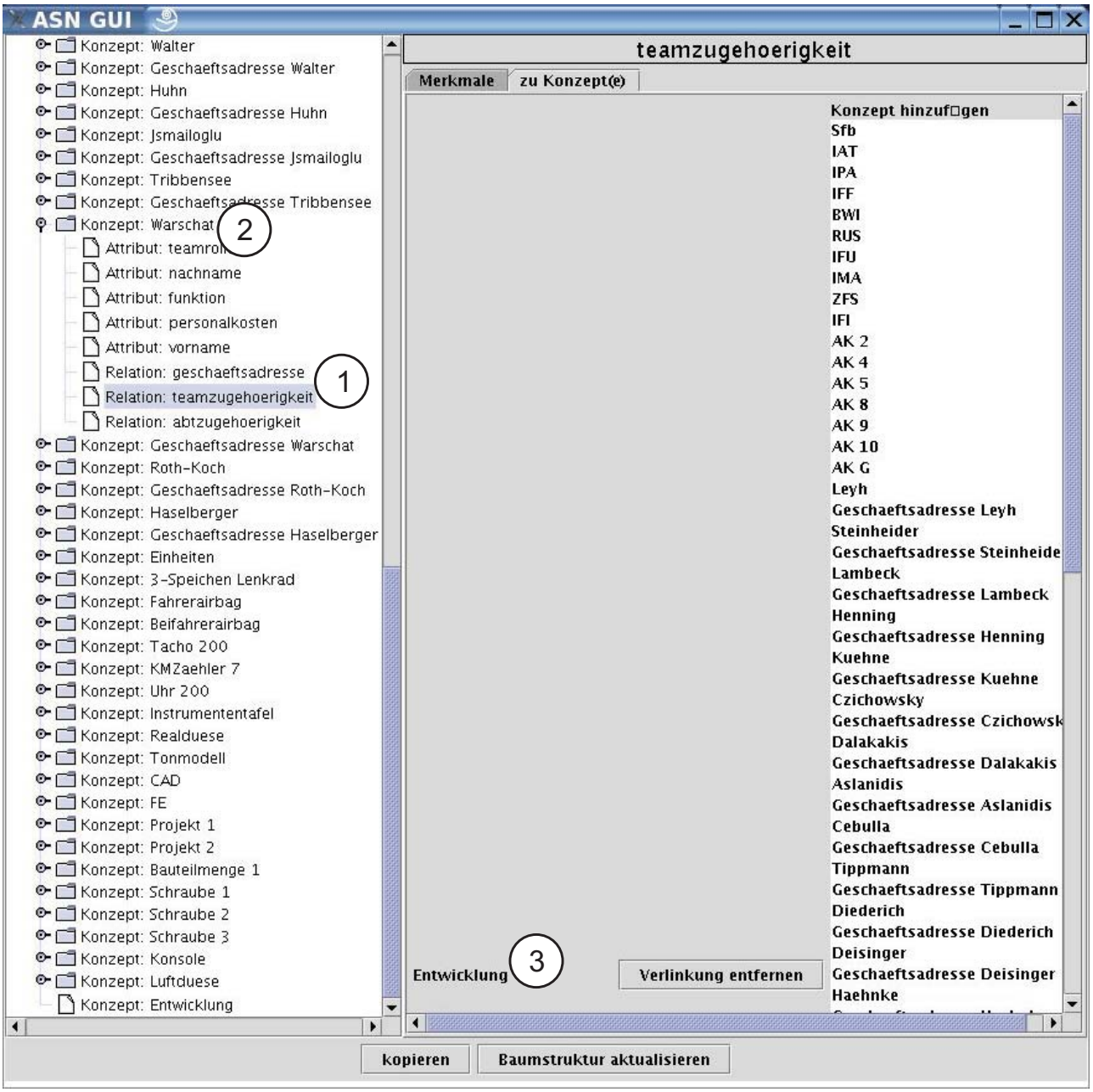


Bild 54: Screenshot: ASN-Browser (Retrieval-Agent) (2)

F.1.3 Retrievalanfrage und -ergebnis (Sequenzdiagramm)

Den Abschluss der genauen Beschreibung des Retrieval-Agenten bildet das Sequenzdiagramm (siehe Bild 55), das den Nachrichtenaustausch zwischen RPD-Anwendung und dem Retrieval-Agenten repräsentiert.

Im ersten Schritt (1) sendet der graphische Client des Retrieval-Agenten die *Retrieval-Request*-Nachricht an die RPD-Middleware. Diese wird dort durch den Agenten,

der im Moment die Masterfunktionalität besitzt, empfangen. Der Master-Agent instanziiert einen neuen Retrieval-Agenten und schickt diesem nach erfolgreicher Instanziierung den Request des Clients (2). Der Retrieval-Agent teilt dem Client mit der *Retrieval-Ack*-Nachricht mit, dass er empfangsbereit ist. Der Client beauftragt nun im Schritt (3) den Retrieval-Agenten. Dieser überprüft die Anfrage auf syntaktische Richtigkeit (4) und teilt das Ergebnis der Überprüfung dem Client mit (5). Anschließend berechnet er das Ergebnis und schickt dieses mit der *Retrieval-Result*-Nachricht an den Client (6). Da der Client keine weiteren Anfragen an den Retrieval-Agent stellen möchte, beendet er diesen mit der *Retrieval-Stop*-Nachricht (7).

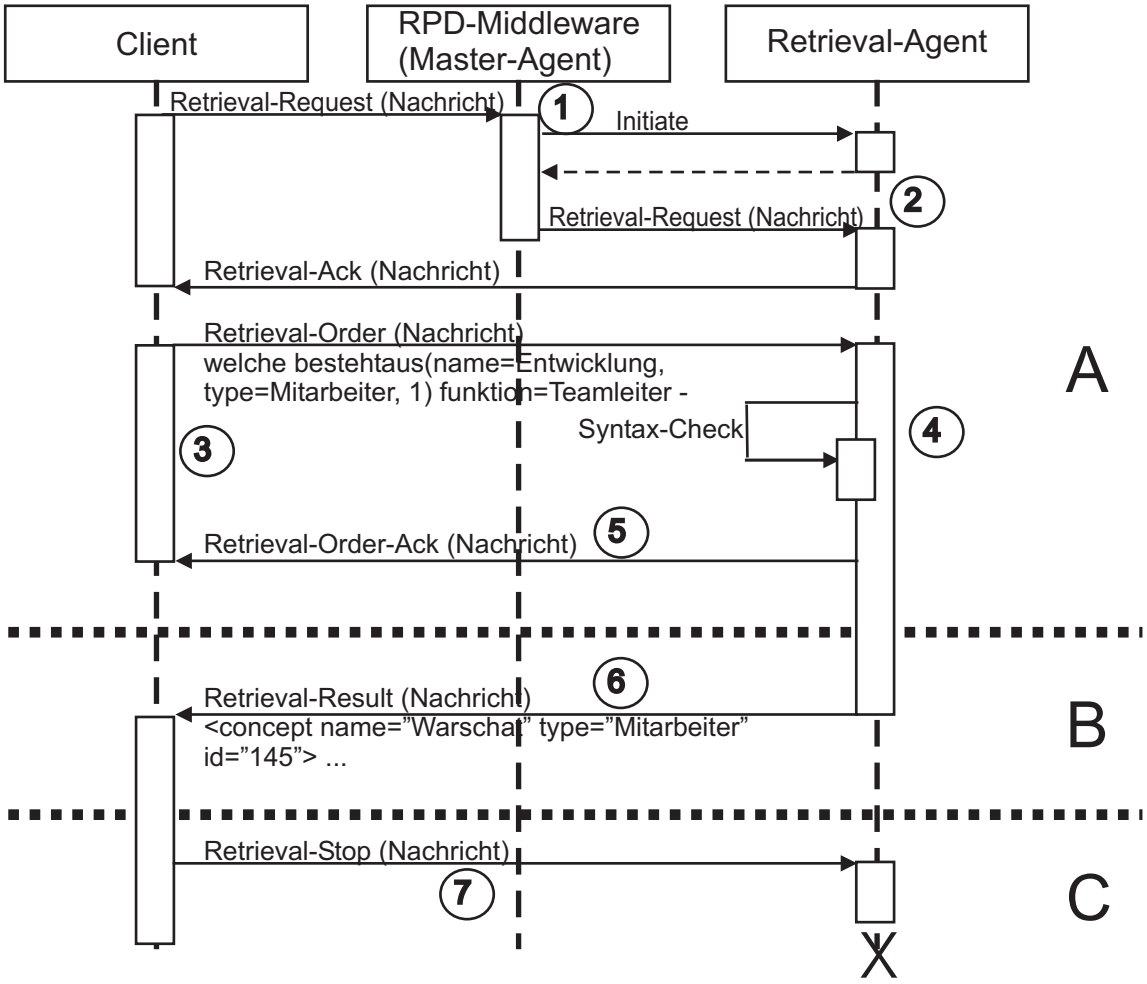


Bild 55: Sequenzdiagramm: Retrieval-Agent

F.2 Aggregations-Agent

F.2.1 Aggregationsanfrage und -ergebnis (XML-Notation)

Für das Anwendungsbeispiel aus Abschnitt 8.1 ist im Folgenden exemplarisch die Anfrage und das Ergebnis des Aggregations-Agenten in der XML-Notation dargestellt.

Anfrage an den Aggregations-Agenten

```
<aggregation-order>
   <sender>Projektplaner/LocalAgent/PP</sender>
   <destination>AggregationTeaminfo/AggregationAgent/49</destination>
   <aggregation name="Luftduese" type="" context="1">
      <element name="Teammitglieder" type="mitarbeiter">
         <query id="0" name="mitarbeiter" resultType="Mitarbeiter">
            <querySource queryType="asnql">
               welche bestehtaus(name = Entwicklung, type = Mitarbeiter, 1) -
            </querySource>
         </query>
      </element>

      <element name="Teamleiter" type="mitarbeiter">
         <query id="1" name="Teamleiter" resultType="Mitarbeiter">
            <querySource queryType="asnql">
               welche bestehtaus(name = Entwicklung, type = Mitarbeiter, 1)
               funktion = Teamleiter -
            </querySource>
         </query>
      </element>
   </aggregation>
</aggregation-order>
```

Ergebnis des Aggregations-Agenten

```
<aggregation-result>
   <sender> AggregationTeaminfo/AggregationAgent/49</sender>
   <destination>Projektplaner/LocalAgent/PP</destination>
   <aggregation name="Luftduese" type="" context="1">
      <element name="Teammitglieder" type="mitarbeiter">
         <result queryId="0">
            <concept name="Leyh" type="Mitarbeiter" id="105">
               <searchinfo rating="0" retrievedby="bestehtaus"
                  foundrelatedto="Entwicklung" direction="up" />
               <attributes>
                  <attribute name="nachname">Leyh</attribute>
                  <attribute name="vorname">Jens</attribute>
                  <attribute name="teamrolle"></attribute>
                  <attribute name="personalkosten">leer</attribute>
                  <attribute name="funktion">Mitarbeiter</attribute>
               </attributes>
               <relations>
                  <relation name="geschaeftsadresse" multiplicity="one"
```

```
        direction="Mitarbeiter" type="Aggregation">
        Geschaeftsadresse Leyh</relation>
        <relation name="teamzugehoerigkeit" multiplicity="one"
        direction="Team" type="Aggregation">Entwicklung</relation>
        <relation name="abtzugehoerigkeit" multiplicity="one"
        direction="ungerichtet" type="Assoziation">IAT</relation>
    </relations>
</concept>

<concept name="Lambeck" type="Mitarbeiter" id="109">
    <searchinfo rating="0" retrievedby="bestehtaus"
    foundrelatedto="Entwicklung" direction="up" />
    <attributes>
        <attribute name="nachname">Lambeck</attribute>
        <attribute name="vorname">Peter</attribute>
        <attribute name="teamrolle"></attribute>
        <attribute name="funktion">Mitarbeiter</attribute>
        <attribute name="personalkosten">leer</attribute>
    </attributes>
    <relations>
        <relation name="geschaeftsadresse" multiplicity="one"
        direction="Mitarbeiter" type="Aggregation">
        Geschaeftsadresse Lambeck</relation>
        <relation name="teamzugehoerigkeit" multiplicity="one"
        direction="Team" type="Aggregation">Entwicklung</relation>
        <relation name="abtzugehoerigkeit" multiplicity="one"
        direction="ungerichtet" type="Assoziation">IMA</relation>
    </relations>
</concept>

<concept name="Warschat" type="Mitarbeiter" id="145">
    <searchinfo rating="0" retrievedby="bestehtaus"
    foundrelatedto="Entwicklung" direction="up" />
    <attributes>
        <attribute name="nachname">Warschat</attribute>
        <attribute name="vorname">Joachim</attribute>
        <attribute name="teamrolle"></attribute>
        <attribute name="funktion">Teamleiter</attribute>
        <attribute name="personalkosten">leer</attribute>
    </attributes>
    <relations>
        <relation name="geschaeftsadresse" multiplicity="one"
        direction="Mitarbeiter" type="Aggregation">
        Geschaeftsadresse Warschat</relation>
        <relation name="teamzugehoerigkeit" multiplicity="one"
```

```
            direction="Team" type="Aggregation">Entwicklung</relation>
            <relation name="abtzugehoerigkeit" multiplicity="one"
            direction="ungerichtet" type="Assoziation">IAT</relation>
         </relations>
      </concept>

      <concept name="Haselberger" type="Mitarbeiter" id="149">
         <searchinfo rating="0" retrievedby="bestehtaus"
         foundrelatedto="Entwicklung" direction="up" />
         <attributes>
            <attribute name="nachname">Haselberger</attribute>
            <attribute name="vorname">Frank</attribute>
            <attribute name="teamrolle"></attribute>
            <attribute name="funktion">Mitarbeiter</attribute>
            <attribute name="personalkosten">leer</attribute>
         </attributes>
         <relations>
            <relation name="geschaeftsadresse" multiplicity="one"
            direction="Mitarbeiter" type="Aggregation">
            Geschaeftsadresse Haselberger</relation>
            <relation name="teamzugehoerigkeit" multiplicity="one"
            direction="Team" type="Aggregation">Entwicklung</relation>
            <relation name="abtzugehoerigkeit" multiplicity="one"
            direction="ungerichtet" type="Assoziation">IAT</relation>
         </relations>
      </concept>
   </result>
</element>

<element name="Teamleiter" type="mitarbeiter">
   <result queryId="1">
      <concept name="Warschat" type="Mitarbeiter" id="145">
         <searchinfo rating="0" retrievedby="bestehtaus"
         foundrelatedto="Entwicklung" direction="up" />
         <attributes>
            <attribute name="nachname">Warschat</attribute>
            <attribute name="vorname">Joachim</attribute>
            <attribute name="teamrolle"></attribute>
            <attribute name="funktion">Teamleiter</attribute>
            <attribute name="personalkosten">leer</attribute>
         </attributes>
         <relations>
            <relation name="geschaeftsadresse" multiplicity="one"
            direction="Mitarbeiter" type="Aggregation">
            Geschaeftsadresse Warschat</relation>
```

```
                <relation name="teamzugehoerigkeit" multiplicity="one"
                direction="Team" type="Aggregation">Entwicklung</relation>
                <relation name="abtzugehoerigkeit" multiplicity=”one"
                direction="ungerichtet" type="Assoziation">IAT</relation>
              </relations>
            </concept>
          </result>
        </element>
    </aggregation>
</aggregation-result>
```

F.2.2 Aggregationsanfrage und -ergebnis (graphischer Client)

Im Screenshot Bild 56 ist der Client des Aggregations-Agenten für die Anfrage aus dem Anwendungsbeispiel abgebildet. Das rechte Texteingabefeld (1) zeigt die Anfrage in XML-Notation, die aus den Teilanfragen (2) und (3) bestehen. Die Baumstruktur (4) auf der linken Seite repräsentiert die Anfrage in einer graphischen Darstellung. Im Texteingabefeld (5) wird die UAI des Aggregations-Agenten definiert und mit der Schaltfläche (6) wird der RPD-Middleware und damit dem Aggregations-Agenten die Anfrage zur Bearbeitung übergeben.

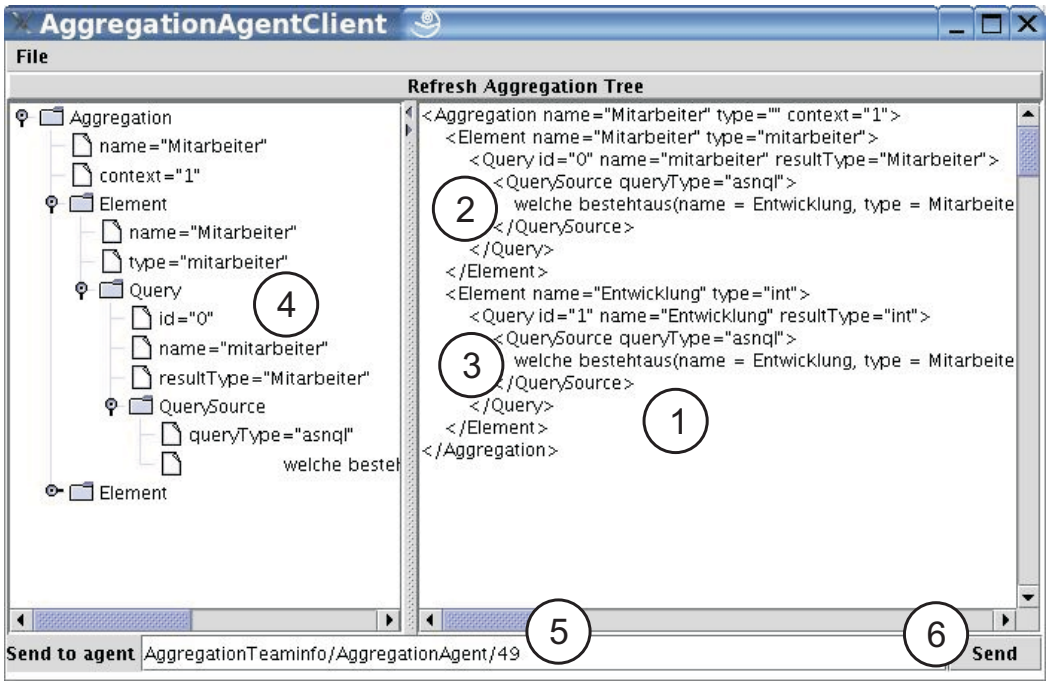


Bild 56: Screenshot: Anfrage an den Aggregations-Agenten

Nach der Beauftragung der RPD-Middleware durch den Client wird der Aggregations-Agent (siehe Bild 57) instanziiert. Im Textfeld (1) sind bereits die Ergebnisse der beauftragten Retrieval-Agenten (siehe Bild 58 und Bild 59) dargestellt. Auch hier weisen die Anzeigeelemente (2) auf den aktuellen Zustand des Agenten, den

er innerhalb des Master-Agenten-Prozesses inne hat (siehe Abschnitt 5.4), hin. In der Statusleiste (3) ist ersichtlich, dass der Agent im Moment Standby-Master ist.

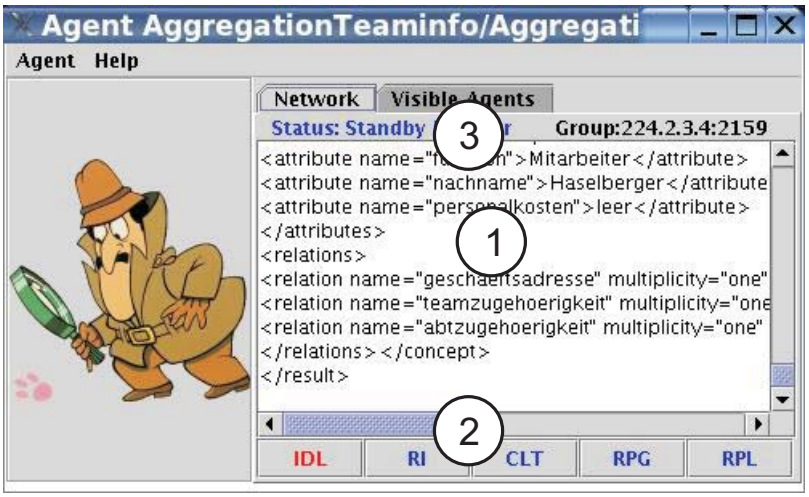


Bild 57: Screenshot: Aggregations-Agent

Um die beiden Teilanfragen zu einem Ergebnis zusammenführen zu können, beauftragt der Aggregations-Agent die beiden Retrieval-Agenten (siehe Bild 58 und Bild 59). Im Textfeld (1) ist jeweils die Anfrage zu finden, während die Anzeigeelemente (2) den Zustand des Agenten innerhalb des Master-Protokolls zeigen. Auch hier zeigt die Statusleiste (3) des Retrieval-Agenten an, dass beide Agenten als Standby-Master arbeiten.

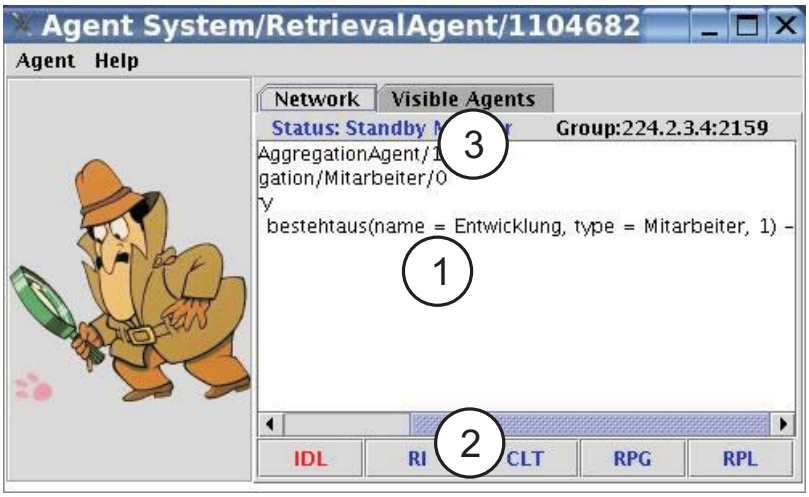


Bild 58: Screenshot: Retrieval-Agent 1 für Teilanfrage 1

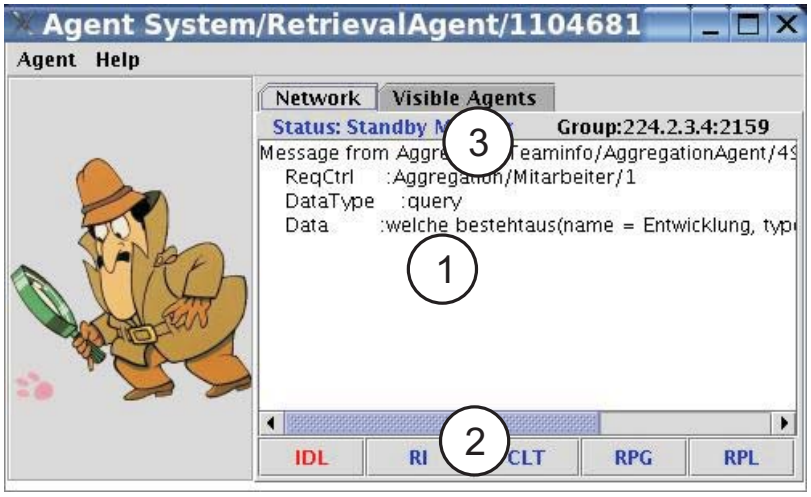


Bild 59: Screenshot: Retrieval-Agent 2 für Teilanfrage 2

In Bild 60 ist das Ergebnis des Aggregations-Agenten präsentiert. Im Textfeld (1) auf der rechten Seite ist das Ergebnis in der XML-Notation dargestellt, während die Baumstruktur (2) auf der linken Seite, das Ergebnis in graphischer Form zeigt.

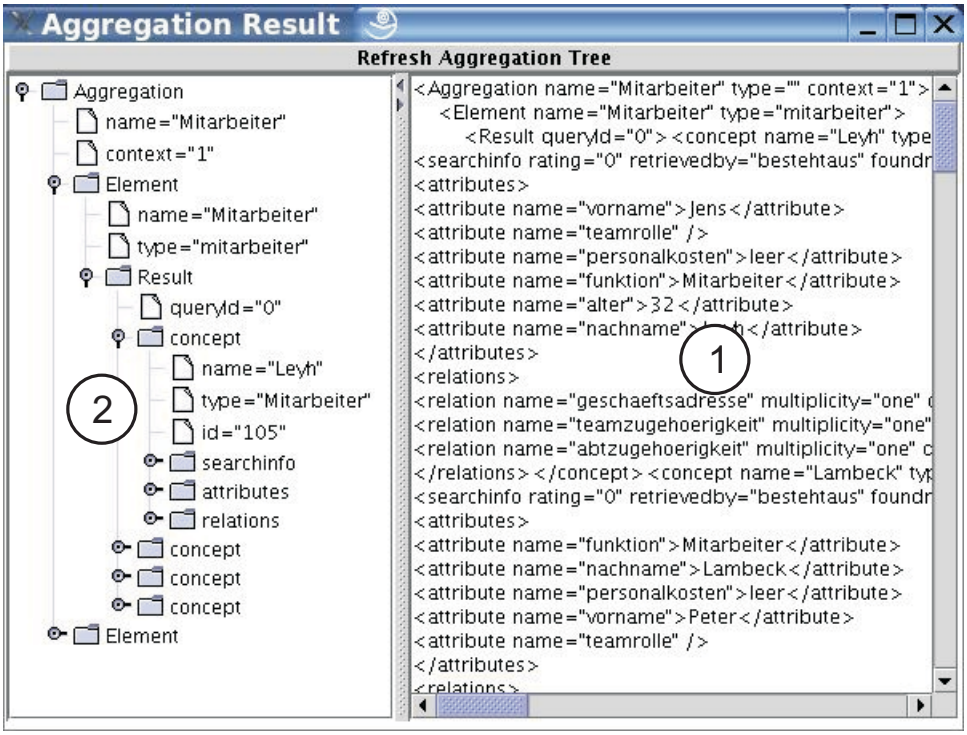


Bild 60: Screenshot: Ergebnis des Aggregations-Agenten

F.2.3 Aggregationsanfrage und -ergebnis (Sequenzdiagramm)

Den Abschluss der genauen Beschreibung des Aggregations-Agenten bildet das Sequenzdiagramm (siehe Bild 61), das den Nachrichtenaustausch zwischen der RPD-Anwendung und dem Aggregations-Agenten repräsentiert.

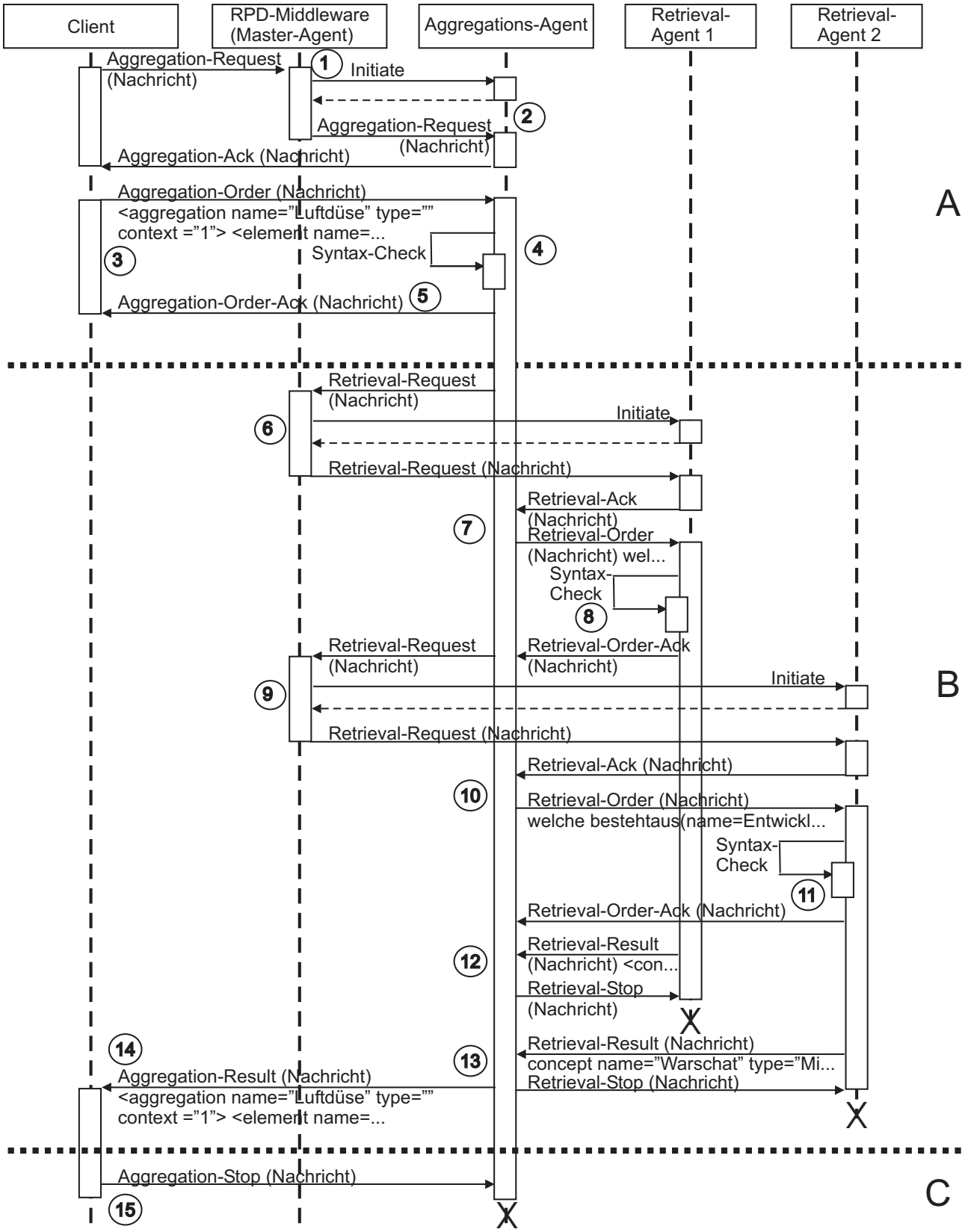


Bild 61: Sequenzdiagramm: Aggregations-Agent

Ähnlich dem Retrieval-Agenten aus Anhang F.1.3 fordert der Client des Aggregations-Agenten im ersten Schritt (1) einen neuen Aggregations-Agenten vom Master-Agenten an. Dieser instanziiert den Aggregations-Agenten und übergibt ihm die Anforderung des Clients (2). Der Aggregations-Agent bestätigt dem Client die Anforderung und teilt ihm so seine Empfangsbereitschaft mit. Der Client beauftragt nun den Aggregations-Agenten im Schritt (3). Dieser überprüft die Anfrage auf syntaktische Richtigkeit (4) und bestätigt sie dem Client (5). In der Arbeitsphase zerlegt der Aggregations-Agent die Aggregationsanfrage in zwei Retrieval-Anfragen. Für die beiden Anfragen beantragt er im Schritt (6) und (9) bei der RPD-Middleware jeweils einen Retrieval-Agenten. Sind diese empfangsbereit erhalten sie von Aggregations-Agenten im Schritt (7) und (10) ihre Aufgabe. Beide Retrieval-Agenten führen im Schritt (8) und (11) eine syntaktische Überprüfung der Anfrage durch und teilen das Ergebnis der Überprüfung dem Aggregations-Agenten mit. Im Schritt (12) und (13) werden die Ergebnisse der Retrieval-Agenten an den Aggregations-Agenten geliefert und beide Retrieval-Agenten beendet. Diese Ergebnisse fügt der Aggregations-Agent zu einem Ergebnis zusammen und übergibt es im Schritt (14) dem Client. Der Client beendet daraufhin den Aggregations-Agenten (15).

F.3 Monitor-Agent

F.3.1 Monitoranfrage (XML-Notation)

Für das Anwendungsbeispiel aus Abschnitt 8.2 ist im Folgenden für den Projektplaner die Anfrage an den Monitor-Agenten sowohl in der Anfragesprache als auch in der XML-Notation dargestellt.

Anfragesprache

String(Value(Luftdüse, Simulation_Prototyp_Luftdüse, Bearbeitungsstatus)) = String(fertig)

XML-Notation

```
<monitor-order>
   <sender>Projektplaner/LocalAgent/PP</sender>
   <destination>Michael/MonitorAgent/1104681895</destination>

   <expr>
      <operator>equal</operator>
      <parameter>
         <net>Luftdüse</net>
         <concept>Simulations_Prototyp_Luftdüse</concept>
         <attribute>Bearbeitungsstatus</attribute>
      </parameter>
      <parameter>
```

```
            <string>fertig</string>
        </parameter>
    </expr>
</monitor-order>
```

F.3.2 Monitoranfrage (graphischer Client)

In Bild 62 ist der Client des Monitor-Agenten dargestellt. Im Texteingabefeld (1) wird die Überwachungsbedingung eingegeben. Über die Schaltflächen (2) wird der Client gesteuert, wobei mit *Send* die Anfrage an die RPD-Middleware geschickt wird, mit *Stop* der Monitor-Agent und mit *Exit* zusätzlich der Client beendet wird. Die Anzeigenelemente (3) zeigen den syntaktischen und semantischen Zustand der Monitor-Anfrage und den Zustand des ASN. So ist im Moment hier die Überwachungsbedingung (*Monitor-Order*) nicht erfüllt, aber die Bedingung (*Monitor-Order-State*) syntaktisch und semantisch korrekt formuliert. Ebenfalls ist das ASN (*ASN-State*) verfügbar.

Bild 62: Screenshot: Client des Monitor-Agenten

Der Screenshot in Bild 63 zeigt zur Verdeutlichung den Wert (1) des Attributs, der durch die Überwachungsbedingung überwacht wird. Da der Wert *nicht fertig* ist, kann die Bedingungsanweisung nicht wahr sein. Damit die Bedingung wahr wird, muss der Wert auf *fertig* geändert werden (2).

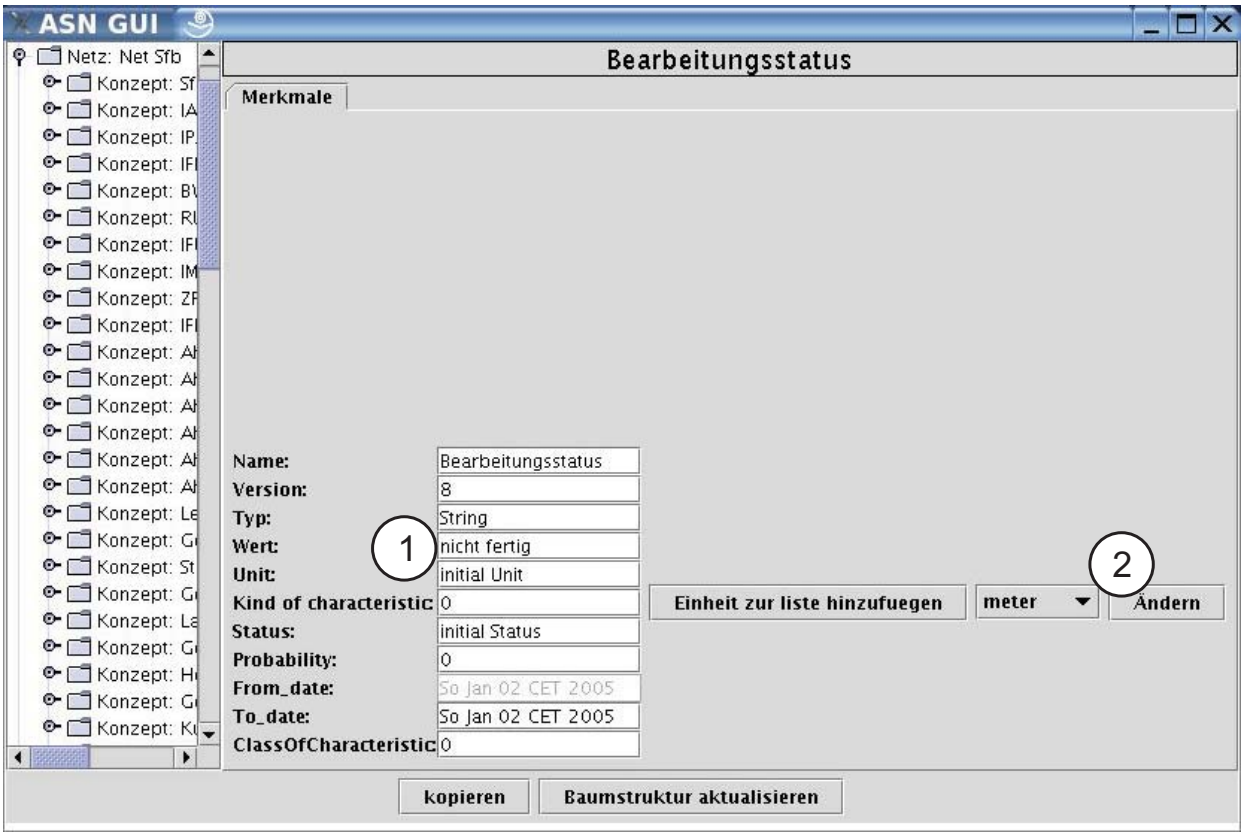


Bild 63: Screenshot: ASN-Browser (Monitor-Agent)

Nach der Beauftragung der RPD-Middleware durch den Client wird der Monitor-Agent (siehe Bild 64) instanziiert. Im Textfeld (1) ist eine Nachricht vom ASN über die letzte Änderung des Attributs *Bearbeitungsstatus* angezeigt. Auch hier weisen die Anzeigeelemente (2) auf den aktuellen Zustand des Agenten hin, den er innerhalb des Master-Agenten-Prozesses inne hat (siehe Abschnitt 5.4). In der Statusleiste (3) ist ersichtlich, dass der Agent im Moment Standby-Master ist.

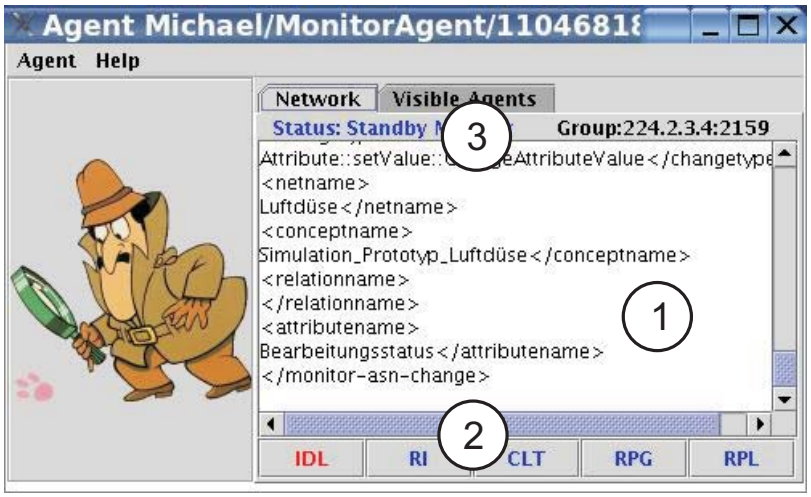


Bild 64: Screenshot: Monitor-Agent

F.3.3 Monitoranfrage (Sequenzdiagramm)

Den Abschluss der genauen Beschreibung des Monitor-Agenten bildet das Sequenzdiagramm (siehe Bild 65), das den Nachrichtenaustausch zwischen der RPD-Anwendung und dem Monitor-Agenten repräsentiert.

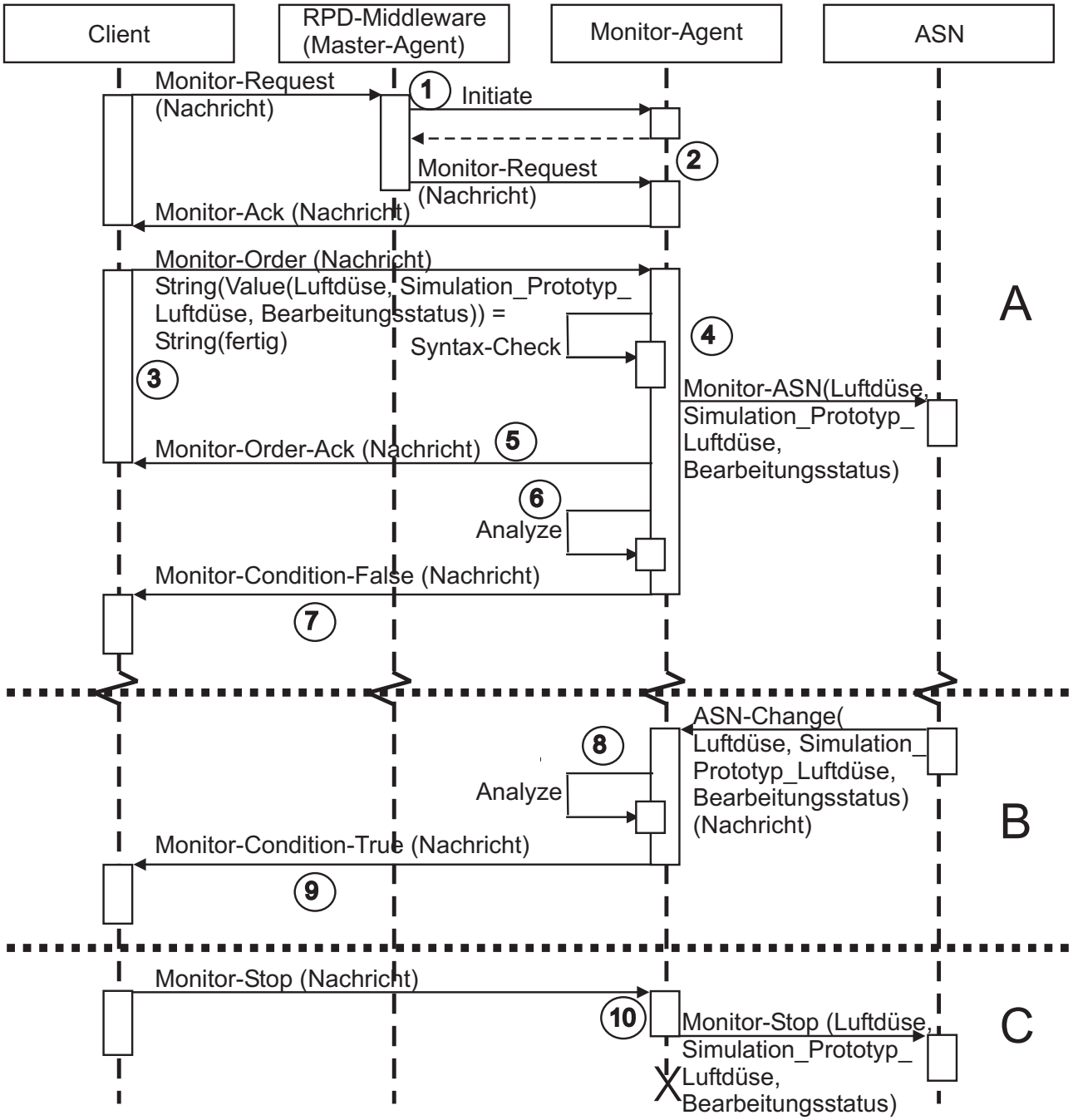


Bild 65: Sequenzdiagramm: Monitor-Agent

Der Client des Monitor-Agenten beantragt bei der RPD-Middleware im Schritt (1) einen neuen Monitor-Agenten. Der Master-Agent instanziiert diesen und leitet ihm den Request des Clients weiter (2). Der neue Monitor-Agent teilt daraufhin dem

Client seine Empfangsbereitschaft durch die Nachricht *Monitor-Ack* mit. Im Schritt (3) übergibt der Client dem Monitor-Agent die Überwachungsbedingung. Diese wird durch den Monitor-Agenten syntaktisch und semantisch überprüft (4). Anschließend werden die benötigten Überwachungspunkte im ASN gesetzt, bevor die Anfrage dem Client durch die Nachricht *Monitor-Order-Ack* im Schritt (5) bestätigt wird. Daraufhin wertet der Monitor-Agent erstmalig die Überwachungsbedingung aus (6) und teilt dem Client mit, dass diese falsch ist (7). In der Bearbeitungsphase tritt eine Veränderung am überwachten Attribut im ASN auf. Das ASN meldet dies mit der Nachricht *ASN-Change* dem Monitor-Agenten. Dieser berechnet im Schritt (8) seine Überwachungsbedingung neu und teilt dem Client im Schritt (9) mit, dass sie von *falsch* nach *wahr* gewechselt hat. Der Client beschließt daraufhin im Schritt (10), dass er die Dienste des Monitor-Agenten nicht mehr benötigt und beendet diesen mit der Nachricht *Monitor-Stop*. Der Monitor-Agent entfernt seine Überwachungspunkte im ASN und beendet sich selbst.

F.4 Koordinations-Agent

F.4.1 Koordinationsprotokoll (XML-Notation)

Für das Anwendungsbeispiel aus Abschnitt 8.6 ist im Folgenden das Koordinationsprotokoll in der Anfragesprache und in der XML-Notation dargestellt.

Anfragesprache

(Start) Start (Projektplan überprüfen) Normal (Projektplan verletzt) Normal (Fertigung prüfen) Normal (Projektplan prüfen) Normal (Kosten neu berechnen) Normal (Kostenplan verletzt) Normal (Fertigungsverfahren geändert) Normal (Materialkosten geändert) Normal (Kostenplan Fertigungsverfahren eingehalten) Normal (Qualität prüfen) Normal (Qualität eingehalten) Normal (Qualitätsstandard verletzt) Normal (Qualität Bauteile geändert) Normal (Qualität Material geändert) Normal (Qualität Bauteile eingehalten) Normal (Qualität Material eingehalten) Normal (Projektplan eingehalten) Ende (Fertigung terminiert) Ende (Projektplan anpassen) Ende

(Start) (Projektplan verletzt) (Zeitüberschreitung erkannt) PP (Zeitüberschreitung erkannt) PB
(Start) (Kostenplan verletzt) (Kosten überschritten) KR (Kosten senken) PB
(Start) (Qualitätsstandard verletzt) (Qualität unterschritten) QM (Qualität unterschritten) PB

(Projektplan überprüfen) (Projektplan eingehalten) (Projektplan eingehalten) PP (Projektplan eingehalten) KR, QM, PB
(Projektplan überprüfen) (Projektplan verletzt) (Projektplan verletzt) PP (Projektplan verletzt) KR, QM, PB
(Projektplan verletzt) (Fertigung prüfen) (neuer Fertigstellungszeitpunkt vorschlagen) PP (neuer Fertigstellungszeitpunkt) PB
(Projektplan verletzt) (Projektplan prüfen) (neuer Fertigstellungszeitpunkt vorschla-

gen) PB (neuer Fertigstellungszeitpunkt) PP
(Fertigung prüfen) (Projektplan verletzt) (Fertigstellungszeitpunkt ablehnen) PB
(Fertigstellungszeitpunkt abgelehnt) PP
(Fertigung prüfen) (Fertigung terminiert) (Fertigstellungszeitpunkt annehmen) PB
(Fertigstellungszeitpunkt angenommen) PP
(Projektplan prüfen) (Projektplan verletzt) (Fertigstellungszeitpunkt ablehnen) PP
(Fertigstellungszeitpunkt abgelehnt) PB
(Projektplan prüfen) (Projektplan anpassen) (Fertigstellungszeitpunkt annehmen)
PP (Fertigstellungszeitpunkt angenommen) PB

(Kosten neu berechnen) (Projektplan überprüfen) (Kostenplan eingehalten) KR
(Projektplan überprüfen) PB, PP
(Kosten neu berechnen) (Kostenplan verletzt) (Kostenplan verletzt) KR (Kostenplan verletzt) PB, QM
(Kostenplan verletzt) (Fertigungsverfahren geändert) (Fertigungsverfahren geändert) PB (Kosten Fertigungsverfahren neu berechnen) KR
(Kostenplan verletzt) (Materialkosten geändert) (Material geändert) PB (Kosten Material neu berechnen) KR
(Fertigungsverfahren geändert) (Kostenplan verletzt) (Kostenplan verletzt) KR
(Kostenplan Fertigungsverfahren verletzt) PB
(Fertigungsverfahren geändert) (Kostenplan Fertigungsverfahren eingehalten)
(Kostenplan eingehalten) KR (Kostenplan eingehalten) PB
(Materialkosten geändert) (Kostenplan verletzt) (Kostenplan verletzt) KR (Kostenplan Material verletzt) PB
(Materialkosten geändert) (Qualität prüfen) (Qualität überprüfen) KR (Qualität überprüfen) PB, QM
(Kostenplan Fertigungsverfahren eingehalten) (Projektplan überprüfen) (Projektplan überprüfen) KR (Projektplan überprüfen) PB, PP

(Qualität prüfen) (Qualität eingehalten) (Qualität eingehalten) QM (Qualität eingehalten) KR
(Qualität prüfen) (Qualitätsstandard verletzt) (Qualität unterschritten) QM (Qualität unterschritten) PB, KR
(Qualität eingehalten) (Projektplan prüfen) (Projektplan überprüfen) QM (Projektplan überprüfen) PB, PP
(Qualitätsstandard verletzt) (Qualität Bauteile geändert) (Bauteile ausgetauscht) PB
(Qualität neu berechnen) QM
(Qualitätsstandard verletzt) (Qualität Material geändert) (Material geändert) PB
(Qualität neu berechnen) QM
(Qualität Bauteile geändert) (Qualitätsstandard verletzt) (Qualität Bauteile nicht eingehalten) QM (Qualität Bauteile nicht eingehalten) PB
(Qualität Bauteile geändert) (Qualität Bauteile eingehalten) (Qualität eingehalten) QM (Qualität eingehalten) PB
(Qualität Material geändert) (Qualitätsstandard verletzt) (Qualität Material nicht eingehalten) QM (Qualität Material nicht eingehalten) PB

(Qualität Material geändert) (Qualität Material eingehalten) (Qualität eingehalten) QM (Qualität eingehalten) PB
(Qualität Bauteile eingehalten) (Kosten neu berechnen) (Kosten überprüfen) QM (Kosten neu berechnen) PB, KR
(Qualität Material eingehalten) (Kosten neu berechnen) (Kosten überprüfen) QM (Kosten neu berechnen) PB, KR

XML-Notation

```
<fsm-definition>
   <sender>Prototypenbauer/LocalAgent/PB</sender>
   <destination>Michael/CoordinationAgent/11046783215</destination>

   <std>
      <startstate>Start</startstate>
      <normalstate>Projektplan überprüfen</normalstate>
      <normalstate>Projektplan verletzt</normalstate>
      <normalstate>Fertigung prüfen</normalstate>
      <normalstate>Projektplan prüfen</normalstate>
      <normalstate>Kosten neu berechnen</normalstate>
      <normalstate>Kostenplan verletzt</normalstate>
      <normalstate>Fertigungsverfahren geändert</normalstate>
      <normalstate>Materialkosten geändert</normalstate>
      <normalstate>Kostenplan Fertigungsverfahren eingehalten</normalstate>
      <normalstate>Qualität prüfen</normalstate>
      <normalstate>Qualität eingehalten</normalstate>
      <normalstate>Qualitätsstandard verletzt</normalstate>
      <normalstate>Qualität Bauteile geändert</normalstate>
      <normalstate>Qualität Material geändert</normalstate>
      <normalstate>Qualität Bauteile eingehalten</normalstate>
      <normalstate>Qualität Material eingehalten</normalstate>
      <endstate>Projektplan eingehalten</endstate>
      <endstate>Fertigung terminiert</endstate>
      <endstate>Projektplan anpassen</endstate>

      <transition>
         <state1>Start</state1>
         <state2>Projektplan verletzt</state2>
         <input>
            <info>Zeitüberschreitung erkannt</info>
            <sender>PP</sender>
         </input>
         <output>
            <info>Zeitüberschreitung erkannt</info>
            <recipient>PB</recipient>
```

```
      </output>
    </transition>
    <transition>
      <state1>Start</state1>
      <state2>Kostenplan verletzt</state2>
      <input>
        <info>Kosten überschritten</info>
        <sender>KR</sender>
      </input>
      <output>
        <info>Kosten senken</info>
        <recipient>PB</recipient>
      </output>
    </transition>
    <transition>
      <state1>Start</state1>
      <state2>Qualitätsstandard verletzt</state2>
      <input>
        <info>Qualität unterschritten</info>
        <sender>QM</sender>
      </input>
      <output>
        <info>Qualität unterschritten</info>
        <recipient>PB</recipient>
      </output>
    </transition>

    <transition>
      <state1>Projektplan überprüfen</state1>
      <state2>Projektplan eingehalten</state2>
      <input>
        <info>Projektplan eingehalten</info>
        <sender>PP</sender>
      </input>
      <output>
        <info>Projektplan eingehalten</info>
        <recipient>KR, QM, PB</recipient>
.     </output>
    </transition>
    <transition>
      <state1>Projektplan überprüfen</state1>
      <state2>Projektplan verletzt</state2>
      <input>
        <info>Projektplan verletzt</info>
```

```
        <sender>PP</sender>
    </input>
    <output>
        <info>Projektplan verletzt</info>
        <recipient>KR, QM, PB</recipient>
    </output>
</transition>
<transition>
    <state1>Projektplan verletzt</state1>
    <state2>Fertigung prüfen</state2>
    <input>
        <info>neuer Fertigstellungszeitpunkt vorschlagen</info>
        <sender>PP</sender>
    </input>
    <output>
        <info>neuer Fertigstellungszeitpunkt</info>
        <recipient>PB</recipient>
    </output>
</transition>
<transition>
    <state1>Projektplan verletzt</state1>
    <state2>Projektplan prüfen</state2>
    <input>
        <info>neuer Fertigstellungszeitpunkt vorschlagen</info>
        <sender>PB</sender>
    </input>
    <output>
        <info>neuer Fertigstellungszeitpunkt</info>
        <recipient>PP</recipient>
    </output>
</transition>
<transition>
    <state1>Fertigung prüfen</state1>
    <state2>Projektplan verletzt</state2>
    <input>
        <info>Fertigstellungszeitpunkt ablehnen</info>
        <sender>PB</sender>
    </input>
    <output>
        <info>Fertigstellungszeitpunkt abgelehnt</info>
        <recipient>PP</recipient>
    </output>
</transition>
<transition>
```

```
      <state1>Fertigung prüfen</state1>
      <state2>Fertigung terminiert</state2>
      <input>
        <info>Fertigstellungszeitpunkt annehmen</info>
        <sender>PB</sender>
      </input>
      <output>
        <info>Fertigstellungszeitpunkt angenommen</info>
        <recipient>PP</recipient>
      </output>
    </transition>
    <transition>
      <state1>Projektplan prüfen</state1>
      <state2>Projektplan verletzt</state2>
      <input>
        <info>Fertigstellungszeitpunkt ablehnen</info>
        <sender>PP</sender>
      </input>
      <output>
        <info>Fertigstellungszeitpunkt abgelehnt</info>
        <recipient>PB</recipient>
      </output>
    </transition>
    <transition>
      <state1>Projektplan prüfen</state2>
      <state2>Projektplan anpassen</state2>
      <input>
        <info>Fertigstellungszeitpunkt annehmen</info>
        <sender>PP</sender>
      </input>
      <output>
        <info>Fertigstellungszeitpunkt angenommen</info>
        <recipient>PB</recipient>
      </output>
    </transition>

.   <transition>
      <state1>Kosten neu berechnen</state1>
      <state2>Projektplan überprüfen</state2>
      <input>
        <info>Kostenplan eingehalten</info>
        <sender>KR</sender>
      </input>
      <output>
```

```
        <info>Projektplan überprüfen</info>
        <recipient>PB, PP</recipient>
      </output>
    </transition>
    <transition>
      <state1>Kosten neu berechnen</state1>
      <state2>Kostenplan verletzt</state2>
      <input>
        <info>Kostenplan verletzt</info>
        <sender>KR</sender>
      </input>
      <output>
        <info>Kostenplan verletzt</info>
        <recipient>PB, QM</recipient>
      </output>
    </transition>
    <transition>
      <state1>Kostenplan verletzt</state1>
      <state2>Fertigungsverfahren geändert</state2>
      <input>
        <info>Fertigungsverfahren geändert</info>
        <sender>PB</sender>
      </input>
      <output>
        <info>Kosten Fertigungsverfahren neu berechnen</info>
        <recipient>KR</recipient>
      </output>
    </transition>
    <transition>
      <state1>Kostenplan verletzt</state1>
      <state2>Materialkosten geändert</state2>
      <input>
        <info>Material geändert</info>
        <sender>PB</sender>
      </input>
      <output>
        <info>Kosten Material neu berechnen</info>
        <recipient>KR</recipient>
      </output>
    </transition>
    <transition>
      <state1>Fertigungsverfahren geändert</state1>
      <state2>Kostenplan verletzt</state2>
      <input>
```

```
        <info>Kostenplan verletzt</info>
        <sender>KR</sender>
      </input>
      <output>
        <info>Kostenplan Fertigungsverfahren verletzt</info>
        <recipient>PB</recipient>
      </output>
    </transition>
    <transition>
      <state1>Fertigungsverfahren geändert</state1>
      <state2>Kostenplan Fertigungsverfahren eingehalten</state2>
      <input>
        <info>Kostenplan eingehalten</info>
        <sender>KR</sender>
      </input>
      <output>
        <info>Kostenplan eingehalten</info>
        <recipient>PB</recipient>
      </output>
    </transition>
    <transition>
      <state1>Materialkosten geändert</state1>
      <state2>Kostenplan verletzt</state2>
      <input>
        <info>Kostenplan verletzt</info>
        <sender>KR</sender>
      </input>
      <output>
        <info>Kostenplan Material verletzt</info>
        <recipient>PB</recipient>
      </output>
    </transition>
    <transition>
      <state1>Materialkosten geändert</state1>
      <state2>Qualität prüfen</state2>
      <input>
        <info>Qualität überprüfen</info>
        <sender>KR</sender>
      </input>
      <output>
        <info>Qualität überprüfen</info>
        <recipient>PB, QM</recipient>
      </output>
    </transition>
```

```
<transition>
   <state1>Kostenplan Fertigungsverfahren eingehalten</state1>
   <state2>Projektplan überprüfen</state2>
   <input>
      <info>Projektplan überprüfen</info>
      <sender>KR</sender>
   </input>
   <output>
      <info>Projektplan überprüfen</info>
      <recipient>PB, PP</recipient>
   </ouput>
</transititon>

<transition>
   <state1>Qualität prüfen</state1>
   <state2>Qualität eingehalten</state2>
   <input>
      <info>Qualität eingehalten</info>
      <sender>QM</sender>
   </input>
   <output>
      <info>Qualität eingehalten</info>
      <recipient>KR</recipient>
   </output>
</transition>
<transititon>
   <state1>Qualität prüfen</state1>
   <state2>Qualitätsstandard verletzt</state2>
   <input>
      <info>Qualität unterschritten</info>
      <sender>QM</sender>
   </input>
   <output>
      <info>Qualität unterschritten</info>
      <recipient>PB, KR</recipient>
   </output>
</transition>
<transition>
   <state1>Qualität eingehalten</state1>
   <state2>Projektplan prüfen</state2>
   <input>
      <info>Projektplan überprüfen</info>
      <sender>QM</sender>
   </input>
```

```
    <output>
      <info>Projektplan überprüfen</info>
      <recipient>PB, PP</recipient>
    </output>
  </transition>
  <transititon>
    <state1>Qualitätsstandard verletzt</state1>
    <state2>Qualität Bauteile geändert</state2>
    <input>
      <info>Bauteile ausgetauscht</info>
      <sender>PB</sender>
    </input>
    <output>
      <info>Qualität neu berechnen</info>
      <recipient>QM</recipient>
    </output>
  </transition>
  <transition>
    <state1>Qualitätsstandard verletzt</state1>
    <state2>Qualität Material geändert</state2>
    <input>
      <info>Material geändert</info>
      <sender>PB</sender>
    </input>
    <output>
      <info>Qualität neu berechnen</info>
      <recipient>QM</recipient>
    </output>
  </transition>
  <transition>
    <stage1>Qualität Bauteile geändert</stage1>
    <stage2>Qualitätsstandard verletzt</stage2>
    <input>
      <info>Qualität Bauteile nicht eingehalten</info>
      <sender>QM</sender>
    </input>
    <output>
      <info>Qualität Bauteile nicht eingehalten</info>
      <recipient>PB</recipient>
    </output>
  </transition>
  <transition>
    <state1>Qualität Bauteile geändert</state1>
    <state2>Qualität Bauteile eingehalten</state2>
```

```
    <input>
      <info>Qualität eingehalten</info>
      <sender>QM</sender>
    </input>
    <output>
      <info>Qualität eingehalten</info>
      <recipient>PB</recipient>
    </output>
  </transition>
  <transition>
    <state1>Qualität Material geändert</state1>
    <state2>Qualitätsstandard verletzt</state2>
    <input>
      <info>Qualität Material nicht eingehalten</info>
      <sender>QM</sender>
    </input>
    <output>
      <info>Qualität Material nicht eingehalten</info>
      <recipient>PB</recipient>
    </output>
  </transition>
  <transition>
    <state1>Qualität Material geändert</state1>
    <state2>Qualität Material eingehalten</state2>
    <input>
      <info>Qualität eingehalten</info>
      <sender>QM</sender>
    </input>
    <output>
      <info>Qualität eingehalten</info>
      <recipient>PB</recipient>
    </output>
  </transition>
  <transition>
    <state1>Qualität Bauteile eingehalten</state1>
    <state2>Kosten neu berechnen</state2>
    <input>
      <info>Kosten überprüfen</info>
      <sender>QM</sender>
    </input>
    <output>
      <info>Kosten neu berechnen</info>
      <recipient>PB, KR</recipient>
    </output>
```

```
      </transition>
      <transition>
         <state1>Qualität Material eingehalten</state1>
         <state2>Kosten neu berechnen</state2>
         <input>
            <info>Kosten überprüfen</info>
            <sender>QM</sender>
         </input>
         <output>
            <info>Kosten neu berechnen</info>
            <recipient>PB, KR</recipient>
         </output>
      </transition>
   </std>

</fsm-definition>
```

F.4.2 Koordinationsprotokoll (graphischer Client)

Im ersten Schritt muss das Koordinationsprotokoll definiert werden. Hierzu werden die einzelnen RPD-Anwendungen definiert (siehe Bild 66). Dies geschieht mit Hilfe des Texteingabefeldes (1) und der Schaltfläche *Add* (2). Die angelegten Clients erscheinen dann im Textfeld (3) auf der linken Seite.

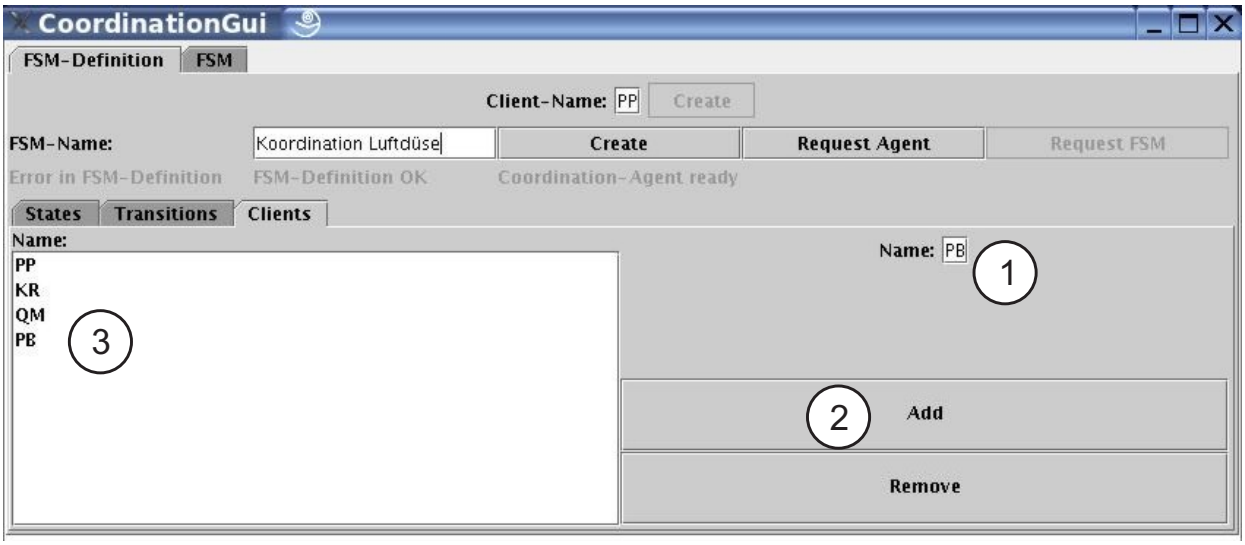


Bild 66: Screenshot: Eingabe der Clientdaten für das Koordinationsprotokoll *Koordination Luftdüse*

Im zweiten Schritt sind nun die Zustände des Koordinationsprotokolls anzulegen (siehe Bild 67). Hierzu wird für jeden Zustand zum einen der Name im Texteingabefeld (1) und zum anderen der Typ im Auswahlfeld (2) spezifiziert, bevor dieser mit der Schaltfläche *Add* (3) der Zustandsliste (4) hinzugefügt wird.

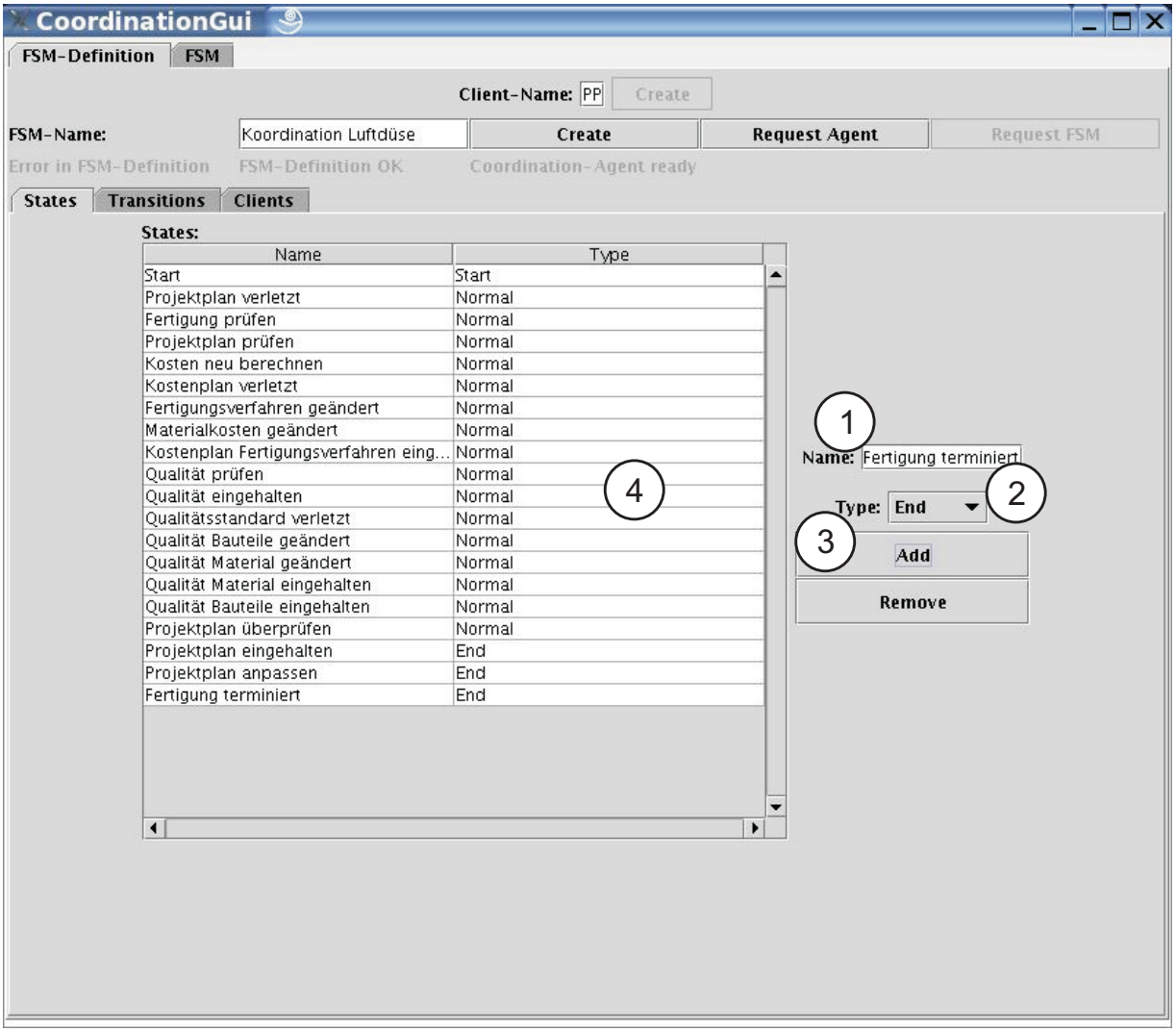


Bild 67: Screenshot: Eingabe der Zustände für das Koordinationsprotokoll *Koordination Luftdüse*

Zuletzt werden die Zustandsübergänge des Koordinationsprotokolls definiert (siehe Bild 68). Hierzu werden sowohl der Ausgangszustand, die Eingabenachricht und die Senderliste spezifiziert (1), als auch der Zielzustand, die Ausgabenachricht und die Empfängerliste (2). Anschließend wird der so definierte Zustandsübergang mit der Schaltfläche *Add* in die Zustandsübergangsliste übernommen (3).

Nach Abschluss dieser Tätigkeit wird nun der Koordinations-Agent gestartet. Dies geschieht dadurch, dass zum einen der erste Client (4) angelegt wird, in diesem Fall der Projektplaner (PP), und zum anderen der Name des Koordinationsprotokolls (Koordination Luftdüse) (5) festgelegt wird, bevor über die Schaltfläche *Create* (6) die RPD-Middleware mit der Instanziierung des Koordinations-Agenten beauftragt wird. Ist dieser Vorgang erfolgreich abgeschlossen, so wird dies durch das Anzeigeelement *FSM-Definition OK* (7) angezeigt. Sind hingegen tote Wege im

Koordinationsprotokoll so wird das Protokoll abgelehnt und dies durch das Anzeigeelement *Error in FSM-Definition* (8) deutlich gemacht.

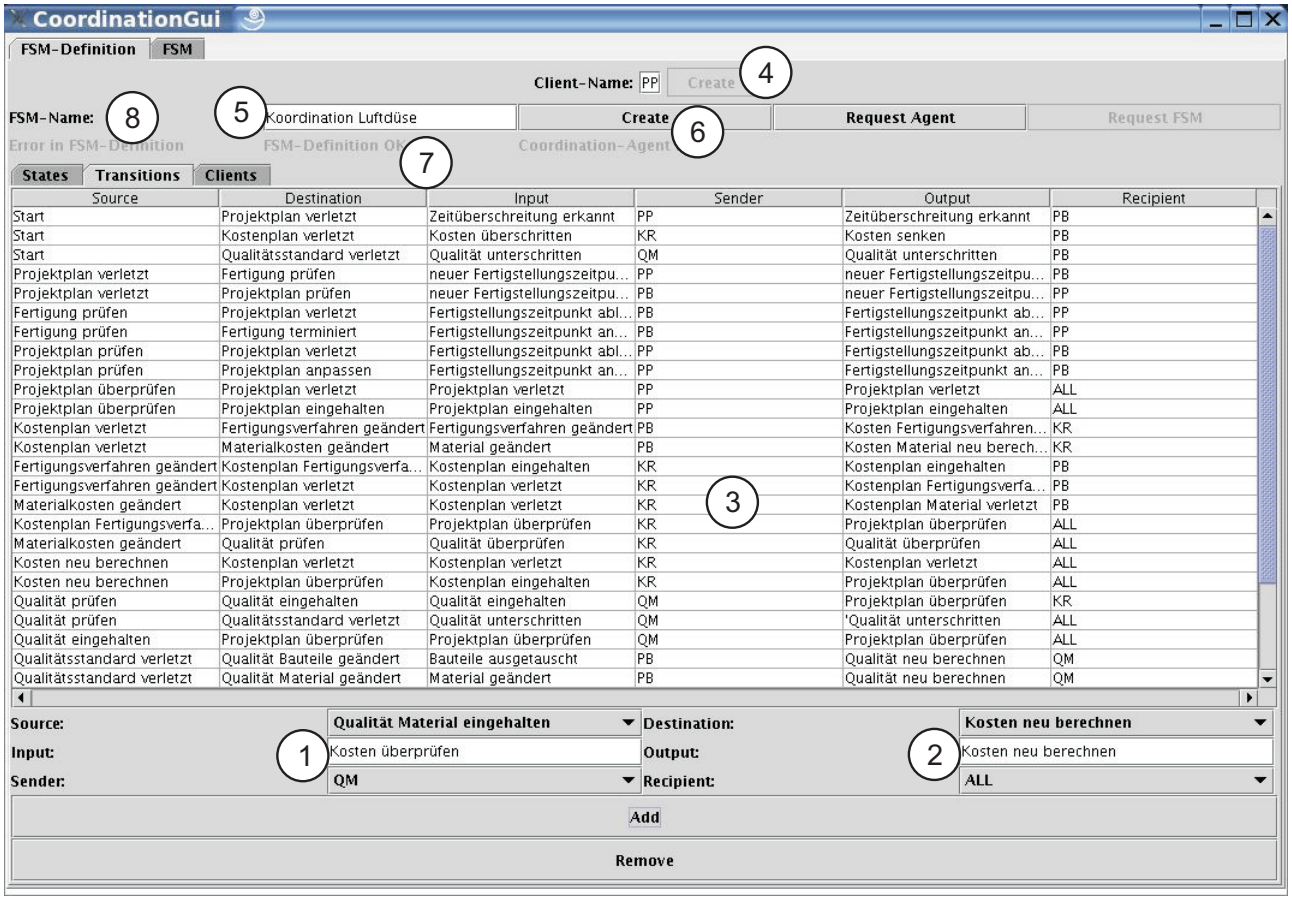


Bild 68: Screenshot: Eingabe der Zustandsübergänge für das Koordinationsprotokoll *Koordination Luftdüse*

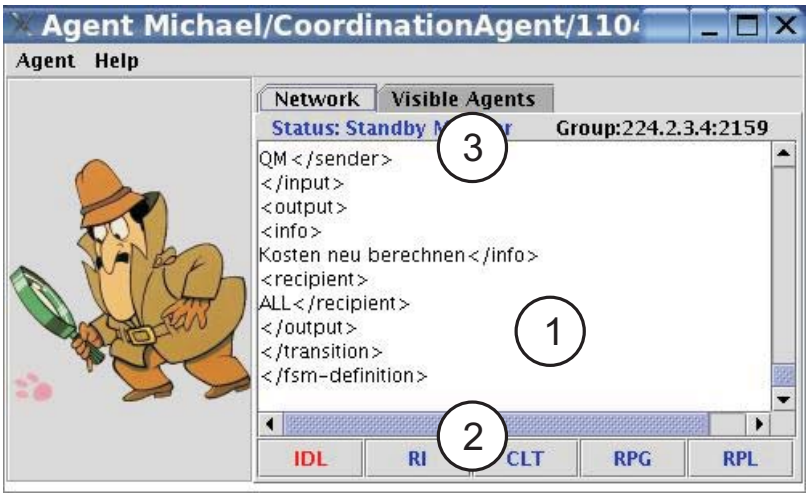


Bild 69: Screenshot: Koordinations-Agent

In Bild 69 ist nun der instanziierte Koordinations-Agent abgebildet. Im Textfeld (1) ist das Ende der Definition des Koordinationsprotokolls dargestellt. Auch hier weisen die Anzeigeelemente (2) auf den aktuellen Zustand des Agenten hin, den er innerhalb des Master-Agenten-Prozesses inne hat (siehe Abschnitt 5.4). In der Statusleiste (3) ist ersichtlich, dass der Agent im Moment Standby-Master ist.

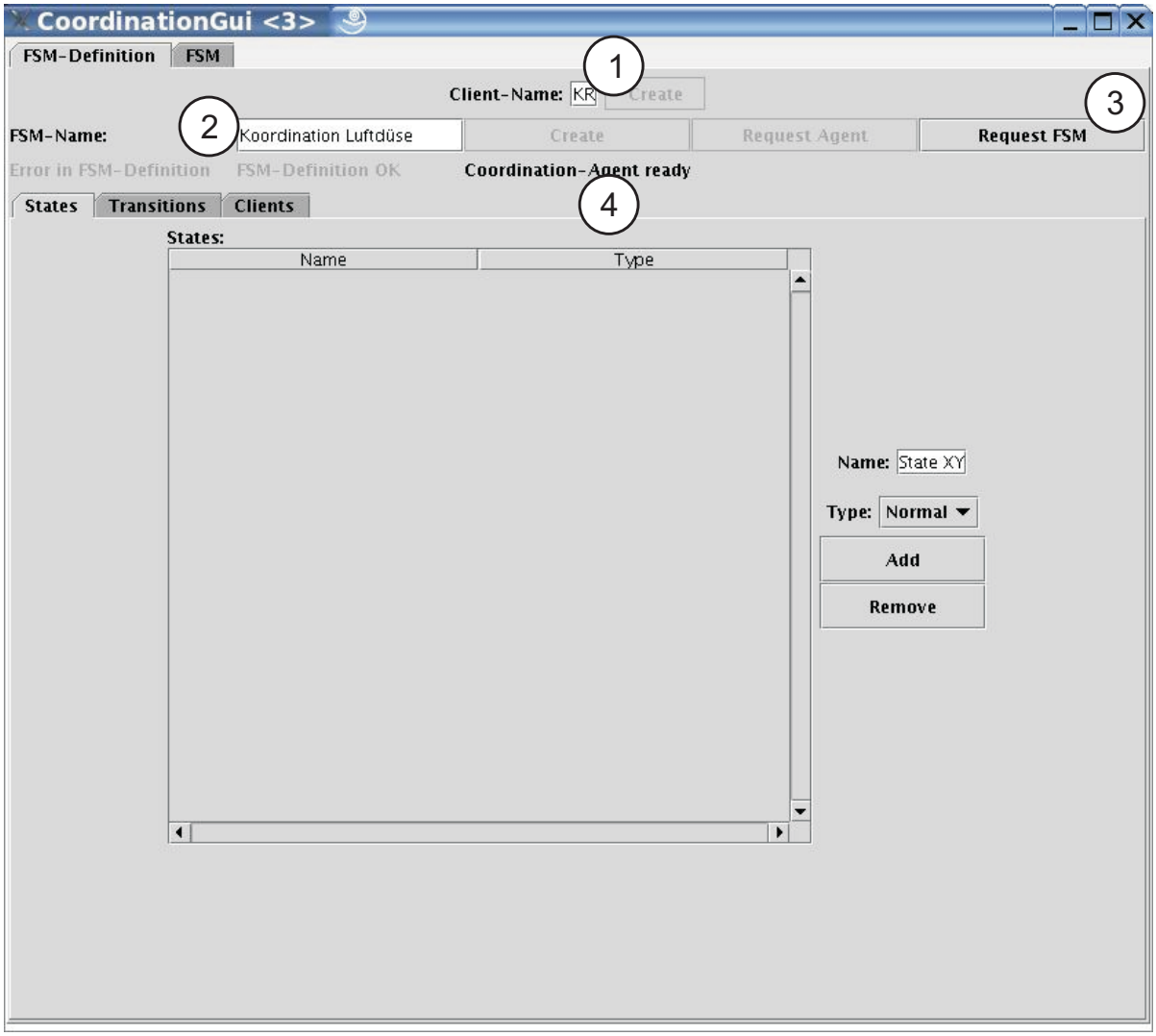


Bild 70: Screenshot: Anmeldung des Client KR (Kostenrechner) am Koordinations-Agenten

Nach der Erzeugung des Koordinations-Agenten durch den ersten Client können sich nun die restlichen drei Clients Prototypenbauer (PB), Kostenrechner (KR) und Qualitätsmanager (QM) an dem Koordinations-Agenten anmelden. In Bild 70 ist dies exemplarisch für den Client Kostenrechner (KR) durchgeführt. Auch hier wird zuerst der Client über das Texteingabefeld und die Schaltfläche *Create* (1) definiert, bevor der Name des Koordinationsprotokolls im Texteingabefeld (2) eingegeben wird. Nun wird allerdings nicht mit der Schaltfläche *Create* ein neuer Koordinations-Agent angefordert, sondern der Client meldet sich mit der Schaltfläche *Request*

FSM (3) an dem bestehenden Koordinations-Agenten an. Ist dieser vorhanden so wird dies durch das Anzeigeelement *Coordination-Agent ready* (4) veranschaulicht.

In den Abbildungen Bild 71 und Bild 72 ist für den Zustand *Qualität Bauteile geändert* des Koordinationsprotokolls *Koordination Luftdüse* die entsprechende Ausgabe des jeweiligen Clients für den Qualitätsmanager (QM) und den Prototypenbauer (PB) angegeben (siehe Anhang F.4.3). Der aktuelle Zustand ist im Textfeld *Currentstate* (1) dargestellt. Dieser Zustand kann auch jederzeit mit Hilfe der Schaltfläche *Request* (2) abgefragt werden. Dies wurde in diesem Beispiel vom Prototypenbauer (PB) gemacht. Im Texteingabefeld *Input* (3) ist die zuletzt getätigte Eingabe angezeigt. Diese wird mit Hilfe der Schaltfläche *Send* (4) an den Koordinations-Agenten geschickt. Die Anzeigeelemente *Input Ack* und *Wrong Input* (5) geben Auskunft darüber, ob die letzte Eingabe innerhalb des Koordinationsprotokolls zulässig war (siehe Bild 72) oder nicht (siehe Bild 71). Das Textfeld *Last Output* (6) beinhaltet die zuletzt empfangene Ausgabe. Im Falle vom Qualitätsmanager ist das die aktuelle Ausgabe, während dem Prototypenbauer noch eine Ausgabe eines früheren Zustandswechsels angezeigt wird.

Bild 71: Screenshot: Zustand *Qualität Bauteile geändert* des Client QM (Qualitätsmanager)

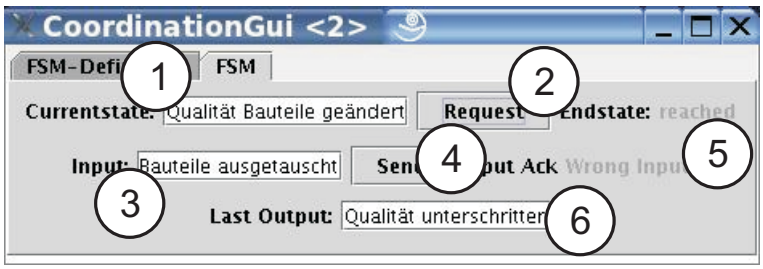


Bild 72: Screenshot: Zustand *Qualität Bauteile geändert* des Client PB (Prototypenbauer)

Die Abbildungen Bild 73 und Bild 74 stellen dasselbe Verhalten wie die beiden vorangegangenen Abbildungen für den Zustand *Projektplan überprüfen* und die Clients Kostenrechner (KR) und Projektplaner (PP) dar. In Bild 74 ist das leere Texteingabefeld *Input* (1) auffällig. Dies bedeutet, dass der Projektplaner bis zu diesem Zeitpunkt noch keinen Zustandswechsel initiiert hat.

Bild 73: Screenshot: Zustand *Projektplan überprüfen* des Client KR (Kostenrechner)

Bild 74: Screenshot: Zustand *Projektplan überprüfen* des Client PP (Projektplaner)

Ist nach erfolgreicher Koordination der Endzustand erreicht so wird dies, wie in Bild 75 für den Client des Prototypenbauers (PB) durch das Anzeigeelement *End-state reached* (1) angezeigt.

Bild 75: Screenshot: Endzustand *Projektplan anpassen* des Client PB (Prototypenbauer)

Die letzte Abbildung (Bild 76) in diesem Abschnitt zeigt den Master-Agenten mit der Liste (1) der aktuell sich in der RPD-Middleware befindlichen Agenten. Bei den Agenten vom Typ *LocalAgent* handelt es sich um die Client-Agenten, die von den RPD-Anwendungen genutzt werden (siehe Abschnitt 7.1). Der dritte Agent der Liste ist der Koordinations-Agent.

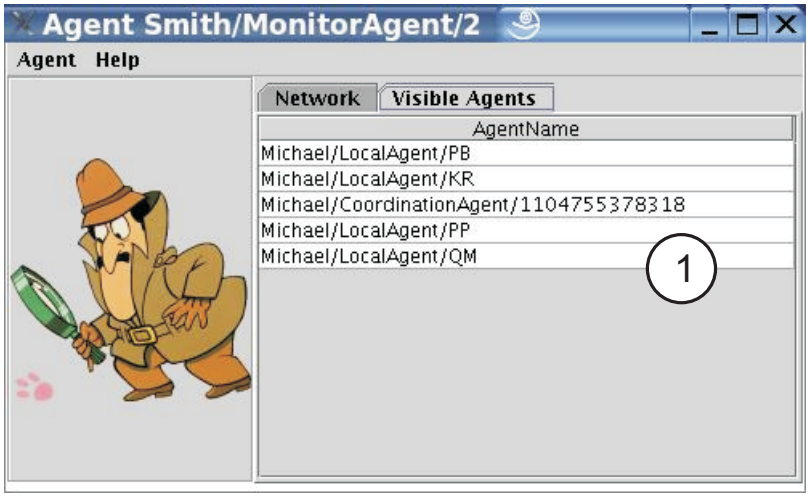


Bild 76: Screenshot: Master-Agent

F.4.3 Koordinationsprotokoll (Sequenzdiagramm)

Den Abschluss der genauen Beschreibung des Koordinations-Agenten bildet das Sequenzdiagramm (siehe Bild 81, Bild 82, Bild 83 und Bild 84), das den Nachrichtenaustausch zwischen RPD-Anwendung und dem Koordinations-Agenten darstellt und sich an dem Anwendungsbeispiel aus Abschnitt 8.6 orientiert. Die Pfeile, die den Koordinationsweg repräsentieren, der im Sequenzdiagramm beschrieben ist, sind in den Zustandübergangsdiagrammen (siehe Bild 77, Bild 78, Bild 79 und Bild 80) fett dargestellt. Die Zustandsübergänge (A) bis (Q) in den Zustandübergangsdiagrammen finden sich auch in den Sequenzdiagrammen an den Stellen, wo die entsprechenden Nachrichten ausgetauscht werden.

Die Koordination startet in Bild 77 und Bild 80 durch Entdeckung eines Qualitätsverlustes beim Prototypen (A). Woraufhin der Prototypenbauer (PB) das Material des Prototypen ändert (B) und der Qualitätsmanager (QM) nach erneuter Überprüfung die Qualität für ausreichend erachtet (C). Daraufhin wird eine Kostenüberprüfung notwendig (D), die im Koordinationsbereich des Kostenrechners (KR) (siehe Bild 79) durchgeführt wird. Eine Verletzung des Kostenplans wird erkannt (E), wodurch der Prototypenbauer (PB) sein Material erneut ändern muss (F) und damit eine erneute Qualitätsprüfung initiiert (G). Der Qualitätsmanager (QM) erkennt auch hier eine Qualitätsunterschreitung (H) (siehe Bild 80), die durch den Prototypenbauer (PB) durch den Austausch von Bauteilen innerhalb des Prototyps behoben wird (I). Die Qualitätsüberprüfung (J) ist erfolgreich und führt zu einer erneuten Kostenüberprüfung (K). Der Kostenrechner (KR) erkennt, dass der Kostenplan eingehalten wurde (L) (siehe Bild 79) und fordert eine Projektplanprüfung. Die Projektplanprüfung erfolgt im Bereich des Projektplaners (PP) (siehe Bild 78) und ergibt, dass der Projektplan nicht mehr eingehalten werden kann (M). Der Projektplaner (PP) schlägt nun einen neuen Fertigstellungszeitpunkt dem Prototypenbauer (PB) vor (N), der von diesem abgelehnt wird (O), da er zu kurzfristig ist. Der Prototypen-

bauer (PB) schlägt nun seinerseits einen neuen Fertigstellungszeitpunkt vor (P), den der Projektplaner (PP) annimmt (Q), womit die Koordination abgeschlossen ist.

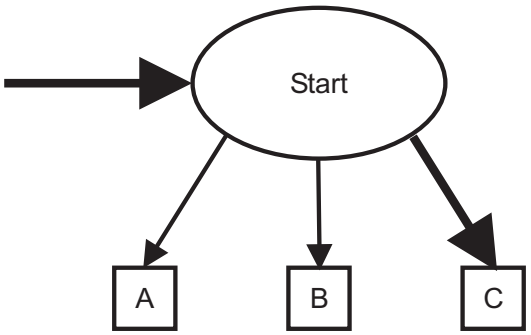


Bild 77: Abarbeitung Koordinationsprotokoll: Start

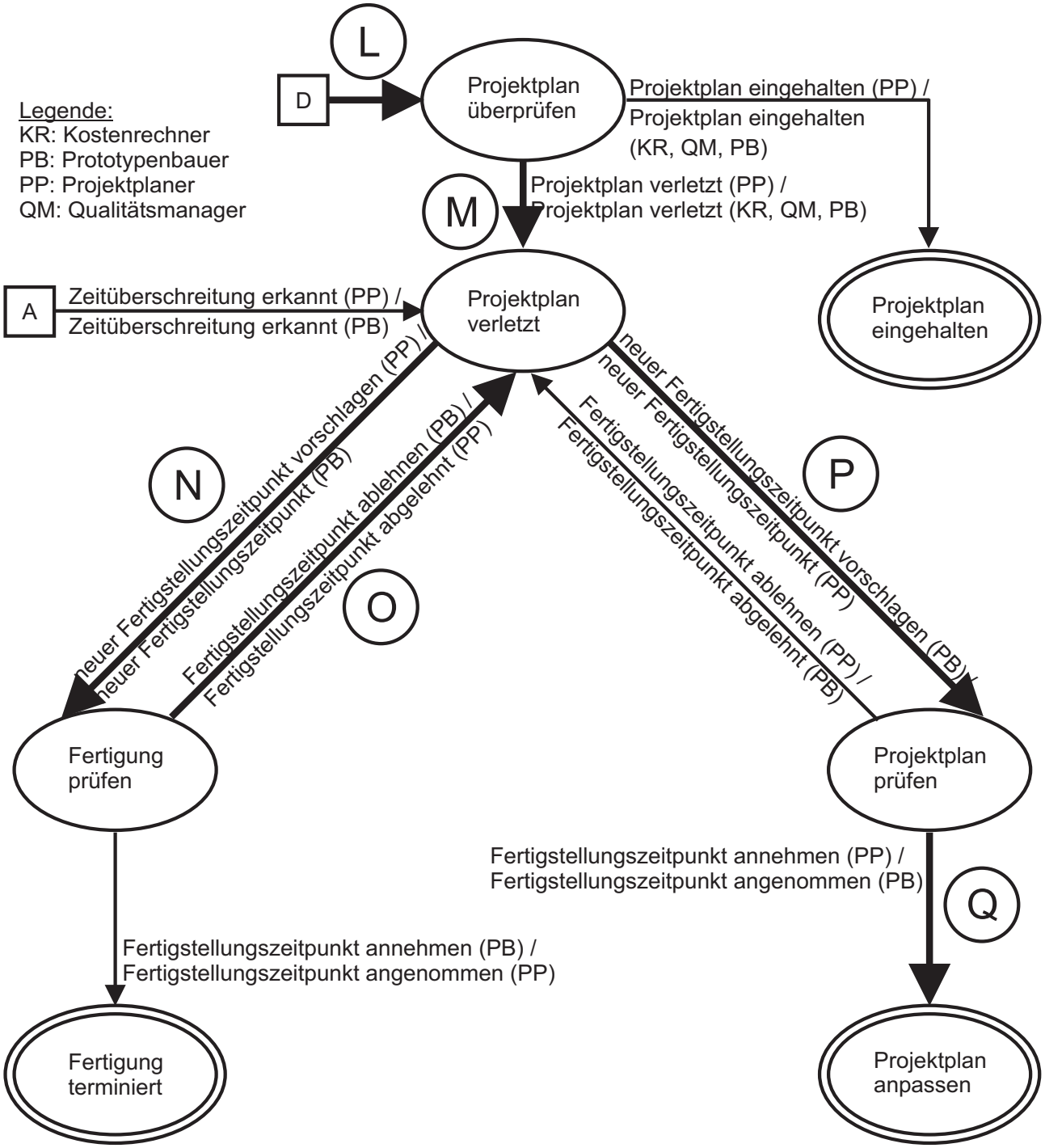


Bild 78: Abarbeitung Koordinationsprotokoll: Projektplaner

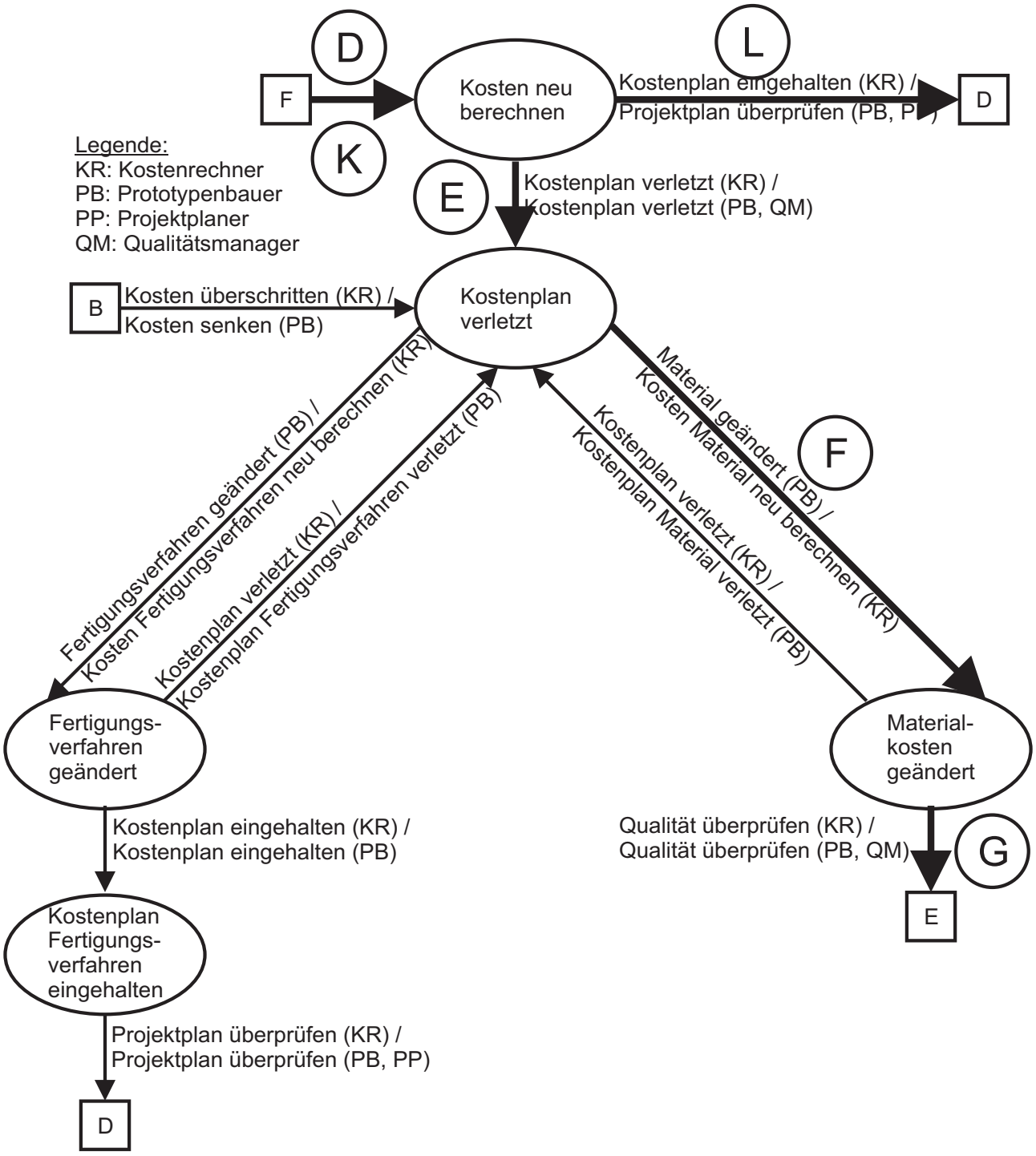


Bild 79: Abarbeitung Koordinationsprotokoll: Kostenrechner

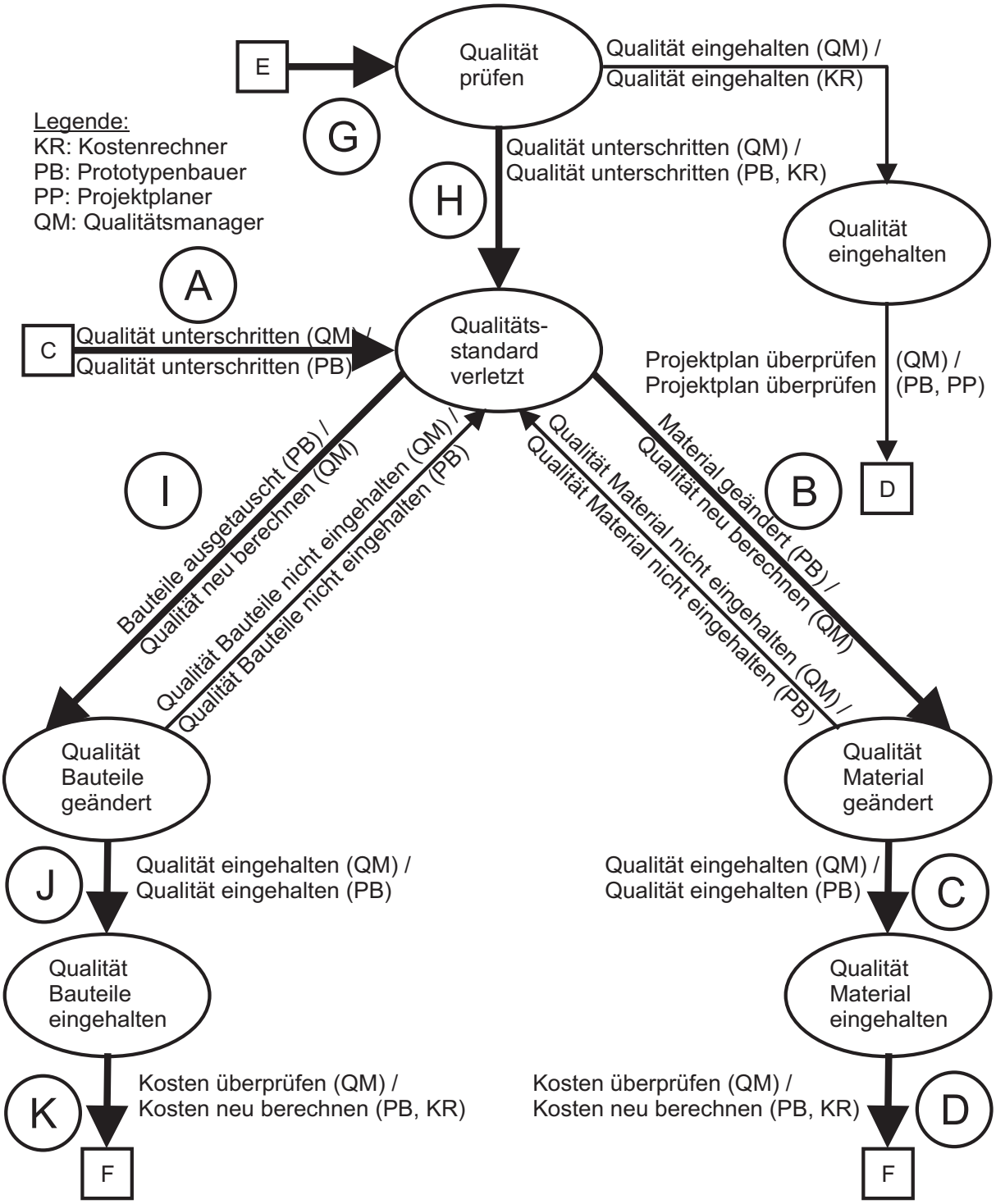


Bild 80: Abarbeitung Koordinationsprotokoll: Qualitätsmanager

In den Abbildungen Bild 81, Bild 82, Bild 83 und Bild 84 ist das Sequenzdiagramm für den beschriebenen Koordinationsablauf dargestellt. In der Dienstanforderungsphase A (siehe Bild 81) fordert der Client des Projektplaners (PP) im Schritt (1) einen neuen Koordinations-Agenten bei der RPD-Middleware an. Dieser wird vom Master-Agenten instanziiert und die Anforderung des Clients an den Koordinations-

Agenten weitergeleitet. Der Koordinations-Agent teilt dem Client dann mit der Nachricht *Initiate-Coordinations-Agent-Ack* seine Empfangsbereitschaft mit. Der Projektplaner (PP) übermittelt nun dem Koordinations-Agenten das Koordinationsprotokoll (2). Dieses wird vom Koordinations-Agenten im Schritt (3) syntaktisch und semantisch überprüft und dem Projektplaner (PP) bestätigt. Die Dienstanforderungsphase wird durch das Anmelden der Clients des Kostenrechners (KR), des Prototypenbauers (PB) und des Qualitätsmanagers (QM) im Schritt (4) abgeschlossen.

In der Dienstbearbeitungsphase B sind die einzelnen Nachrichten dargestellt, die für die Zustandswechsel wie oben beschrieben benötigt werden. In Bild 82 sind die Zustandswechsel A–F dargestellt:

- **A**: Qualitätsmanager (QM) meldet Qualität unterschritten an Prototypenbauer (PB).
- **B**: Prototypenbauer (PB) ändert daraufhin das Material des Prototyps und fordert den Qualitätsmanager (QM) auf die Qualität neu zu berechnen.
- **C**: Der Qualitätsmanager (QM) stellt fest, dass die Qualität eingehalten ist, und teilt dies dem Prototypenbauer (PB) mit.
- **D**: Der Qualitätsmanager (QM) fordert den Kostenrechner (KR) auf, die Kosten neu zu berechnen und informiert den Prototypenbauer (PB) darüber.
- **E**: Der Kostenrechner (KR) stellt fest, dass der Kostenplan verletzt wurde und meldet dies dem Qualitätsmanager (QM) und Prototypenbauer (PB).
- **F**: Daraufhin ändert der Prototypenbauer (PB) erneut das Material des Prototyps und fordert den Kostenrechner (KR) erneut auf den Kostenplan zu überprüfen.

In Bild 83 sind die folgenden Zustandswechsel abgebildet:

- **G**: Der Kostenrechner (KR) fordert den Qualitätsmanager (QM) auf, die Qualität neu zu bestimmen und informiert den Prototypenbauer (PB) über diese Vorgehensweise.
- **H**: Der Qualitätsmanager (QM) stellt erneut eine Qualitätsunterschreitung fest und informiert den Kostenrechner (KR) und den Prototypenbauer (PB).
- **I**: Der Prototypenbauer (PB) tauscht daraufhin einzelne Bauteile des Prototyps aus und fordert den Qualitätsmanager (QM) auf die Qualität neu zu bestimmen.
- **J**: Der Qualitätsmanager (QM) stellt die Richtigkeit der Qualität fest und setzt den Prototypenbauer (PB) darüber in Kenntnis.
- **K**: Der Qualitätsmanager (QM) fordert den Kostenrechner (KR) erneut zu Kostenüberprüfung auf und informiert den Prototypenbauer (PB).
- **L**: Der Kostenrechner (KR) stellt fest, dass der Kostenplan eingehalten wurde, und fordert vom Projektplaner (PP) einer Überprüfung des Projektplans. Des

Weiteren setzt er den Prototypenbauer (PB) über diese Vorgehensweise in Kenntnis.

In Bild 84 sind die folgenden Zustandswechsel abgebildet:

- **M**: Der Projektplaner entdeckt eine Verletzung des Projektplans und teilt dies dem Kostenrechner (KR), dem Qualitätsmanager (QM) und dem Prototypenbauer (PB) mit.
- **N**: Des Weiteren schlägt der Projektplaner (PP) einen neuen Fertigstellungszeitpunkt dem Prototypenbauer (PB) vor.
- **O**: Dieser neue Fertigstellungszeitpunkt wird vom Prototypenbauer (PB) als zu kurzfristig gegenüber dem Projektplaner (PP) abgelehnt.
- **P**: Daraufhin schlägt der Prototypenbauer (PB) dem Projektplaner (PP) einen neuen Fertigstellungszeitpunkt vor.
- **Q**: Dieser wird vom Projektplaner (PP) angenommen und der Prototypenbauer (PP) über die Annahme in Kenntnis gesetzt.

Mit Ablauf des Koordinationsprotokolls wird der Koordinations-Agent im Schritt (5) beendet.

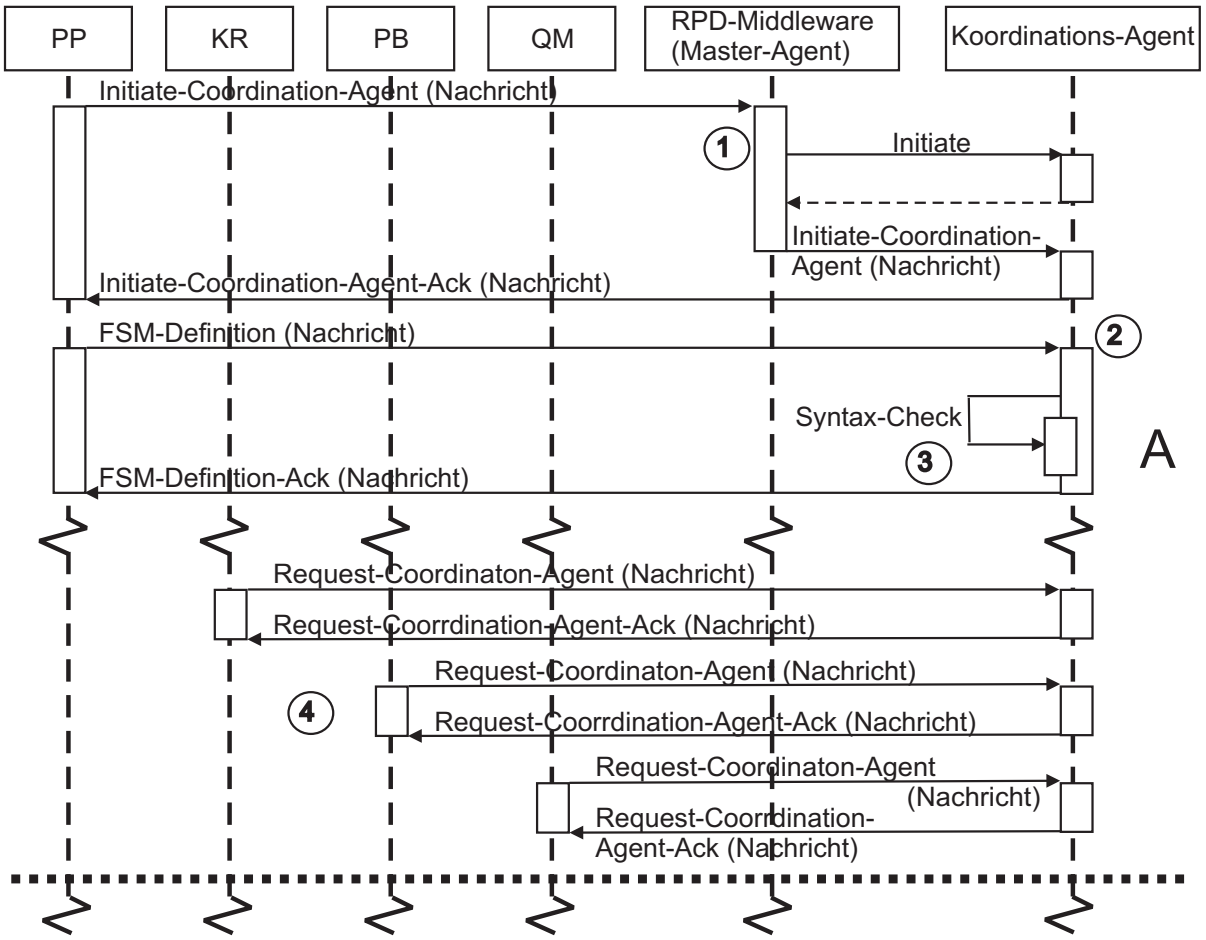


Bild 81: Sequenzdiagramm: Koordinations-Agent (1)

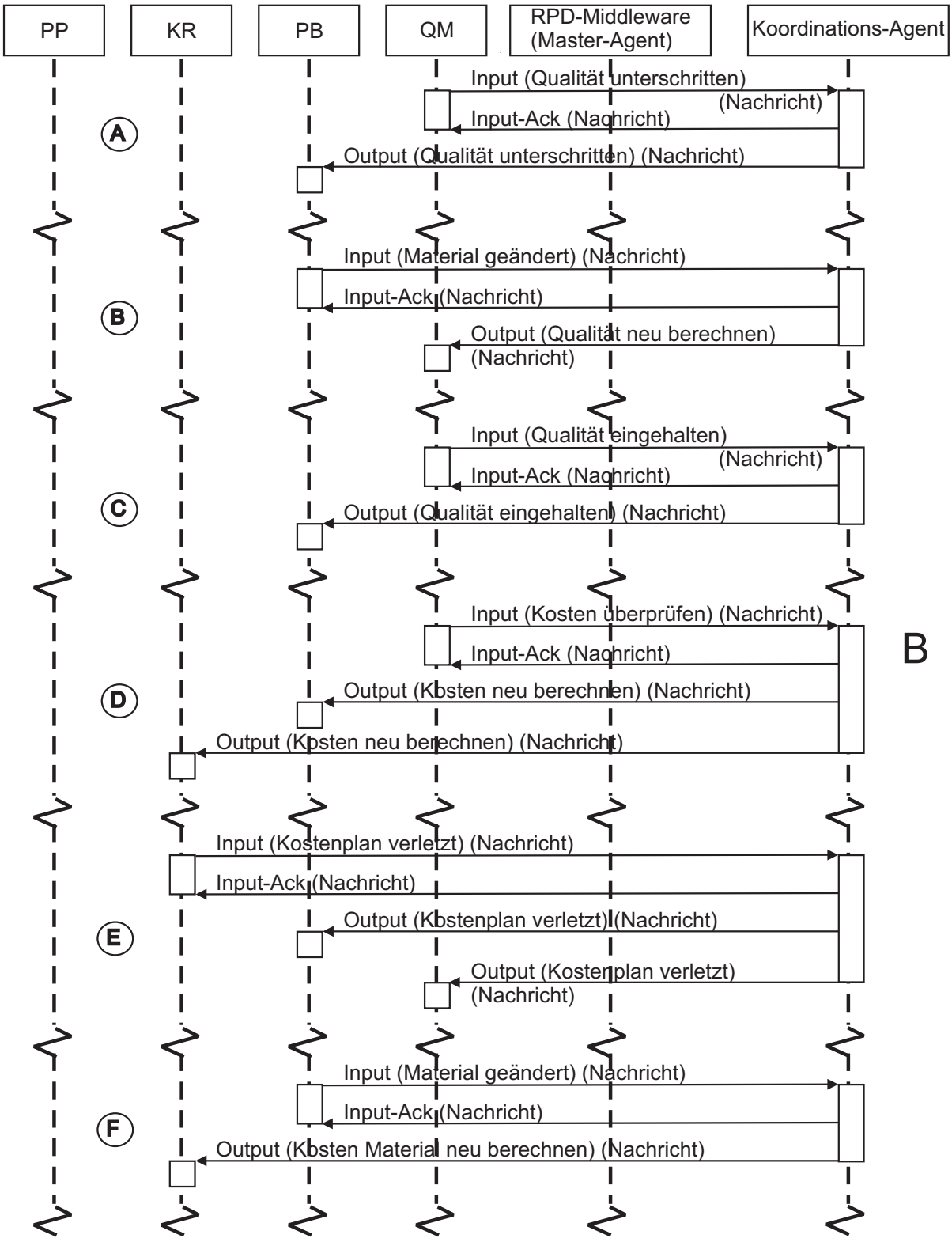


Bild 82: Sequenzdiagramm: Koordinations-Agent (2)

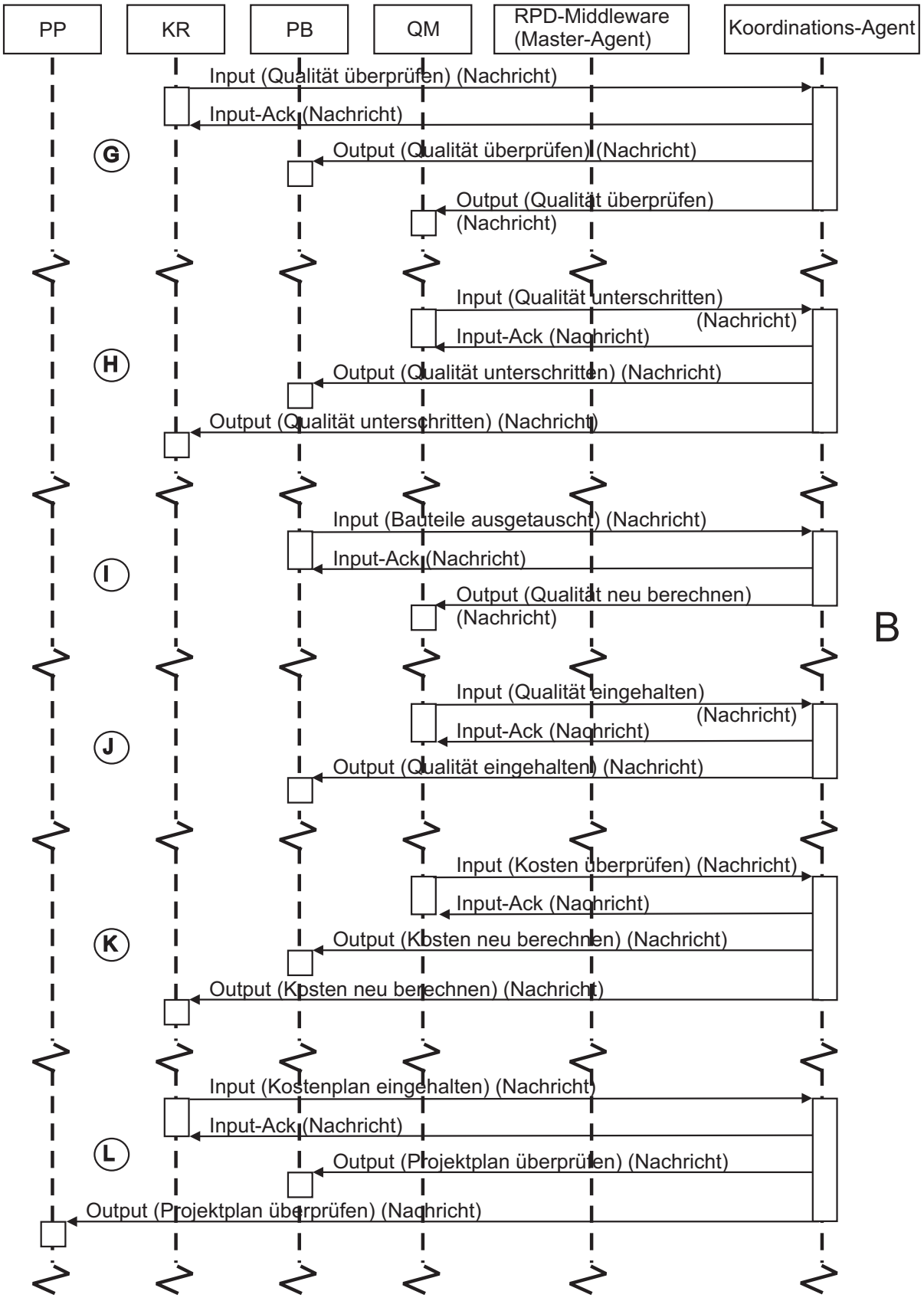


Bild 83: Sequenzdiagramm: Koordinations-Agent (3)

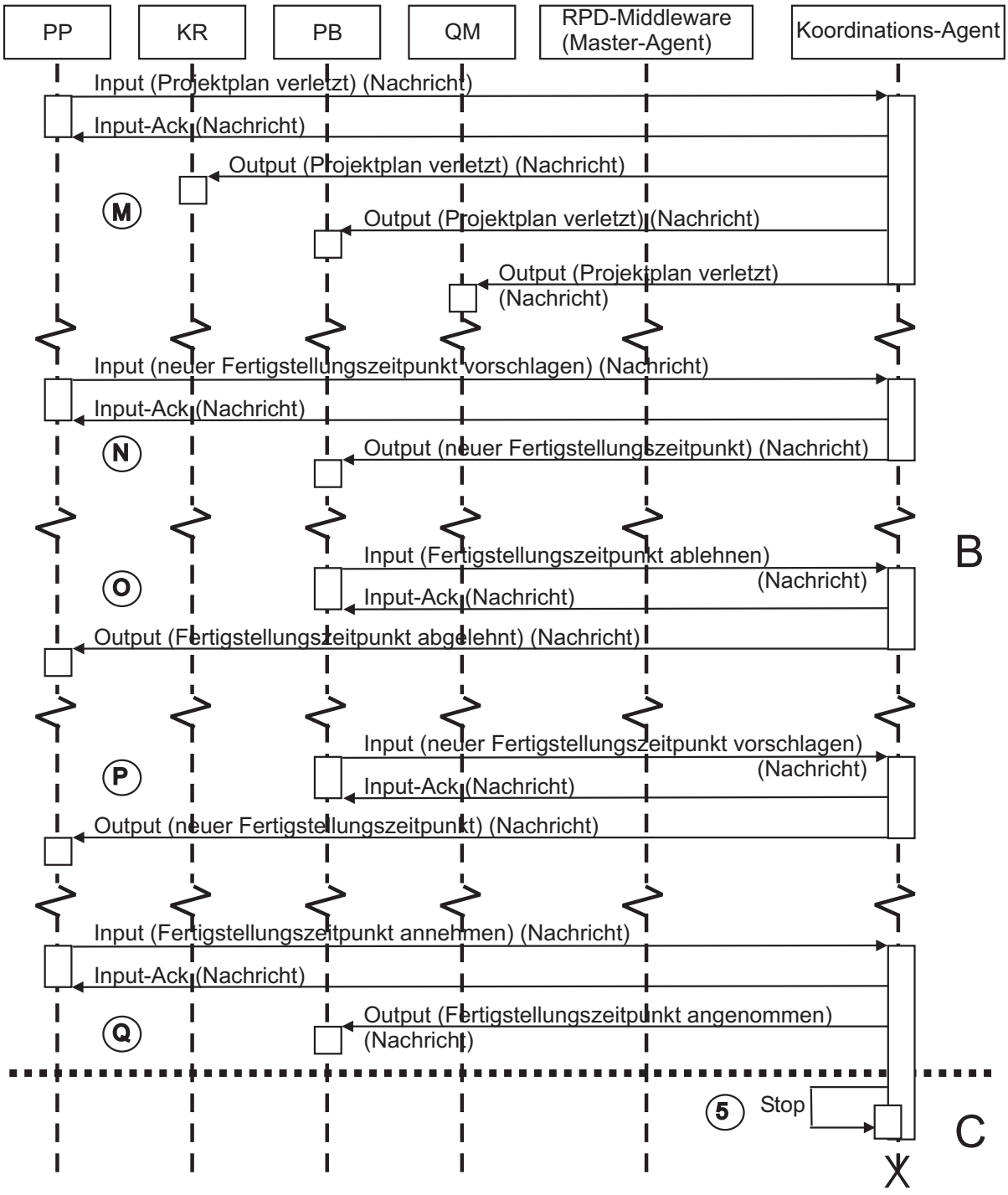


Bild 84: Sequenzdiagramm: Koordinations-Agent (4)

Lebenslauf

Persönliche Daten

Name	Michael Karsten Diederich
Geburtstag	16.03.1972
Geburtsort	Stuttgart
Staatsangehörigkeit	deutsch
Familienstand	ledig

Ausbildung

1979 - 1983	Grundschule Österfeld, Stuttgart
1983 - 1992	Fanny-Leicht-Gymnasium, Stuttgart
Mai 1992	Abschluss mit Abitur
1992 - 1997	Diplomstudium der Informatik, Universität Stuttgart Nebenfach Elektrotechnik
1995 - 1997	Wissenschaftliche Hilfstätigkeit, Institut für Arbeitswissenschaften und Technologiemanagement, Universität Stuttgart
1998	Allgemeine Grundausbildung, 4./ABCAbwLBtl 210, Sonthofen Stabsdienstsoldat, StammKp - ABC/SeS, Sonthofen

Beruf

1999 – Februar 2006	Wissenschaftlicher Mitarbeiter im Marktstrategieteam „Enterprise Networking“ am Institut für Arbeitswissenschaft und Technologiemanagement (IAT), Universität Stuttgart
seit März 2007	Leiter des Dienstleistungsteam „IuK-Dienste“ am Institut für Arbeitswissenschaft und Technologiemanagement (IAT), Universität Stuttgart und Fraunhofer Institut für Arbeitswirtschaft und Organisation (IAO)

Stuttgart, im März 2007