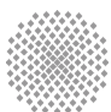
IKE Institut für Kernenergetik
und Energiesysteme

Entwicklung und Erprobung eines Dienstleistungskonzepts zur Integration von Simulationen in die Kernreaktor-Fernüberwachung

Axel Grohmann

Januar 2002 IKE 4-155



Universität Stuttgart

IKE Institut für Kernenergetik
und Energiesysteme

Entwicklung und Erprobung eines Dienstleistungskonzepts zur Integration von Simulationen in die Kernreaktor-Fernüberwachung

Von der Fakultät Energietechnik der Universität Stuttgart
zur Erlangung der Würde eines Doktors (Dr.-Ing.)
genehmigte Abhandlung

vorgelegt von

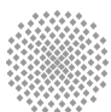
Axel Grohmann

geboren in Kufstein/Tirol

Hauptberichter: Prof. Dr.-Ing. habil. F. Schmidt
Mitberichter: Prof. Dr.-Ing. A. Voß

Tag der Einreichung: 30. Januar 2001
Tag der mündlichen Prüfung: 11. Januar 2002

Januar 2002 IKE 4-155


Universität Stuttgart

Kurzfassung

Das in dieser Arbeit vorgestellte Dienstleistungskonzept ermöglicht eine einfache und neuartige Integration von Software-Komponenten in komplexen Software-Systemen. Funktionalitäten der einzelnen Komponenten werden als Dienstleistungen angeboten und können von anderen Komponenten nachgefragt werden. Dienstleistungen werden syntaktisch und protokollar einheitlich aufgerufen, was eine große Flexibilität und den Einsatz in Systemen ermöglicht, die situationsbedingt agieren, reagieren und kommunizieren können müssen. Laufende Dienstleistungen können angehalten, wieder aufgenommen und abgebrochen werden.

Die Umsetzung erfolgte in C++, stützt sich auf CORBA als Verteilungsmechanismus, ist voll multi-threading-fähig und verfügt über die Fähigkeit, nach einem Ausfall verlorengegangene Dienstleistungen wiederherzustellen und in das laufende System einzugliedern. Das Dienstleistungsframework wurde unter den Gesichtspunkten Flexibilität, Erweiterbarkeit, Klarheit und unter Ausnutzung moderner, objektorientierter Software-Technologien entwickelt. Es ist vom jeweiligen Anwendungsgebiet und den übertragenen Informationsobjekten unabhängig.

Das Dienstleistungsframework wird im komplexen System der Kernreaktor-Fernüberwachung eingesetzt. In diesem sicherheitskritischen Überwachungssystem mussten die Einzelschritte zur Simulation der Ausbreitung, Deposition und Wirkung von luftgetragenen, radioaktiven Nukliden bisher von Menschen koordiniert werden. Diese Einzelschritte werden nun als rechnergestützte Informationsdienstleistungen in das System integriert und sind ohne weiteres menschliches Zutun ständig nutzbar. Als Makroarchitektur wurde für das System ein am Institut konzipiertes und entwickeltes Multi-Agenten-System gewählt. Dabei ist das Dienstleistungsframework das verbindende Rückgrat des Multi-Agenten-Systems.

Erfahrungen auf Systemebene werden diskutiert. Dabei geht es vor allem um die Verteilung von Systemwissen und den damit verbundenen Wartungsaufwand und die Gegenüberstellung von eher prozeduralen und eher regelbasierten Vorgängen.

Abstract

In this dissertation a framework for distributed software services is developed. This allows the easy integration of software components in complex systems. The capabilites of the components are offered as services, which can be consumed by others. Services share a common and generic interface, which increases flexibility and the potential use in systems that have to act, react and communicate flexibly. Services can be suspended, resumed and terminated at run-time.

The service framework has been implemented in C++ and uses CORBA as distribution mechanism. It is fully multi-threaded and provides recovery mechanisms in order to safe computation time after a crash. The implementation aims at high flexibility, extensibility, clarity and the use of modern, object-oriented software technologies. The framework is independent of the actual domain and of the data objects that need to be exchanged.

The service framework is used in the large-scale Remote Nuclear Reactor Monitoring System of Baden-Württemberg. This mission-critical system simulates the dispersion and deposition of radioactive nuclides and their effects on humans. In the past, the single steps of the simulation had to be co-ordinated by humans. Now, they are provided as automated information services, integrated into the system and are always available. For the system, a special multi-agent system has been employed. The service framework is the backbone of that multi-agent system.

Some experiences and insights on the system level are discussed, which cover mainly the distribution of system knowledge and its impact on maintenance costs as well as procedural vs. rule-based problems and approaches.

Danksagungen

Mein Dank für die finanzielle Unterstützung dieser Promotionsarbeit gebührt der Deutschen Forschungsgemeinschaft, welche im Rahmen des Graduiertenkollegs Parallele und Verteilte Systeme (GK PVS) an der Universität Stuttgart diese Promotion gefördert hat. Darüber hinaus hat das Institut für Kernenergetik und Energiesysteme diese Arbeit unterstützt.

Roland Kopetzky hat mit seinem Konzept des Logischen Klienten den Anstoß und viele weitere Anregungen zu dieser Arbeit geliefert. Dafür sei ihm herzlich gedankt.

Für viele fundierte Ratschläge, den Meinungsaustausch und die ausgezeichnete Zusammenarbeit mit meinem Kollegen und Freund Kurt De Marco bin ich sehr dankbar.

Zuletzt sei noch meinem Betreuer und Abteilungsleiter, Herrn Prof. Fritz Schmidt unter vielem anderem auch dafür gedankt, dass er es verstanden hat, auch in für das Institut für Kernenergetik und Energiesystemen schwierigen Zeiten das Abteilungs- und Arbeitsklima freundschaftlich-kollegial und produktiv zu halten.

INHALT

1 EINLEITUNG 1

1.1 RAHMENBEDINGUNGEN 2

1.2 PRINZIPIEN UND ANFORDERUNGEN 4

1.3 AUFBAU DIESER ARBEIT 6

2 AUSGANGSPUNKTE 7

2.1 AUFGABEN DER SIMULATION BEI DER KERNREAKTOR-FERNÜBERWACHUNG (KFÜ) 7

2.2 DIENSTE- UND AGENTENBASIERTE SYSTEME 9

2.3 DER LOGISCHE KLIENT 11

2.4 ANWENDUNG DES LOGISCHEN KLIENTEN IN DER AUSBREITUNGSRECHNUNG DER KFÜ 21

2.5 EINBINDUNG DER AUSBREITUNGSRECHNUNG IN DIE KFÜ 24

2.6 VERGLEICHBARE SYSTEME 27

3 DIE SAL ALS DIENSTLEISTUNGSFRAMEWORK 33

3.1 NACHRICHTENÜBERTRAGUNG 34

3.2 DAS DIENSTLEISTUNGSKONZEPT IN DER SAL 38

3.3 TRANSAKTIONSPROTOKOLL UND DIENSTLEISTUNGSZUSTÄNDE 43

3.4 ÜBERTRAGUNG VON INFORMATIONEN 47

3.5 INTERNE STRUKTUR DER SAL 54

3.6 FEHLERBEHANDLUNG 65

3.7 PROTOKOLLIERUNG 67

3.8 PERSISTENZ 68

3.9 ALLGEMEINE DIENSTLEISTUNGEN 69

3.10 VERWENDUNG DER SAL 70

4 DAS DIENSTLEISTUNGSKONZEPT IN DER PRAXIS 73

4.1 ERFAHRUNGEN IN DER KFÜ 73

4.2 SEMANTISCHE ABHÄNGIGKEIT VON DIENSTERBRINGER UND DIENSTNUTZER 75

4.3 LEBENSDAUER UND LEBENSRAUM VON INFORMATIONSOBJEKTEN 77

4.4 DIENSTLEISTUNGEN UND DAS PROZEDURALE PROGRAMMIERPARADIGMA 78

5 DIENSTE IN REGELBASIERTEN SYSTEMEN 80

5.1 REGELBASIERTE AUFTRAGSVERGABE 80

5.2 EXKURS: DATEN – INFORMATIONEN – WISSEN 81

5.3 AUSBAUMÖGLICHKEITEN DER KFÜ 83

6 ZUSAMMENFASSUNG 85

6.1 DIE SAL ALS DIENSTLEISTUNGSFRAMEWORK 85

6.2 ERFAHRUNGEN MIT DER SAL 86

6.3 ANMERKUNGEN ZUR ENTWICKLUNG VON AGENTENSYSTEMEN 87

7 LITERATURVERZEICHNIS 88

Verzeichnis der Tabellen

Tabelle 1: Von der SAL verwendete Nachrichtentypen 45

Tabelle 2: Zusammenhänge zwischen Fehlerkategorien und Rückgabeperformativen.................66

Verzeichnis der Bilder

Abbildung 1: Verschiedene Arten der Anbindung eines Dienstes 12

Abbildung 2: Makroarchitektur des Logischen Klienten...13

Abbildung 3: Ein einfacher Beispiel-Workflow...16

Abbildung 4: Daten- und Kontrollfluss bei Verwendung des Repositories 18

Abbildung 5: Übersicht der verwendeten Dienste in der Ausbreitungsrechnung..........................22

Abbildung 6: Architektur der KFÜ Baden-Württemberg (Quelle debis Systemhaus) 25

Abbildung 7: Nutzung der Ausbreitungsrechnung in der KFÜ und die Rolle des Dienstleistungsframeworks im ABR-System.. 26

Abbildung 8: Ebenen der Kommunikation...33

Abbildung 9: Eine typische KQML-Nachricht...35

Abbildung 10: Nachrichtenfolgen bei der Verwendung von ask-all und ask-one (links) bzw. bei stream-all (rechts).. 44

Abbildung 11: Zustandsübergänge und Nachrichtenfolgen. Nachrichtentypen nach Tab. 1.........46

Abbildung 12: Zusammenspiel zwischen Ontologie und SAL beim Serialisieren und Deserialisieren.. 50

Abbildung 13: Basisklassen der SAL..54

Abbildung 14: Gemeinsamkeiten von Jobs und Requests...55

Abbildung 15: CSAL_Job und das Zusammenspiel mit dem CSAL_Core..................................57

Abbildung 16: Blockierende und nicht blockierende Aufträge 58

Abbildung 17: Erzeugung von Jobs nach dem Factory-Entwurfsmuster [3]................................60

Abbildung 18: Senden und Empfangen von CORBA-Nachrichten 61

Abbildung 19: Aufgaben des CSAL_StandardCore...63

Abbildung 20: Erzeugung der SAL durch eine Builder-Klasse..65

Abbildung 21: Volumen und Wert von Daten und Informationen nach [53]...............................82

Verzeichnis der Abkürzungen

ABR	Ausbreitungsrechnung
CLIPS	C Language Integrated Production Shell, eine Umgebung für regelbasierte Systeme, die in C geschrieben ist und in C- Programme leicht integriert werden kann [51] [51].
COM	Component Object Model, eine von Microsoft entwickelte Komponententechnologie.
CORBA	Common Object Request Broker Architecture, ein von der OMG entwickelter Standard zur Verteilung von Objekten [29] [30], vgl. Kapitel 2.6.1.
CSAL	Präfix für Namen von Klassen, die im Rahmen der SAL erstellt wurden und von Benutzern der SAL verwendet werden.
CSCW	Computer-Supported Cooperative Work, Technologien für die computergestützte Tele-Zusammenarbeit zwischen Menschen.
DCOM	Distributed Component Object Model, Erweiterung von COM für verteilte Systeme.
FIPA	Foundation for Intelligent Physical Agents, eine Organisation, welche Standards im Bereich von Agenten entwickelt [42], vgl. Kapitel 2.6.5.
HTML	Hypertext Markup Language, ein Standard zur Beschreibung von für Menschen lesbaren Seiten, welcher z.B. im Internet enorme Verbreitung gefunden hat.
IKE	Institut für Kernenergetik und Energiesysteme der Universität Stuttgart, an welchem diese Arbeit entstanden ist.
J2EE	Java 2 Enterprise Edition, eine von der Firma Sun standardisierte und entwickelte Umgebung für Java-basierte E-Business-Anwendungen, vgl. Kapitel 2.6.2.
JDBC	Java Data Base Connectivity, ein Standard, um über Java auf Datenbanken zugreifen zu können.
KFÜ	Kernreaktor-Fernüberwachung, ein gesetzlich vorgeschriebenes Informationssystem für Kernreaktoren, vgl. Kapitel 2.1.
KIF	Knowledge Interchange Formalism, eine Sprache zur Wissensrepräsentation, vgl. [44].
KQML	Knowledge Query and Manipulation Language, ein Standardisierungsvorschlag für eine Kommunikationssprache zwischen Agenten [43], vgl. Kapitel 3.1 und 3.2.
OMG	Object Management Group, ein firmenübergreifendes Konsortium zur Entwicklung von Standards im Bereich der Objektorientierung [27].

RTTI	Run-Time Type Identification, ein relativ spät in den C++ - Standard aufgenommener Mechanismus zum Ermitteln der Klassenzugehörigkeit von Objekten zur Laufzeit.
SAL	Service Agent Layer, das im Rahmen dieser Arbeit erstellte Dienstleistungsframework.
UML	Unified Modelling Language, ein von der OMG entwickelter Standard zur grafischen Repräsentation von objektorientierten Strukturen [24] [27].
WPDL	Workflow Process Definition Language, ein von der Workflow Management Coalition vorgeschlagenes Austauschformat für Workflow-Beschreibungen [24].
ZDH	Zentrale Datenhaltung, eine Komponente der KFÜ, vgl. Kapitel 2.4.

1 Einleitung

Im Alltag haben wir es mit einer Fülle an verschiedenen Dienstleistungen zu tun, doch trotz der enormen Unterschiede zwischen ihnen fällt es uns nicht schwer, mit Dienstleistungen umzugehen. Dies liegt nicht zuletzt daran, dass in unserer Gesellschaft ein Konsens darüber besteht, wie Dienstleistungen angeboten, nachgefragt, beauftragt, abgewickelt und bezahlt werden. Im Bürgerlichen Gesetzbuch werden die rechtlichen Grundlagen dazu in den Paragraphen 631ff festgelegt. Diese Standardisierung ist eine wichtige Voraussetzung, um die Fülle an Dienstleistungen effizient nutzen zu können, ohne sich jedesmal neu in andere Rahmenbedingungen und Gepflogenheiten einarbeiten zu müssen.

Im Softwarebereich gibt es in Teilbereichen ähnliche Standards. So ist beispielsweise die Hypertext Markup Language (HTML) ein akzeptierter Standard zur Beschreibung von für Menschen lesbaren Seiten. Die Browsertechnologie stellt Werkzeuge zur Visualisierung solcher Seiten zur Verfügung, das TCP/IP-Protokoll regelt ihre Verteilung auf Anfrage, und das Internet stellt eine Hardware- und Netzinfrastruktur bereit, mittels derer HTML-Seiten weltweit abgefragt werden können. Durch diese Standards und Voraussetzungen ist es nun leicht möglich, Informationen global verfügbar zu machen und abzufragen, und zwar ungeachtet der enormen Fülle an Informationen und deren Verschiedenartigkeit.

Auch numerische Softwarebibliotheken sind in ihrer Verwendung sehr ähnlich. Sie bieten ihre Dienste in Form von Funktionsaufrufen an, die in den Anwendungscode eingebunden werden können. Das grundsätzliche Vorgehen ist immer dasselbe, wenn auch die Funktionen andere Namen, Übergabeparameter und Bedeutung haben. So bieten numerische Softwarebibliotheken dem Entwickler komfortable Dienste bei der Lösung numerischer Probleme.

Bei der Simulation komplexer technischer Systeme reichen Funktionsaufrufe nicht mehr aus. Dies hat mehrere Gründe: Die Zahl der Ein- und Ausgabedaten für die einzelnen Teilprobleme ist so groß, dass Funktionsaufrufe unpraktikabel werden. Ein weiteres Problem liegt darin, dass die Simulation nicht mehr innerhalb eines Prozesses abläuft, sondern in verteilten Prozessen. Es wird also ein Verteilungskonzept notwendig. Außerdem sind die einzelnen Aktivitäten umfangreich und dauern lange. Ein Funktionsaufruf kann nach seinem Absetzen nicht mehr beeinflußt werden. Doch genau dies wird in komplexen Systemen notwendig: Es kann sinnvoll sein, Rechnungen anzuhalten, wiederaufzunehmen oder abzubrechen, ihren Zustand zu überwachen oder Zwischenergebnisse abzufragen. Und schließlich ist – im Gegensatz zu numerischen Softwarebibliotheken, wo hinsichtlich der Bedeutung der einzelnen Funktionen und Übergabeparameter ein einheitliches Verständnis herrscht – bei komplexen Simulationssystemen oft nicht ausreichend geklärt, was die einzelnen Teile exakt tun, welche Bedeutung die Ein- und Ausgabedaten tatsächlich haben und welche Nebenbedingungen erfüllt sein müssen.

All diese Probleme sind zwar gelöst bzw. lösbar, aber die Lösungswege sind uneinheitlich. Damit tritt bei der Simulation komplexer technischer Systeme ein neues Problem in den Vordergrund, nämlich die große Heterogenität der einzelnen Komponenten. Dies betrifft Plattformabhängigkeiten, technische Aspekte, Verteilungsstrategien, Aufrufkonventionen und die oft nicht einmal explizit vorhandene Beschreibung der bearbeiteten Daten und Vorgänge.

Im Rahmen dieser Arbeit wird nun versucht, das im Alltag so erfolgreiche Konzept der Dienstleistungen auf die Simulation komplexer technischer Systeme zu übertragen. Dienstleistungsanbieter und Dienstleistungsnachfrager sind dabei Programme. In Einzelfällen kann ein solches Programm eine grafische Schnittstelle zu einem menschlichen Benutzer sein, doch im Regelfall werden im System Dienstleistungen ohne menschliches Zutun erbracht und genutzt. Rechnergestützte (Informations-)Dienstleistungen sollen auf einheitliche Art angeboten, nachgefragt und genutzt werden können. Damit dieses Zusammenspiel automatisch erfolgen kann, wird es unter anderem notwendig sein, wie bei der Übertragung von HTML-Seiten Standards und Konventionen festzulegen und Werkzeuge zur Verwendung des Standards zu entwickeln. Der erwartete Nutzen ist mehrfach:

- Durch die Einheitlichkeit der Dienstleistungen reduziert sich der Einarbeitungsaufwand von System- und Programmentwicklern.
- Reduktion des Implementierungsaufwands bei der Integration und bei der Benutzung von Dienstleistungen durch Werkzeuge und allgemein verwendbaren Code.
- Inhärentes, aber verstecktes Verteilungskonzept. Dienstleistungen sollen im System verteilt werden können, aber für einen Auftraggeber soll es irrelevant sein, auf welchem Rechner die Dienstleistung erbracht wird.
- Durch die Einheitlichkeit der Dienstleistungen kann prinzipiell jeder Teilnehmer im System mit jedem anderen in Kontakt treten und Dienstleistungen in Auftrag geben. Dadurch wird es möglich, dass intelligente Systeme zur Laufzeit neu hinzukommende Dienstleistungen in Anspruch nehmen und so flexibel auf neue Situationen reagieren.

Einschränkend muss jedoch erwähnt werden, dass durch eine Umsetzung des Dienstleistungskonzept zwar eine technische, syntaktische und protokollare Interoperabilität geschaffen wird. Ob die Dienstleistungsergebnisse aber dem entsprechen, was von ihnen erwartet wurde, und ob es überhaupt sinnvoll ist, eine bestimmte Dienstleistung in einem bestimmten Kontext in Anspruch zu nehmen, liegt außerhalb des Rahmens der vorliegenden Arbeit. Eine semantische Kompatibilität wird durch die Umsetzung des Dienstleistungskonzept ebensowenig gewährleistet, wie dies im Rahmen unserer Dienstleistungsgesellschaft gewährleistet ist. Allerdings sind ähnlich wie dort Mechanismen zur Prüfung der semantischen Kompatibilität einführbar.

1.1 Rahmenbedingungen

Die Entwicklung und die Umsetzung des Dienstleistungskonzepts erfolgte in einem konkreten, praktischen Rahmen. Dieser wurde durch folgende Bedingungen gegeben:

- Einsatzfähigkeit bei der Kernreaktor-Fernüberwachung Baden-Württemberg (KFÜ). Dabei handelt es sich um ein integriertes Informationssystem, in dessen Rahmen die Ausbreitung luftgetragener radioaktiver Nuklide bei kerntechnischen Unfällen oder auch natürlichen Ereignissen, ihre Deposition und die Auswirkungen auf Menschen simuliert werden. Diese Simulation ist sehr komplex und stellt hohe Anforderungen. Sie wird in Kapitel 2.1 genauer eingeführt.

- Für die Umsetzung der KFÜ fiel die Wahl auf eine spezielle Makroarchitektur, den sogenannten Logischen Klienten. Er wurde von meinem Kollegen Roland Kopetzky konzipiert und stützt sich stark auf Dienstleistungen [1], ohne diese jedoch konkret zu beschreiben. Der Logische Klient ist von Ideen aus dem Bereich der intelligenten Software-Agenten geprägt und legt allgemeine Abläufe der Kooperation zwischen den Agenten fest. Gleichzeitig sieht er bestimmte Agenten mit allgemeinen Aufgaben vor. Er wird in Kapitel 2.3 näher beschrieben. Dienstleistungskonzept und Logischer Klient mussten zusammenpassen. Ihre Entwicklungen haben sich in der Praxis immer wieder gegenseitig beeinflusst und gefördert.

Die enge Verbindung zwischen Logischem Klienten und den Konzepten der Agentenorientierung einerseits und dem Dienstleistungskonzept andererseits hat auch Konsequenzen auf die Namensgebung gehabt: Die Umsetzung des Dienstleistungskonzepts erfolgte als Softwarebibliothek unter dem Namen Service Agent Layer (SAL), da sie Dienste (Services)[1] mit der Fähigkeit zur Kommunikation und Kooperation versieht und sie dadurch zu Agenten macht.

Aus den oben dargestellten Rahmenbedingungen resultieren einige Anforderungen an die SAL, die über die Anforderungen an ein rein akademisches System hinausgehen:

- Verläßlichkeit: Jeder Dienst in diesem anspruchsvollen und sicherheitsrelevanten System hängt von der SAL ab. Deshalb muss sich die SAL durch eine sehr große Verläßlichkeit auszeichnen.

- Stabilität der Schnittstellen: Da das Volumen des von der SAL abhängigen Codes ständig wuchs, musste bei allen Schnittstellenänderungen der SAL sorgfältig abgewogen werden, ob sie tatsächlich das bringen, was von ihnen erwartet wird und was sie die Entwickler der Dienste kosten würde. Gleichzeitig musste der Entwurf der SAL so früh und vorausschauend wie möglich erfolgen.

- Performanz und Ressourcenverbrauch: Da die SAL in jedem der 25 Dienste der KFÜ vorkommt, die alle auf einem Rechner laufen sollen, muss der Ressourcenverbrauch der SAL im Rahmen bleiben. Gleichzeitig stellt die KFÜ relativ harte Anforderungen an die Rechenzeit, weshalb auch die Performanz der SAL zu einer wichtigen Größe wurde.

- Praktische Anforderungen: Die SAL soll den Dienstentwicklern möglichst weit entgegen kommen, um den Gesamterstellungsaufwand des Systems zu reduzieren. Dies hat an einigen Stellen erheblichen Einfluss auf die SAL gehabt.

Diese Rahmenbedingungen haben einige Konsequenzen nach sich gezogen. So ist es bei intelligenten Software-Agenten zweckmäßig, auf eine regelbasierte Programmiersprache zurückzugreifen. Diese Sprachen werden zur Laufzeit interpretiert, was eine enorme Flexibilität und Leistungsfähigkeit ermöglicht und dazu führt, dass solche Sprachen gerne für Aufgaben der Künstlichen Intelligenz verwendet werden. Die Interpretation von Nachrichten beispielsweise ist durch regelbasierte Sprachen erheblich einfacher. Der Nachteil besteht darin, dass im Vergleich

---

[1] Mit Dienst ist im Kontext dieser Arbeit ein Programm gemeint, das Dienstleistungen erbringen kann. Eine genauere Definition erfolgt in Kapitel 2.2.

zu in Maschinencode übersetzten Sprachen die Programme wesentlich langsamer sind. Innerhalb der gegebenen Rahmenbedingungen war dieser Nachteil zu schwerwiegend.

Die Wahl der Programmiersprache für die SAL fiel letztlich auf C++, welche effizient und mächtig ist. Die Objektorientierung stellt sehr brauchbare Strukturierungskonzepte zur Verfügung. Es gibt viele Software-Bibliotheken in C++, auf die man aufbauen kann. Java hätte ähnliche Vorteile, schied aber auf Grund von Performanzüberlegungen aus.

1.2 Prinzipien und Anforderungen

In diesem Kapitel sollen grundsätzliche Prinzipien und Anforderungen erläutert werden, die beim Entwurf und bei der Implementierung der SAL relevant waren.

Objektorientierung. C++ ist eine Programmiersprache, die die leistungsfähigen Möglichkeiten der Objektorientierung anbietet, aber nicht einfordert. Die SAL soll diese Möglichkeiten ausnutzen. Außerdem sind bei Bedarf fortgeschrittene objektorientierte Programmiertechniken und Entwurfsmuster anzuwenden [2] [3].

Frameworkansatz[2]. Die SAL stellt eine Reihe von Klassen zur Verfügung, die in ihrer Zusammenarbeit aufeinander abgestimmt sind. Ein Teil dieser Klassen ist konkret und kann direkt verwendet werden. Ein anderer Teil ist abstrakt und muss für den jeweiligen Dienst in einer spezialisierten Form programmiert werden. Zu diesen Klassen gehört beispielsweise die Job-Klasse. Für jede Dienstleistung eines Agenten muss eine solche Klasse programmiert werden, die dafür zuständig ist, die spezifische Dienstleistung auch tatsächlich zu erbringen. Dazu muss sie eine allgemeine, von der SAL vorgegebene und benutzte Schnittstelle implementieren. Somit ist es mit den Klassen der SAL alleine nicht möglich, einen funktionsfähigen Dienst aufzubauen; der Aufbau eines solchen Dienstes ist aber durch Spezialisierung von manchen SAL-Klassen sehr leicht möglich.

Plattformunabhängigkeit. Da die SAL in heterogenen Systemen eingesetzt werden können soll, muss sie selbst auf verschiedenen Plattformen verfügbar sein. Um ein aufwendiges Vorhalten verschiedener Versionen für verschiedene Plattformen zu vermeiden, ist bei der Implementierung auf die Plattformunabhängigkeit des Codes großer Wert zu legen. In der Praxis bedeutet dies die Verwendung von plattformunabhängigen Softwarebibliotheken und eine zurückhaltende Verwendung von C++ - Sprachmitteln, die nicht für alle Compiler zur Verfügung stehen.

---

[2] Zur Klärung des Begriffs *Framework* sei auf Gregory F. Rogers zurückgegriffen, der ihn folgendermaßen definiert [4]:

> A *framework* is a class library that captures patterns of interaction between objects. A framework consists of a suite of concrete and abstract classes, explicitly designed to be used together. Applications are developed from a framework by completion of the implementations of the abstract classes.
>
> A framework can also include additional utilities to aid in the completion of end-user applications. A utility can be a code generator or algorithm, for example.

Unabhängigkeit von der verwendeten Middleware. Unter Middleware versteht man Mechanismen, die die Übertragung von Daten- und Kontrollflüssen in verteilten Umgebungen gewährleisten. Wichtig ist, dass die SAL diese zwar im Sinne der Wiederverwendung benutzt, aber davon so weit wie möglich entkoppelt und unabhängig bleibt. So ist es möglich, die SAL ohne größeren Aufwand an eine andere Middleware anzupassen, sollte dies zum Beispiel auf Grund von Plattformüberlegungen sinnvoll sein.

Asynchronität: Wenn ein Programm eine Dienstleistung in Anspruch nimmt, soll es für die Dauer der Dienstleistung weiterhin erreichbar sein und auch selbst andere Tätigkeiten ausführen können. Gleichzeitig sollte es aber auch möglich sein, solche Aktivitäten blockierend aufzurufen. Wenn das Programm ohne die Ergebnisse einer externen Dienstleistung ohnehin nicht weitermachen kann, dann soll es diese blockierend aufrufen können, ohne sich selbst um die Synchronisation kümmern zu müssen. Die Asynchronität der Nachrichten ist technisch leicht zu realisieren; schwieriger ist es, die notwendigen Synchronisationen an der richtigen Stelle durchzuführen. Programmiertechnisch bedeutete dies, dass die SAL zu wesentlichen Anteilen Nebenläufigkeiten verwenden und unterstützen muss, also threadsicher und multithreaded sein muss.[3]

Erweiterbarkeit und Anpassbarkeit. Es ist zu erwarten, dass die SAL nicht immer in exakt den Zusammenhängen und Agentensystemen verwendet wird, die bei ihrer Entwicklung trotz aller Bemühungen um Allgemeingültigkeit angenommen wurden. Deshalb soll die SAL es einem Entwickler leicht machen, wenn er sie erweitern oder anpassen muss.

Unabhängigkeit vom Anwendungssystem. Die SAL darf weder von der Art der übertragenen Datenobjekte abhängig sein noch von Code, der für das Anwendungssystem spezifisch ist. Programmtechnisch gesehen bedeutet dies, dass ein Dienst bei dienst- oder domänenspezifischen Änderungen zwar neu übersetzt und gebunden werden muss, nicht jedoch die SAL als eigenständige Softwarebibliothek. Durch dieses Prinzip soll sichergestellt werden, dass die SAL auch in anderen Systemen ohne weitere Anpassungen eingesetzt werden kann.

Persistenz. Die SAL muss Anknüpfungspunkte und Mittel bereitstellen, um nach einem Ausfall des Systems oder einzelner Komponenten in Zusammenarbeit mit dem anwendungsspezifischen Code den Verlust von Rechenzeit und Daten minimieren zu können und einen gültigen Systemzustand wiederherstellen zu können.

---

[3] Es ist oft sinnvoll, dass innerhalb eines Software-Prozesses zwei oder mehrere Aktivitäten parallel ausgeführt werden, insbesondere dann, wenn eine Aktivität sehr lange warten muss. Jeder (Teil-)Kontrollfluss wird dabei Thread (englisch für Faden) genannt. Die Threads bekommen nacheinander vom Betriebssystem Rechenzeit zugewiesen. Die Gesamtrechenzeit ist deshalb bei Multi-Threaded-Anwendungen nie geringer als bei Single-Threaded-Anwendungen, aber die Gesamtlaufzeit des Programms kann durch das Ausnützen von Wartezeiten reduziert werden. Das Problem bei Multi-Threaded-Anwendungen ist die Synchronisation der verschiedenen Threads sowie der Schutz von gemeinsam benutzten Ressourcen vor gleichzeitigen oder in sehr kurzen Abständen erfolgenden Zugriffen mit inkonsistenten Zwischenständen. Siehe z.B. [5] für eine umfassende Einführung in Threads.

Systemadministration. Die SAL muss die Systemadministration durch geeignete Maßnahmen und allgemeine Dienstleistungen unterstützen.

1.3 Aufbau dieser Arbeit

Das folgende Kapitel ("Ausgangspunkte") enthält eine Darstellung der Technologien und Konzepte, die für die oben geschilderte Aufgabenstellung relevant sind. Zunächst werden die Aufgaben der Kernreaktor-Fernüberwachung (KFÜ) Baden-Württemberg beschrieben. Im Anschluß daran werden dienste- und agentenbasierte Systeme sowie der Logische Klient als Multi-Agenten-System erläutert. Dann werden die in Zusammenhang mit seiner Anwendung auf die KFÜ notwendigen Anpassungen und zusätzlichen Anforderungen dargestellt. Den Abschluß des Kapitels bildet eine Übersicht über vergleichbare Systeme und andere Agentensysteme. Diese Übersicht wurde unter dem Gesichtspunkt einer verbesserten Möglichkeit des Vergleichs und der Einordnung der vorliegenden Konzepte und Techniken gestaltet.

Das dritte Kapitel ("Die SAL als Dienstleistungsframework") stellt den Hauptabschnitt dieser Arbeit dar. Es enthält eine ausführliche Darstellung des Dienstleistungskonzepts und seiner Umsetzung im Rahmen der SAL sowie eine genaue Beschreibung der Struktur, Prinzipien und Verwendungsaspekte der SAL.

In Kapitel 4 wird der Blickwinkel wieder erweitert und Erfahrungen mit der SAL und dem Dienstleistungskonzept beschrieben und analysiert. Kapitel 5 enthält darauf aufbauend einen Ausblick über mögliche und naheliegende Weiterentwicklungen.

Am Schluß der Arbeit wird eine kurze und prägnante Zusammenfassung gegeben.

2 Ausgangspunkte

In diesem Kapitel wird von verschiedenen Seiten das Umfeld des Dienstleistungskonzepts und der dieses Konzept realisierenden Softwarebibliothek SAL (Service Agent Layer) beleuchtet. Zunächst werden die Aufgaben der Simulation im Rahmen der Kernreaktor-Fernüberwachung (KFÜ) beschrieben. Sie umfassen den Bereich der Emission, Ausbreitung und Auswirkung luftgetragener radioaktiver Schadstoffe. Dieses technisch-physikalisch-biologische System ist sehr komlex [6], [7]. Gleichzeitig werden an die KFÜ hohe Anforderungen gestellt, weil sie im Rahmen des Notfallschutzes verwendbar sein soll. In diesem Umfeld müssen das Dienstleistungskonzept und seine Umsetzung durch die SAL ihre Brauchbarkeit und Mächtigkeit unter Beweis stellen.

Im Anschluß daran werden Prinzipien von dienste- und agentenbasierten Systemen erläutert. Das nächste Unterkapitel beschreibt das von Roland Kopetzky entwickelte Konzept des sogenannten "Logischen Klienten", einem Multi-Agenten-System mit einem fixen Grundinventar von Agenten und deren Aufgaben [1]. Der Logische Klient gibt den Rahmen vor, in dem die SAL eingesetzt wird. Danach werden die für die Anwendung des Logischen Klienten in der KFÜ notwendigen Anpassungen, Erschwernisse und Erleichterungen dargestellt.

Im fünften Unterkapitel wird ein Überblick über einige vergleichbare, konkurrierende, aber auch unterstützende Systeme und Lösungsansätze gegeben und dabei eine Standortbestimmung hinsichtlich relevanter Prinzipien und Technologien vorgenommen.

2.1 Aufgaben der Simulation bei der Kernreaktor-Fernüberwachung (KFÜ)

Das "Gesetz zum vorsorgenden Schutz der Bevölkerung gegen Strahlenbelastung (Strahlenschutzvorsorgegesetz)" gebietet in §1 die Überwachung der Radioaktivität in der Umwelt und die Minimierung der Strahlenexposition der Menschen und der radioaktiven Kontamination der Umwelt. Dies muß "im Falle von Ereignissen mit möglichen nicht unerheblichen radiologischen Auswirkungen unter Beachtung des Standes der Wissenschaft und unter Berücksichtigung aller Umstände durch angemessene Maßnahmen" erfolgen [8].

Daraus ergeben sich mehrere Konsequenzen:

- Notwendigkeit eines Informationssystems, das "zur Berücksichtigung aller Umstände" die relevanten Daten so vorhält, dass sie bei einem radioaktiven Unfall sofort zur Verfügung stehen.
- Festlegung von Berechnungsverfahren nach dem Stand der Wissenschaft, die auf Grund der Datenbasis die Bestimmung "angemessener Maßnahmen" ermöglichen. Dies ist in §6 des Strahlenschutzvorsorgegesetzes angeordnet, und die Strahlenschutzkommission hat die geforderten Berechnungsverfahren im "Leitfaden für den Fachberater Strahlenschutz der Katastrophenschutzleitung bei kerntechnischen Notfällen" [9] festgelegt.
- Umsetzung dieser Berechnungsverfahren in einem computergestützten System mit einem möglichst hohen Automatisierungsgrad, damit auch unter dem enormen Druck und Stress, den ein kerntechnischer Notfall unweigerlich mit sich bringt, die Entscheidungsträger auf

möglichst gute und richtig aufbereitete Daten zurückgreifen und sinnvolle Entscheidungen treffen können.

In Baden-Württemberg existiert seit Mitte der 80er Jahre ein System zur Kernreaktor-Fernüberwachung (KFÜ). Dieses wurde zwar regelmäßig dem Stand der Technik angepasst, jedoch gab es in den vergangenen Jahren insbesondere auf dem Gebiet der Informationstechnik so große Veränderungen, dass Mitte der 90er Jahre anstelle von weiteren, inkrementellen Verbesserungen eine größer angelegte Teilerneuerung angestrebt und in die Wege geleitet wurde. Im Rahmen dieser Teilerneuerung sollte auch die Berechnung der Ausbreitung radioaktiver Emissionen modernisiert werden. Im alten KFÜ wurden angesichts des Rechenzeitaufwandes sehr einfache Ausbreitungsmodelle eingesetzt, die weder die Topographie[4] noch inhomogene Windfelder berücksichtigen konnten. Selbst im Übungsbetrieb mussten Simulationsexperten eingeschaltet werden, welche die Simulation der Ausbreitung und Auswirkung von radioaktiven Emissionen unter den Randbedingungen der Übung durchführen mussten. Ziel der Teilerneuerung ist unter anderem, diese bisher von Menschen erbrachte bzw. zu erbringende Informationsdienstleistung zu automatisieren und so zu einer jederzeit verfügbaren Komponente der KFÜ zu machen.

Mit der Teilerneuerung und Modernisierung dieses Systems hat das Umwelt- und Verkehrsministerium des Landes Baden-Württemberg die Firma debis Systemhaus (dSH) und das Institut für Kernenergetik und Energiesysteme (IKE) der Universität Stuttgart bzw. das Forschungsinstitut Kerntechnik und Energiewandlung (KE e.V.) beauftragt. Debis errichtet dabei ein umfassendes Datenerfassungs- und Verwaltungssystem und die Benutzerschnittstelle. Der KE e.V. implementiert ein Simulationssystem, das die vorliegenden Daten weiterverarbeitet und geeignet aufbereitet, während das IKE die damit verbundenen, forschungsintensiveren Aufgaben übernommen hat.

Die Aufgaben der Simulation im Rahmen der Kernreaktor-Fernüberwachung bestehen darin, auf der Basis genauer Topographiedaten (die für den jeweiligen Kraftwerksstandort, die gewünschte Modellgebietsgröße und die gewünschte räumliche Auflösung aufbereitet werden müssen) und Winddaten den Transport von radioaktiven Nukliden zu simulieren. Zu diesem Zweck wird ein Windfeld berechnet, das sich aus gemessenen oder angenommenen Windwerten und der Topographie ergibt. Der Transport von Nukliden in diesem Windfeld wird berechnet. Auf Grund von Fallout-Koeffizienten und der Kombination von Washout-Koeffizienten und den gemessenen oder angenommenen Niederschlagsdaten wird abgeschätzt, wie lange die Nuklide in der Luft bleiben bzw. wo sie sich am Boden ablagern.

Der nächste Schritt in der Simulation ist die Ermittlung der Strahlenbelastung. Diese wirkt auf drei verschiedene Arten:

- Direkte Strahlung von Nukliden aus der Luft und vom Boden
- Einatmung von radioaktiven Nukliden (Inhalation)

---

[4] Doch gerade die Topographie spielt in Baden-Württemberg eine bedeutende Rolle, da zahlreiche Kernkraftwerke in engen Flusstälern liegen. Die daraus resultierenden lokalen Windfelder beeinflussen die Ausbreitung erheblich.

- Aufnahme radioaktiv verseuchter Nahrung (Ingestion)

Die Belastungen, die sich durch diese sogenannten Expositionspfade ergeben, wirken in verschiedenen Zeiträumen: Direkte Strahlung aus der Luft ist nur relevant, solange die Nuklide noch nicht fortgeblasen sind. Die Wirkung direkter Strahlung vom Boden endet, wenn die Nuklide in nicht aktive Nuklide zerfallen oder entfernt werden. Die Belastungen durch Inhalation und Ingestion hängen wesentlich davon ab, wie lange die Nuklide im Körper bleiben bzw. aktiv bleiben (biologische Halbwertszeit).

Es gibt für die verschiedenen Expositionspfade und Nuklide Parameter, die eine Umrechnung in "effektive Dosen" erlauben. Diese werden für die einzelnen Organe und für verschiedene Zeiträume ermittelt, nämlich für den Berechnungszeitraum, für eine Woche, ein Jahr und für 50 bzw. 70 Jahre (bei der Verwendung von für Kleinkinder gültigen Parametern).

Die Aufbereitung der Ergebnisse erfolgt in farbigen Belastungskarten, aus denen die Entscheidungsträger relativ schnell ablesen können, an welchen Orten welche Maßnahmen zu ergreifen sind.

Die Simulation kann für verschiedene Zwecke eingesetzt werden. Im Alarmfall können auf der Basis aktueller Wetter- und Emissionsdaten Prognosen erstellt werden, die als Entscheidungsgrundlage dienen. Es kann im Nachhinein analysiert werden, ob Störfälle Auswirkungen auf die Bevölkerung gehabt haben. Zuletzt sei der Fall erwähnt, auf den Anwendung in der Praxis hoffentlich für immer beschränkt bleiben wird: Man kann für fiktive Szenarien Simulationen durchführen und Übungen abhalten, Schwachstellen der Strahlenschutzmaßnahmen ermitteln und das Verständnis der Ausbreitung luftgetragener, radioaktiver Schadstoffe, ihrer Deposition und der quantitativen Verteilung der Strahlendosen auf die unterschiedlichen Expositionspfade vertiefen.

Bei der Umsetzung dieses Simulationssystems wird auf bereits vorhandene FORTRAN-Module zurückgegriffen. Während jedoch bisher die Simulation viel Wissen um die einzelnen Module und damit die Bedienung des Systems durch einen Systemexperten erforderte, ist die Aufgabe des zu entwickelnden Simulationssystems, genau dieses Systemwissen vor dem Anwender zu verbergen. Das System soll für Strahlenschutzexperten verwendbar sein, die keine – oder zumindest nur sehr geringe – Kenntnis über den informationstechnischen Aufbau des Systems haben müssen.

Damit liegt das Problem viel weniger in den physikalischen Berechnungen als in der Organisation der Daten und der Abläufe in verschiedenen Varianten. Das System muss außerdem flexibel und erweiterbar sein und den Anwender auf eine möglichst intelligente Art unterstützen.

2.2 Dienste- und agentenbasierte Systeme

Dienste und Agenten sind keine einander ausschließenden Konzepte, aber sie betonen unterschiedliche Aspekte von Systemen mit Komponenten und dem charakteristischem Zusammenspiel zwischen den Komponenten.

Das Konzept der Dienste ist aus dem Alltag motiviert. Ein Dienst ist eine Einheit, die eine (oder mehrere) Dienstleistungen erbringen kann. Wenn man diese Idee auf Computer überträgt, dann

ergeben sich einige Besonderheiten. Hier möchte ich mich Kopetzky [1] anschließen und wiederhole seine Definition von

> Dienste[n] als abgeschlossene Einheiten, deren interne Struktur für das Restsystem ohne Bedeutung ist. Sie bieten im System komplexe Funktionalitäten in Form von Dienstleistungen an und stellen diese über eine syntaktisch und semantisch vollständig definierte Schnittstelle bereit. Dabei ist jeder Dienst selbst für die korrekte Ausführung der von ihm angebotenen Dienstleistung verantwortlich. Dienstleistungen werden von Dienstanbietern erbracht und von Dienstnutzern verwendet. Dabei ist jeder Dienst in der Lage, mit anderen Diensten zur Erbringung einer komplexen Dienstleistung zu kooperieren.

Die Abteilung Wissensverarbeitung und Numerik an der Universität Stuttgart beschäftigt sich schon lange sowohl mit Diensten (insbesonder im Zusammenhang mit dem Umweltinformationssystem Baden-Württemberg [10] - [14]), als auch mit der Simulation komplexer technischer Systeme. Auf dieser Basis hat Martin Schöckle mit seiner Dissertation objektorientierte Ansätze eingebracht [15], während Roland Kopetzky mit seiner Arbeit [1] neue Impulse hinsichtlich einer Gesamtarchitektur gegeben hat. Die vorliegende Arbeit baut mit der Entwicklung und Umsetzung eines Dienstleistungskonzepts harmonisch auf die bisherigen Arbeiten auf und erweitert und verstärkt die vorhandenen Konzepte wesentlich.

Im Vergleich zu Diensten ist es schwieriger, Software-Agenten begrifflich einzukreisen. In den letzten Jahren sind Software-Agenten und agentenbasierte oder Multi-Agenten-Systeme vielbenutzte Schlagwörter geworden, und es gibt eine Vielzahl an Forschungsrichtungen, die sich mit Agenten beschäftigen. Umfassende Definitionsversuche sind so allgemein geworden, dass sie letztlich unbrauchbar sind, und für einzelne Bereiche spezialisierte Definitionen werden nur in Teilbereichen akzeptiert. An dieser Stelle möchte ich mich an eine Definitionstradition halten, die im Bereich der sog. "Intelligenten Agenten" verbreitet ist [16] [17] [18] [19], wonach ein Agent in einer Umgebung angesiedelt ist und mit dieser auf flexible und autonome Weise interagiert. Er verfügt dabei über folgende charakteristische Merkmale:

- Proaktivität: Ein Agent hat Ziele, die er selbständig verfolgt. Dazu kann Opportunismus gehören: Der Agent beobachtet die Umgebung, und wenn die Umstände geeignet und erfolgversprechend sind, dann wird das Programm aktiv.

- Reaktivität: Ein Agent kann auf seine Umwelt reagieren. Ob er diese über Sensoren wahrnimmt, wie zum Beispiel ein mobiler Roboter, oder über eine wohldefinierte Software-Schnittstelle zu anderen Agenten im System, ist dabei nebensächlich.

- Soziales Verhalten: Ein Agent weiß um andere Agenten und kann mit ihnen zusammenarbeiten. In einem offenen System sollte die Anwesenheit eines neuen Agenten dazu führen, dass dieser in die Zusammenarbeit integriert wird.

Ein wichtiges und in fast allen Agentensystemen anzutreffendes Charakteristikum ist die Analogie zu menschlichem Handeln. Bei der Konzeption von Agentensystemen tendiert man im Allgemeinen dazu, sich die Agenten als agierende Menschen vorzustellen und daraus erforderliche Eigenschaften der Agenten, Zusammenarbeits-, Verhandlungs- und Konfliktlösungsstrategien abzuleiten. Die Vorteile dieser Analogie liegen in einer Erleichterung

der Konzeption: Bei Entscheidungen orientiert man sich am menschlichen Handeln. Dies führt in weiterer Konsequenz zu einer eher funktionalen Herangehensweise an das Gesamtsystem und -problem, und die Schnittstellen sind natürlicher und kommen den Bedürfnissen der Endanwender weiter entgegen. Bei einer unbedachten Verwendung der Analogie zum menschlichen Handeln werden Probleme jedoch leicht unterschätzt. Es ist im Alltag relativ einfach, mit anderen Menschen in Geschäftsbeziehungen zu treten und zu entscheiden, wann man Unterstützung von anderen braucht. Derartige Entscheidungsfähigkeiten auf Software-Agenten zu übertragen ist alles andere als trivial, und praktische Agentensysteme lösen entweder einfache, "akademische" Aufgaben, oder sie beschränken sich auf relativ eingeschränkte Aktions- und Interaktionsmuster. Die Analogie greift jedoch dort, wo die "Intelligenz" von Agenten ausreicht, um bestimmte, aus menschlicher Sicht eher wenig anspruchsvolle Tätigkeiten durchzuführen, die gegebenenfalls auch von Menschen erledigt werden. Durch solche Agenten kann der Arbeitsaufwand vom Menschen auf den Computer verlagert werden, und bei Vorhandensein entsprechender Strukturen, Schnittstellen und Benutzeroberflächen können Menschen und Agenten Hand in "Hand" zusammenarbeiten. Dies ist beispielsweise bei kombinierten Workflows realisisert, die typischerweise in großen Organisationen auftreten und bei denen manche Arbeitsschritte von Menschen und andere von Computerprogrammen (Agenten) ausgeführt werden.

Der Unterschied zwischen Diensten und Agenten ist auf der Basis des bisher gesagten verschwimmend. In der Praxis wird der Begriff Dienste eher bei rechenzeitintensiven Aufgaben verwendet, während bei Gebrauch des Begriffs Agenten der Fokus eher in der Zusammenarbeit und der Systemorganisation liegt. Beides zu vereinen war der Anspruch des im folgenden beschriebenen Konzepts des Logischen Klienten. Die Nähe zum Logischen Klienten bedingt eine auch bei dieser Arbeit übliche Terminologie, welche die Begriffe Agenten und Dienste weitgehend synonym verwendet.

2.3 Der Logische Klient

In seiner Dissertation sowie in anderen Beiträgen entwickelt Roland Kopetzky das Konzept des "Logischen Klienten" [1] [20] [21] [22][5]. Dabei handelt es sich um ein Multi-Agenten-System, in dem das Hauptaugenmerk auf der Makroarchitektur des Systems liegt. Alle Aufgaben im System werden von Agenten übernommen oder gekapselt; daneben gibt es keine wie auch immer geartete Komponenten, die für das Funktionieren des Systems von Bedeutung wären. Eine ausführliche Beschreibung des Systems ist an dieser Stelle unabdinglich, da daraus einige bedeutende Aspekte für das Dienstleistungsframework und auch für die Kooperation zwischen den Agenten erwachsen. Zunächst wird relativ kurz beschrieben, welche Vorgaben der Logische Klient für die einzelnen Agenten macht. Dann werden die für das Funktionieren des Gesamtsystems relevanten Agenten beschrieben. Den Abschluß bilden ein paar allgemeine Prinzipien, deren Einhaltung im Logischen Klienten wesentlich ist.

---

[5] Die letzten drei der genannten Beiträge haben Kopetzky und ich gemeinsam gestaltet, wobei sein Beitrag den Logischen Klienten als Gesamtarchitektur umfaßt, während die Aspekte zu Kommunikation, Dienstleistungen und SAL von mir stammen.

2.3.1 Mikroarchitektur eines Agenten

Die Architektur des Logischen Klienten macht nur wenige Vorgaben zur inneren bzw. Mikroarchitektur eines Agenten:

1. Die Kommunikation zwischen Agenten darf nur über die SAL laufen. Damit ist sichergestellt, dass ein gewisser Standard bei der Kommunikation von allen beteiligten Agenten erreicht wird. Ausnahmen dieses Verbots gibt es nur beim Repository-Service und bei der CSCW-Komponente. Der Sinn des Verbots ergibt sich daraus, dass die Systeme klarer, flexibler und erweiterbarer bleiben, wenn es "unter der Decke" keine Abhängigkeiten gibt.

2. Grundsätzlich läuft jeder Agent in einem eigenen Prozess ab. Dabei gibt es drei Arten, wie eine Funktionalität mit der SAL zu einem Agenten verbunden werden kann. Diese sind in Abbildung 1 dargestellt. Links ist angedeutet, dass SAL und die Funktionalität in der gleichen Programmiersprache geschrieben sind und in einem Prozess ablaufen. In der Mitte ist der Fall, dass die spezifische Funktionalität in einer anderen Programmiersprache geschrieben ist, aber innerhalb desselben Prozesses ablaufen. Rechts läuft die Funktionalität in einem anderen Prozess als die SAL ab. Dies ist bei der Integration von bestehender Software die Möglichkeit der Wahl. Die bestehende Software braucht nicht angepasst zu werden, allerdings muss eine Inter-Prozess-Kommunikation zwischen SAL und Service aufgebaut werden. In einfachen Fällen kann dies über Input- und Output-Dateien erfolgen.

3. Aus Gründen der Plattformunabhängigkeit erfolgt die Kommunikation über CORBA als Middleware (vgl. auch Kapitel 2.6.1 zu CORBA). Wie man jedoch aus Abbildung 1 sieht, ist aus der Sicht des Dienstes die Kommunikation durch die SAL gekapselt. Es ist also möglich, anstelle von CORBA andere Mechanismen der Informationsübertragung zu verwenden, ohne dass der agentenspezifische Code dadurch betroffen ist.

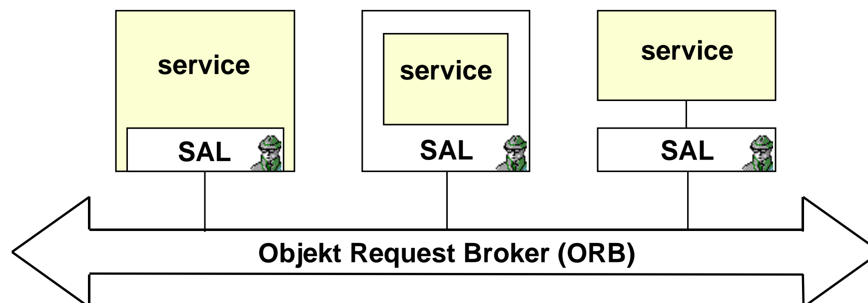


Abbildung 1: Verschiedene Arten der Anbindung eines Dienstes

Aus diesen Vorgaben sieht man deutlich, dass ein Agent im Logischen Klienten auch auf der physikalischen Ebene eine unabhängige Einheit (Prozess) ist, dass die SAL eine zentrale Rolle bei der Kommunikation spielt und einerseits die Dienstespezifika vor dem Rest des Systems versteckt, andererseits die Kommunikationsspezifika vor den Diensten versteckt. Durch diese

Architektur ist der Grundstein dafür gelegt, dass auf Grund der minimalen Kopplung sowohl die einzelnen Agenten unabhängig voneinander eingesetzt und wiederverwendet werden können, als auch eine weitgehende Unabhängigkeit von Implementierungsentscheidungen erreicht wird.

2.3.2 Makroarchitektur des Systems

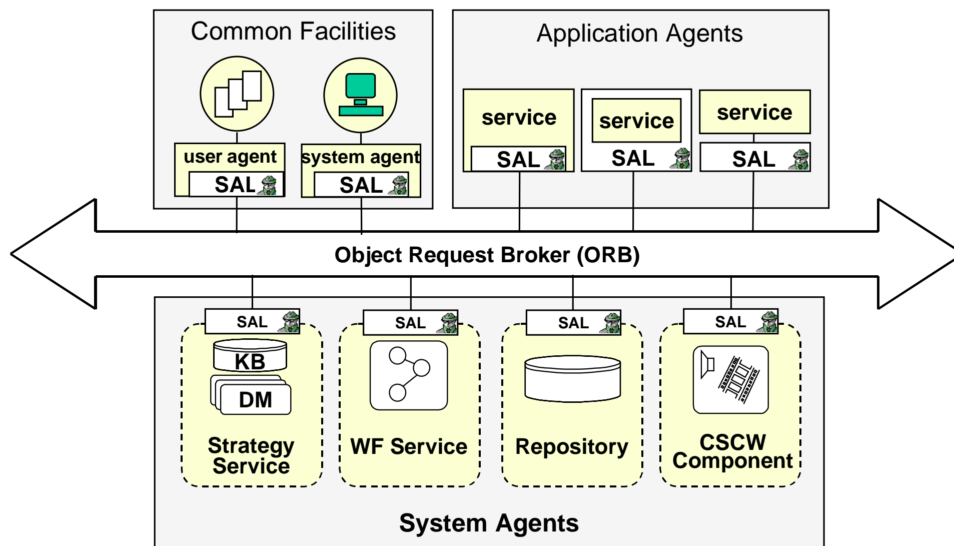


Abbildung 2: Makroarchitektur des Logischen Klienten

Der Logische Klient ist als offenes und verteiltes System konzipiert. Das bedeutet, dass es auch zur Laufzeit möglich sein soll, weitere Agenten hinzuzufügen und zu integrieren. Außerdem ist vorgesehen, mehrere Logische-Klienten-Cluster miteinander zu kombinieren und Ressourcen und Dienste wechselseitig nutzbar zu machen. Abbildung 2 zeigt eine Übersicht von Agenten, die im Logischen Klienten vorkommen. In der Mitte sieht man den Object Request Broker als zentrale Komponente von CORBA, über die in Kapitel 2.6.1 mehr geschrieben wird. Mit seiner Hilfe übernimmt die SAL die Kommunikation zwischen den Agenten. Die Grobeinteilung in Systemagenten, Anwendungsagenten und Common Facilities erfolgt in Anlehnung an die CORBA-Gesamtarchitektur, wo die entsprechenden Gruppen *object services*, *application objects* und *common facilities* genannt werden. Systemagenten stehen allen anderen Agenten zur Verfügung, sie übernehmen zentrale Aufgaben im System. Im Gegensatz dazu übernehmen die Anwendungsagenten anwendungsspezifische Aufgaben. Zu den Common Facilities gehört ein Benutzer-Interface sowie ein Agent, der einen (menschlichen) Systemadministrator unterstützt. Im Folgenden werden die Systemagenten genauer betrachtet.

2.3.3 Strategie-Service

Der Strategie-Service übernimmt im Logischen Klienten einige sehr wichtige und zentrale Auskunftsaufgaben, die mit den Gelben Seiten und mit Vermittlungsdiensten verglichen werden können:

1. Kenntnis aller im System vorhandenen Agenten, der Dienstleistungen, die diese anbieten und der Voraussetzungen, die zu deren Erfüllung gegeben sein müssen.
2. Auskunft über dieses Dienstleistungsangebot und Vermittlung zwischen Dienstnachfrager und Diensterbringer.
3. Erstellen von komplexen Arbeitsplänen (im folgenden Workflows genannt), die die Dienstleistungen der Agenten auf sinnvolle Weise kombinieren.
4. Überprüfen der so gefundenen Workflows hinsichtlich der Verfügbarkeit der betroffenen Agenten und der benötigten Ressourcen.
5. Zusammenarbeit mit Strategie-Services aus anderen Logischen-Klienten-Clustern.

In dynamischen Systemen ergibt sich aus Aufgabe 1, dass jeder Agent, der zum System hinzukommt, sich beim Strategie-Service anmelden und beim Wiederausscheiden abmelden muss. Nur so ist gewährleistet, dass der Strategie-Service eine verläßliche Datenbasis hat. Zur Vermeidung von Redundanzen sowie auf Grund der in Aufgabe 4 geforderten Ressourcenprüfung und –verwaltung ist es darüber hinaus nicht sinnvoll, mehr als einen Strategie-Service in einem Logischen Klienten zu halten. Die Kernkompetenz des Strategie-Service liegt jedoch in seiner Auskunftsfähigkeit: Wenn ein Agent auf der Suche nach einer bestimmten Dienstleistung ist, kann er sich mit einer Beschreibung des Gewünschten an den Strategie-Service wenden. Dieser prüft dann in Erfüllung von Aufgabe 2, ob ihm ein Agent bekannt ist, der diese Dienstleistung erbringen kann. Des weiteren prüft er, ob die dafür notwendigen Ressourcen verfügbar sind. Gibt es jedoch keine passende Dienstleistung, dann versucht der Strategie-Service, das Gewünschte durch eine Kombination verschiedener Dienstleistungen zu erreichen. Dazu erstellt er sogenannte Arbeitspläne oder Workflows (Aufgabe 3) und überprüft deren Verfügbarkeit (Aufgabe 4).

Dabei kann es vorkommen, dass mehr als ein Workflow den Anforderungen genügt. In einem solchen Fall kann der Dienstleistungsnachfragende auf Grund der ebenfalls vom Strategie-Service ermittelten Kenngrößen Kosten, Ausführungszeit und Qualität entscheiden, welchen Workflow er ausführen möchte. Der Dienstleistungsnachfragende wendet sich dann mit dem Workflow an den Workflow-Service, welcher die Ausführung der Teildienstleistungen ansteuert und überwacht.

Dadurch, dass der Dienstleistungsnachfragende beim Strategie-Service auf Basis der momentan im System vorhandenen Agenten Workflows zusammenstellt, können zur Laufzeit hinzukommende Agenten berücksichtigt und genützt werden. Wollte man eine derartige Flexibilität ohne einen zentralen Strategie-Service erreichen, müßte jeder Agent im System die Fähigkeiten zur Bildung von Workflows haben.

Aufgabe 5 wird erst dann relevant, wenn mehrere Logische Klienten zusammengeschlossen werden, und in dem Cluster, in dem der angesprochene Strategie-Service "Platzhirsch" ist, Dienstleistungen zur Erfüllung der Anforderungen des Dienstleistungsnachfragenden fehlen. In einem solchen Fall kann der Strategie-Service die fehlende Dienstleistung bei anderen Strategie-Services suchen und ggf. in den zu erstellenden Workflow einbauen.

Die Beschreibung des gesuchten Workflows beziehungsweise der gesuchten Dienstleistung kann grundsätzlich auf drei Arten erfolgen:

- als Endwertaufgabe unter Angabe der Ergebnisse, die der Workflow liefern soll.
- als Anfangswertaufgabe unter Angabe der Informationen, mit denen der Nachfragende etwas machen möchte, ohne sich zunächst festlegen zu wollen, welche Anforderungen er hinsichtlich der Ergebnisse hat.
- als Randwertaufgabe: Der Nachfragende gibt an, welche Informationen er hat und welche Informationen er daraus generieren möchte.

Die letzte Variante ist insbesondere für die Planung von Ersatzstrategien bei Ausfall eines Teilsystems während der Ausführung eines Workflows notwendig. In diesem Fall soll die Strategiekomponente in der Lage sein, den ausgefallenen Teil des Workflows durch alternative Dienstleistungen zu ersetzen.

Zur Erbringung der verschiedenen Dienstleistungen enthält der Strategie-Service neben der SAL und einigen C++ - Klassen auch ein regelbasiertes System, um die nichttriviale Aufgabe der Kombination von Dienstleistungen zu lösen. Eine Datenbank unterstützt ihn bei der Verwaltung der dienstleistungsbezogenen Daten.

Bisher gibt es keine Implementierung des Strategie-Service in der hier vorgestellten Form. Die für die KFÜ implementierte Teilmenge der Methoden wird später diskutiert. Bei der prototypischen Implementierung des Strategie-Service, die für die Erstellung von Workflows zuständig ist, ist Kurt De Marco in seiner Diplomarbeit [23] auf grundsätzliche Schwierigkeiten gestoßen: Die Kombination von Dienstleistungen anhand ihres Inputs und Outputs führt zu zahlreichen Workflows, deren Sinnhaftigkeit und Anwendbarkeit durch den Computer nur dann feststellbar ist, wenn auch semantische Informationen über die Dienstleistungen zur Verfügung stehen. Dies ist zwar kein unlösbares Problem, erfordert aber viel Zeit und Erfahrung.

2.3.4 Workflows und Workflow-Service

Workflowbeschreibungen werden vom Strategie-Service zur Verfügung gestellt und enthalten die Informationen, in welcher Art und Weise welche Agenten zusammenarbeiten müssen, um im Team Aufgaben zu lösen. Die Aufgabe des Workflow-Service besteht darin, die Abarbeitung eines Workflows zu koordinieren. Zum besseren Verständnis wird in Abbildung 3 ein Beispielworkflow skizziert.

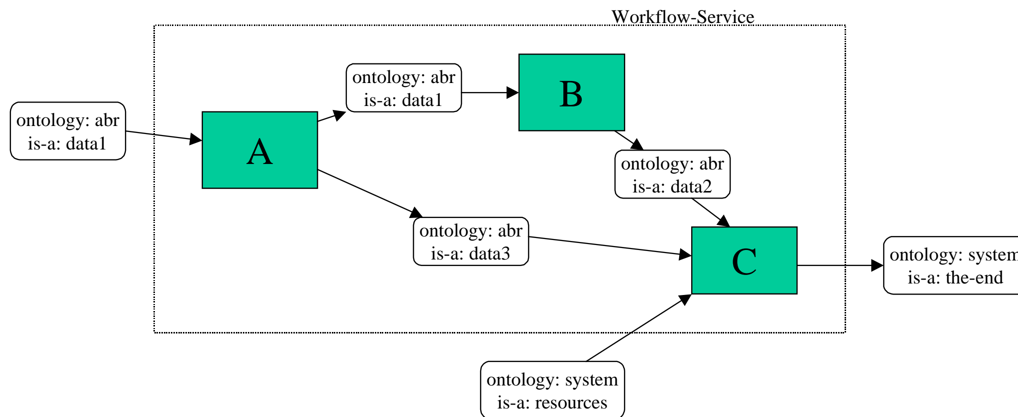
Ein Workflow besteht aus *Aktivitäten*, die in einer bestimmten Reihenfolge durchzuführen sind. Zur Festlegung dieser Reihenfolge werden *Abhängigkeiten* zwischen Aktivitäten angegeben. Für diese Abhängigkeiten ist der Begriff *Transition* gebräuchlich. Wenn also – wie im Beispielworkflow – Aktivität B von Aktivität A abhängig ist, dann muss A abgeschlossen sein, bevor B gestartet wird, und die entsprechende Transition geht von A nach B. Solche Transitionen sind in

der Regel an die Existenz von Datenobjekten geknüpft: A erzeugt ein Datenobjekt (in diesem Fall der Klasse "data1" aus der Ontologie[6] "abr"), welches von B benötigt wird.

Dadurch, dass von einer Aktivität mehrere Transitionen abgehen können, ist eine Parallelität der Ausführung der nachfolgenden Aktivitäten möglich, es sei denn, zwischen den nachgeordneten Aktivitäten besteht eine weitere Abhängigkeit, wie die von C zu B.

Außerdem kann der gesamte Workflow selbst als Aktivität gesehen werden – er benötigt bestimmte Input-Datenobjekte (hier ein "data1"-Objekt aus der Ontologie "abr" und ein "resources"-Objekt aus der Ontologie "system"), und er erzeugt bestimmte Output-Datenobjekte ("the-end" aus der Ontologie "system"). Natürlich handelt es sich bei diesen Klassen um fiktive Klassen; der Beispielworkflow würde in der Praxis kaum einen Nutzen nach sich ziehen.

Eine Aktivität besteht in der Inanspruchnahme einer Dienstleistung, die ein Agent erbringt. Aus diesen Gründen muss in der Workflow-Beschreibung festgehalten werden, welcher Agent dies ist und auf welche Dienstleistung aus dem Angebot des Agenten man sich bezieht.



Legende:

Abbildung 3: Ein einfacher Beispiel-Workflow

[6] Ontologien werden in Kapitel 2.3.9 näher eingeführt. Hier genügt die Vorstellung einer Ontologie als eine festgelegte, formal beschriebene Sammlung von Klassen. Wer die Ontologie verwendet, bekennt sich dazu, die darin enthaltenen Klassen auf genau die Art und Weise zu verwenden, wie sie in der Ontologie vorgegeben ist. So ist ein einheitliches Verständnis – quasi ein Vokabular – definiert, auf dessen Basis man sich austauschen kann und welches das Risiko von Mißverständnissen minimiert.

Ein Workflow kann Schleifen enthalten. In einem solchen Fall sind die an der Schleife beteiligten Aktivitäten in jedem Schleifendurchlauf erneut anzustoßen. Schleifen sind oft vorkommende Konstrukte, die sowohl die Anzahl der möglichen Varianten als auch die Komplexität der Implementierung deutlich erhöhen.

Daraus ergeben sich folgende Aufgaben für den Workflow-Service:

- Einlesen von Workflow-Beschreibungen und Aufbau von zu seiner Abarbeitung notwendiger interner Strukturen.
- Einordnen der Input-Datenobjekte in den Workflow. Der Workflow-Service muss dazu die Art der Input-Datenobjekte auswerten und feststellen, für welche Aktivitäten sie als Input benötigt werden.
- Feststellen, für welche Aktivitäten alle Vorbedingungen erfüllt sind.
- Anstoßen dieser Aktivitäten und Senden der dazu notwendigen Input-Datenobjekte.
- Entgegennahme der Ergebnisse von Aktivitäten und deren Einordnung in den Workflow.
- Feststellen, welche Datenobjekte als Ergebnisse des Workflows dem Auftraggeber zurückgegeben werden sollen.
- Feststellen, wann der Workflow ordnungsgemäß beendet oder aber gescheitert ist.
- Protokollierung der Abläufe zur Dokumentation der Ereignisse sowie zum Lokalisieren eventueller Fehler.
- Umsetzung von Schleifen.
- Persistenz: Der Workflow-Service ist ein Dienst von zentraler Bedeutung. Bei Ausfall des Systems muss er in der Lage sein, bisherige Ergebnisse des Workflows wieder nutzbar zu machen und den Workflow dort fortzusetzen, wo er unterbrochen wurde.
- Unterstützung des Anhaltens, Wiederaufnehmens und Abbrechens von Workflows und von dabei gestarteten Aktivitäten.
- Fehlerbehandlung beim Scheitern einer Aktivität.
- Detaillierte Auskunftserteilung über den Zustand eines Workflows.
- Gleichzeitige Abarbeitung mehrerer Workflows

Dadurch, dass der Workflow-Service die Kontaktierung der beteiligten Agenten übernimmt, muss der Auftraggeber eines Workflows nicht wissen, wer letztlich die Dienstleistungen erbringt. In dieser Unabhängigkeit vom Wissen um externe Agenten und Dienstleistungen liegt eine der Stärken des Logischen Klienten. Für den Auftraggeber eines Workflows bleibt nur die Notwendigkeit, den Workflow-Service und einen geeigneten Workflow zu kennen, dessen Input-Datenobjekte zur Verfügung zu stellen und die Output-Datenobjekten weiterzuverarbeiten, um die Leistungen anderer Agenten nutzbar zu machen. Dadurch ist zumindest theoretisch eine hohe Unabhängigkeit der Agenten voneinander gegeben.

Für die Beschreibung der Workflows wurde die Workflow Process Definition Language (WPDL) herangezogen, die von der Workflow Management Coalition entwickelt wurde und als einheitliches Austauschformat für Workflow-Beschreibungen verschiedener Systeme dienen soll [24]. Ungeachtet aller praktischer Probleme beim tatsächlichen Austausch von Workflow-Beschreibungen und der Wiederverwendung der zugrundeliegenden Abläufe in anderen Systemen und Umgebungen stellt die WPDL eine Menge an Sprachmitteln zur Verfügung, die eigens für die Beschreibung von Workflows entwickelt wurde und sich in diesem Zusammenhang auch bewährt hat.

Für den Workflow-Service existiert eine Implementierung im Rahmen der KFÜ, die bis auf zwei kleine Bereiche vom Anwendungsfall KFÜ unabhängig ist[7]. Diese Implementierung setzt alle wesentlichen Aspekte um.

2.3.5 Repository-Service und die Trennung von Daten und Metadaten

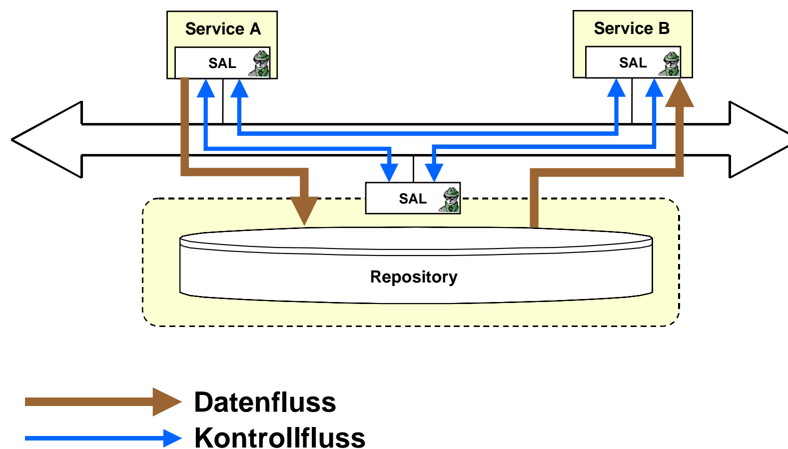


Abbildung 4: Daten- und Kontrollfluss bei Verwendung des Repositories

In vielen Systemen muss mit großen Datenmengen gearbeitet werden. Dies trifft insbesondere auch auf den technisch-wissenschaftlichen Bereich zu. Typischerweise werden diese Daten in Form von Dateien ausgetauscht. Der Austausch von Dateien in einer heterogenen Umgebung ist jedoch nicht ganz einfach. Zur Bewältigung großer Datenmengen hat sich das Prinzip der Trennung von Daten und Metadaten bewährt: Oftmals ist es unnötig, die Daten selbst zu sehen – es genügt, Daten *über* die Daten (sog. Metadaten) zu haben. Dies ist analog zu Paket- oder Lieferscheinen, auf denen lediglich Metadaten vermerkt sind: Was enthält die Sendung, in welchem Zusammenhang ist sie geschehen, wer ist der Sender, etc. Für viele Zwecke (zum

---

[7] Es handelt sich hierbei einerseits um die Umsetzung des Schleifenkonzepts als "Zeitschritte" und die beim Workflow-Service angesiedelte Aufgabe der Generierung dieser Zeitschritte. Ferner wurde für die Protokollierung auf die KFÜ-spezfischen Protokollierungsklassen zurückgeriffen. Beide Bereiche sind jedoch klar gekapselt, so dass eine Anpassung an andere Anwendungsgebiete einfach ist.

Beispiel Buchhaltung und Verwaltung) ist ein Paket- oder Lieferschein vollkommen ausreichend, lediglich der Lieferant und der Belieferte brauchen die Güter selbst in die Hand zu nehmen. Dadurch ergibt sich die Möglichkeit, den Datenfluss stark zu beschränken.

In diesem Zusammenhang übernimmt der Repository-Service wichtige Aufgaben. Ein typisches Verwendungsbeispiel ist in Abbildung 4 skizziert: Service A erzeugt eine Datei und stellt diese in das Repository. Die Datei selbst wird nicht über die SAL übertragen, sondern auf direktem Weg über das Betriebssystem. In der Abbildung ist dieser Datenfluss durch die dicken Pfeile dargestellt. Über die SAL fließt der Kontrollfluss: Service A sagt dem Repository-Service, welche Dateien in das Repository gestellt werden sollen, und dieser gibt als Quittung ein Repository-Objekt, das unter anderem eine eindeutige Repository-Identifikationsnummer enthält. Dieses Repository-Objekt läßt Service A über die SAL Service B zukommen. Dieser kann, wenn er die Datei tatsächlich benötigt, damit beim Repository-Service die Datei erhalten. Dieser Datenfluss erfolgt wieder direkt über das Betriebssystem.

Damit lassen sich die Aufgaben des Repository-Service folgendermaßen zusammenfassen:

- Einchecken von Dateien unter Angabe allgemeiner Metadaten.
- Auschecken von Dateien unter Angabe des Repository-Objekts. Mit dem Auschecken kann das Löschen der Datei beim Repository-Service veranlasst werden.
- Verwaltung der Dateien: Löschen von Dateien nach Ablauf der Aufbewahrungsfrist, Suche nach Dateien mit bestimmten Dateiattributen, ...
- Kapselung der Heterogenität. Wenn ein Agent auf Plattform A eine Datei beim Repository eincheckt und einem anderen Agenten auf Plattform B die Repository-Identifikationsnummer zukommen lässt, dann kann dieser die Datei auschecken und muss sich um Plattform-, Transfer- und Konvertierungsfragen nicht mehr kümmern.
- Kapselung der physikalischen Speicherbedingungen. Es liegt in der Hand des Repository-Service, für die Speicherung von Dateien geeignete Speichermedien und Speichersysteme zu verwenden, ohne dass die Benutzer des Repository-Service darüber Bescheid wissen oder darauf Rücksicht nehmen müssen.
- Transaktionssicherheit: Wenn das Einchecken von Dateien mißlingt, dann muss der Repository-Service in der Lage sein, wieder zu einem wohldefinierten Ausgangszustand zurückzukehren.

Wichtig ist an dieser Stelle, daß die Metadaten des Repository-Objekts die Beschreibung der dateirelevanten Metainformationen umfassen, nicht aber die datenrelevanten Metainformationen. Das bedeutet, dass die Beschreibung, was in der Datei zu finden ist, über weitere Metadatenobjekte erfolgen muss, die jedoch für den Repository-Service irrelevant sind.

Für den Repository-Service gibt es eine voll funktionsfähige Implementierung für Windows NT.

2.3.6 CSCW-Komponente

CSCW steht für Computer-Supported Cooperative Work. Gemeint sind damit Technologien der Tele-Zusammenarbeit wie zum Beispiel Video-Konferenzen, die über Computernetze abgehalten

werden. Auf Grund des dabei entstehenden massiven Datenstroms ist nicht vorgesehen, dass alle Informationen über die SAL ausgetauscht werden. Vielmehr übernimmt die CSCW-Komponente mit Hilfe der SAL die Koordination und Vermittlung zwischen am System beteiligten Menschen. Motiviert ist der Einsatz von CSCW dadurch, dass der Komplexitätsgrad von verteilten Systemen zunimmt, und dass es trotz verbesserter Dokumentationssysteme immer wieder notwendig ist, Rücksprache mit Experten zu halten. Genau das soll durch die CSCW-Komponente unterstützt werden.

Es existiert momentan keine CSCW-Komponente im Rahmen des Logischen Klienten. Sie wird in der KFÜ nicht benötigt, weil die Benutzerschnittstelle durch ein unabhängiges System der Firma debis Systemhaus realisiert wird.

2.3.7 User-Interface

Das User-Interface dient dazu, einem menschlichen Benutzer Zugang zum System zu verschaffen. Dabei wird es zwangsläufig notwendig sein, auf die jeweiligen Gegebenheiten des Anwendungssystems Rücksicht zu nehmen. Für das User-Interface existiert eine funktionelle Lösung, die auf generische Weise auf der Basis von HTML-Seiten und marktüblichen Browsern Informationen darstellen kann.

2.3.8 Admin-Interface

Das Admin-Interface stellt einige Dienstleistungen zur Verfügung, die es dem Systemadministrator erlauben, die momentane Situation des Systems zu beurteilen und ggf. einzugreifen. Dazu bedient er sich einerseits allgemeinen Funktionalitäten der SAL und des Workflow-Service, z.B. um alle laufenden Dienstleistungen zu sehen, andererseits kann es sinnvoll sein, auch hier auf systemspezifische Agenten zurückzugreifen. Für die KFÜ gibt es eine rudimentäre Version des Admin-Interface, das im Rahmen der Integration weiterentwickelt wird.

2.3.9 Ontologie

Damit die Kommunikation zwischen den Software-Agenten funktioniert, muss sichergestellt werden, dass sie ein einheitliches Verständnis der Informationen haben, die ausgetauscht werden. Anstelle mündlicher Absprachen zwischen den Programmierern oder in Text-Dokumenten festgehaltener Spezifikationen ist es im Software-Bereich angebracht, solche Spezifikationen in formalen Sprachen festzuhalten. Diese können dann von Programmen mehr oder weniger direkt ausgewertet werden. In der Arbeit von Roland Kopetzky findet sich eine ausführliche Diskussion solcher formaler Spezifikationen, die meist *Ontologie* genannt werden [1]. An dieser Stelle wird auf Hintergründe, Geschichte, Probleme und Alternativen bei der Modellierung von Ontologien nur insofern eingegangen, als sie für das Kommunikationsframework und die Analyse, den Vergleich und den Ausblick (vgl. Kap. 4.1) relevant sind.

In der Ontologie wird festgehalten, welche Informationsklassen es gibt, welche Attribute diese haben und welchen Typ die Attribute haben. Damit ist zunächst über die Semantik dieser Informationsklassen noch nichts ausgesagt – die Namen der Klassen und ihrer Attribute sind zwar für den Menschen sprechend, aber für den Computer irrelevant. Dennoch wird durch die Namensgebung und durch die Dokumentation eine falsche Verwendung schon weitgehend

ausgeschlossen. Zusätzlich können einfache und komplexe Nebenbedingungen formuliert werden, um zum Beispiel Gültigkeitsbereiche für Attribute einzuschränken. Somit bildet die Ontologie ein Nachschlagewerk und einen Bezugspunkt für die weiteren Spezifikationen und Entwürfe.

Für die Modellierung der Ontologie wurde die Unified Modelling Language (UML) verwendet, die von James Rumbaugh, Grady Booch und Ivar Jacobson aus deren individuellen Vorarbeiten entwickelt und inzwischen von der Object Management Group (OMG) als Standard übernommen wurde [24] [26]. Die OMG ist eine firmenübergreifende Organisation mit dem Ziel, Standards zur Objektorientierung zu etablieren. Die UML ist ein sehr erfolgreicher Standard, für den es inzwischen eine breite Palette an unterstützenden Werkzeugen und eine Unmenge an Literatur gibt [27]. Als Werkzeug kam das kommerzielle Programm Rational Rose zum Einsatz, mit dessen Hilfe man Klassen erzeugen, graphisch repräsentieren und bearbeiten kann. Auf diese Weise wird ein Klassenmodell – die Ontologie – entwickelt. Außerdem wird im Rahmen der UML auch eine Sprache zur Beschreibung von Nebenbedingungen spezifiziert, die Object Constraint Language (OCL) genannt wird und für die Ontologie verwendet wurde.

Aus diesem Klassenmodell können dann weitere Dokumente erzeugt werden. Rational Rose ist mit einer Skriptsprache ausgestattet, die es erlaubt, durch das Modell zu navigieren und auf alle darin enthaltenen Informationen in strukturierter Weise zuzugreifen. Mit Hilfe solcher Skripts wurden z.B. C++ - Klassendefinitionen erzeugt, auf die der Programmierer zugreifen muss, wenn sein Agent mit anderen Agenten Informationen austauscht. Weitere Skripts erzeugen eine Online-Dokumentation, durch die man mit Hilfe eines Browsers navigieren kann, überprüfen die Konsistenz des Modells, erzeugen Klassendefinitionen in anderen Programmiersprachen, etc.

Die explizite Festlegung dessen, was kommuniziert wird, und die Vielzahl an Weiterverarbeitungsmöglichkeiten sind die großen Stärken von Ontologien, auch wenn deren Erstellung keine leichte Aufgabe ist und unter anderem ein hohes Abstraktionsvermögen erfordert. In Kapitel 3.4 werden die engen Zusammenhänge zwischen der Kommunikation und der Ontologie genauer zur Sprache kommen.

Darüber hinaus wird unter Ontologie oft auch ein Modell verstanden, das nicht nur die genannten strukturellen Informationen und Nebenbedingungen enthält, sondern auch Abläufe und Regeln definiert.

2.4 Anwendung des Logischen Klienten in der Ausbreitungsrechnung der KFÜ

Der Logische Klient wurde in der Ausbreitungsrechnung angewandt, welche ein Teilsystem der KFÜ ist. Dabei ergeben sich sowohl Vereinfachungen als auch Erschwernisse. Vereinfachungen entstehen dadurch, dass die Ausbreitungsrechnung ein abgeschlossenes[8] System ist und alle Agenten und Dienstleistungen im Voraus bekannt sind. Erschwerend ist, dass das System eine Reihe von zusätzlichen Anforderungen hat, die nicht nur einzelne Agenten betreffen, sondern

---

[8] Die Abgeschlossenheit bedeutet hier nicht, dass es keine Schnittstellen nach außen gäbe, sondern dass im Gegensatz zu offenen Systemen zur Laufzeit keine Änderungen der Systemstruktur auftreten.

vom System als Ganzem zu lösen sind. Bevor beides näher ausgeführt wird, ist es sinnvoll, zunächst die Systemarchitektur im Groben zu besprechen, die für die Ausbreitungsrechnung ausgewählt wurde (vgl. Abbildung 5).

Als zentrales Element ist in der Grafik die Kommunikation in Form eines Doppelpfeils abgebildet. Auf eine Ausweisung der SAL in jedem einzelnen Agenten wurde aus Platzgründen verzichtet.

Oberhalb des Pfeils finden sich die Systemdienste. Neben Strategie-, Workflow- und Repository-Service gibt es den Resource-Service, der für die Verteilung der Systemressourcen zuständig ist. Dies ist in der KFÜ sehr wichtig, da im Falle einer Alarmrechnung die Ressourcen richtig verteilt werden müssen. Es darf nicht vorkommen, dass eine Alarmrechnung Ressourcenprobleme bekommt, weil weniger wichtige Rechnungen laufen.

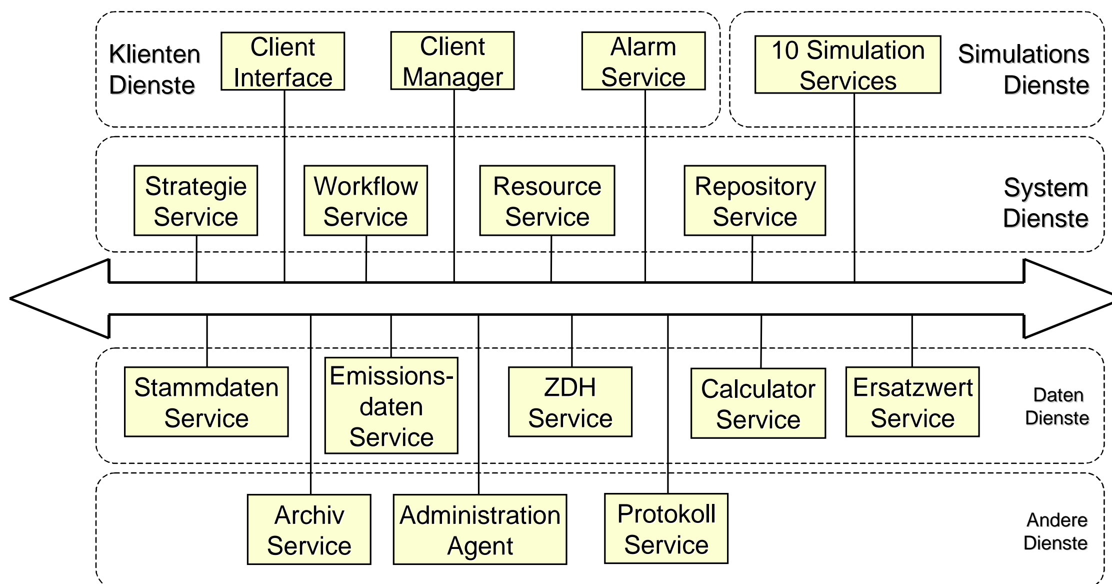


Abbildung 5: Übersicht der verwendeten Dienste in der Ausbreitungsrechnung

Links oben ist die Gruppe der Klientendienste zu sehen. Über eine externe Benutzeroberfläche wird für jeden angemeldeten Benutzer ein Client Interface gestartet. Die Verwaltung der verschiedenen Benutzer und ihrer Kommunikations- und Simulationssessions liegt beim Client Manager. Dieser ist für das System immer verfügbar, z.B. um Anfragen nach Input-Daten an den Benutzer abzusetzen. Er vertritt also alle Benutzer für das System und stellt für alle Client Interfaces einen Single Point of Entry zum System dar. Ein spezialisierter Klientendienst ist der

Alarm-Service. Er empfängt von außerhalb Aufträge zur Alarmauslösung und –beendigung und veranlasst über den Client Manager das Starten und Beenden von Alarmrechnungen.

Rechts oben ist die Gruppe der Simulationsdienste zu sehen. Diese kapseln Fortran-Module, welche die rechenzeitintensiven Aufgaben bewältigen und die dabei benötigten und anfallenden Dateien verwalten und in das Repository stellen. Es gibt Simulationsdienste für die Aufbereitung der Topographie, die Berechnung von Windfeldern, die Transportrechnung von Partikeln in diesen Windfeldern, die Berechnung des radioaktiven Fallouts und des Washouts und die Bestimmung der radioaktiven Belastungen über die Expositionspfade (direkte Strahlung, Atmung, Nahrungsaufnahme) auf die verschiedenen Bevölkerungsgruppen (Erwachsene, Kinder) und Organe, hochgerechnet auf verschiedene Zeiträume.

Die Gruppe der Datendienste ist für die Beschaffung von Daten zuständig. Der Stammdatenservice verteilt auf Anfrage die sogenannten Stammdaten, welche sich nicht ändern und konfigurierbar sind. Dazu gehören beispielsweise die Koordinaten der Kraftwerke und die Abmessungen der verschiedenen Modellgebietsgrößen. Der Emissionsdatenservice implementiert die Bestimmungen des Leitfadens für Strahlenschutzexperten [9]: Unter bestimmten Annahmen hinsichtlich des aufgetretenen Unfalls bestimmt er für die verschiedenen Zeitintervalle die radioaktiven Emissionen. Diese Daten können mit gemessenen Emissionsdaten skaliert werden. Der ZDH-Service stellt den Zugang zur externen Zentralen Datenhaltung her, in welche in der Regel alle 10 Minuten Wetter- und Kraftwerksdaten eingespeist werden. Der Calculator-Service übernimmt weiterführende Berechnungen, zum Beispiel hinsichtlich der Stabilität der Atmosphäre, während der Ersatzwert-Service bei Datenausfall Ersatzwerte zur Verfügung stellt. Momentan sind dies einfache Default-Werte, aber aufwendigere Ersatzwertstrategien sind leicht denkbar.

Ganz unten im Bild sind diejenigen Agenten, die zu keiner der bisherigen Gruppen passen: Der Archiv-Service übernimmt die Speicherung und das Abrufen von Daten aus den Simulationsberechnungen. Dabei greift er auf die externe Zentrale Datenhaltung zurück, auf der die Daten gespeichert und gesichert werden können. Der Administrationsagent erlaubt es einem Systemadministrator, sich ein Bild über die laufenden Aktivitäten zu machen und ggf. einzugreifen, einzelne Agenten oder das Gesamtsystem neu zu starten, Rechnungen abzubrechen, usw. Der Protokoll-Service ist hingegen dafür zuständig, von allen Agenten in regelmäßigen Abständen die Protokolldateien abzuholen und zu speichern.

In der Übersicht dieser verschiedenen Dienste und Agenten sieht man deren Vielfalt: Es gibt Agenten, die externe Komponenten kapseln und in das System bringen (ZDH-Service, Alarm-Service, Archiv-Service, Client Interface), während andere existierende Fortran-Programme anbinden, indem sie diese als eigene Prozesse starten (Simulationsdienste und Emissionsdatenservice). Es gibt sehr einfache Dienste (Ersatzwert-Service, Calculator-Service, Stammdatenservice) und solche, die eine Vielzahl an Aufgaben übernehmen und koordinieren (Client Manager).

Die Vereinfachungen auf Grund der Abgeschlossenheit des Systems sind:

- Alle Workflows können im Voraus erstellt werden. Dadurch kann der Strategie-Service wesentlich vereinfacht werden. Die Workflows (es sind 47) wurden von Hand erstellt und werden vom Strategie-Service auf Anfrage herausgegeben.

- Wenn ein Workflow in Auftrag gegeben wird, ist bereits zur Implementierung dieses Auftrags bekannt, wie dieser Workflow vom Strategie-Service anzufordern ist, welche Eingabeobjekte er benötigt und welche Ausgabeobjekte verwendet werden können. Damit können die Auftraggeber von Workflows wesentlich vereinfacht werden.

Erschwerend kommt jedoch hinzu, dass das System hochgradig dezentral aufgebaut ist, es aber andererseits auch einige Aufgaben gibt, die leichter zentral zu lösen wären und für die im gewählten System die Zusammenarbeit vieler oder auch aller Agenten erfordert:

- Ein zentrales Ressourcenmanagement ist notwendig. Dazu müssen einerseits alle rechenzeitintensiven Vorgänge beim Resource-Service angemeldet werden. Ggf. verweigert dieser die Zustimmung zu den geplanten Vorgängen. Andererseits überwacht der Client Manager, dass zu einer bestimmten Zeit nicht mehr als eine festgelegte Höchstzahl an Rechnungen laufen. Außerdem muss der Client Manager im Alarmfall alle normalen Rechnungen anhalten oder – falls notwendig – beenden.
- Die Protokollierung verläuft dezentral an allen Stellen im System, die Protokolle müssen jedoch gesammelt werden und auf die Simulationsberechnung aufgeteilt werden. Dies ist eine zentrale Aufgabe, zu der alle Agenten im System beitragen müssen. Softwaretechnisch ist dies durch die Verwendung einer Protokollierungs-Bibliothek in allen Agenten gelöst.
- Der Administrationsagent muss mehrere Stellen überwachen, um seinen Aufgaben nachzukommen. Dies sind in erster Linie der Workflow-Service und der Client Manager, weil sie zentrale Knoten im System sind. Andererseits gibt es auch Aufgaben, für die der Administrationsagent jeden einzelnen Agent ansprechen muss.

Mit dieser Beschreibung des Logischen Klienten und seiner Anwendung in der Ausbreitungsrechnung als Teilsystems der KFÜ sollte ein Grundverständnis der wesentlichen Aspekte des Systems vermittelt werden. Dabei wurden jedoch bloß Aspekte der Ausbreitungsrechnung behandelt. Deren Einbindung in die KFÜ wird im Folgenden überblicksweise dargestellt.

2.5 Einbindung der Ausbreitungsrechnung in die KFÜ

In diesem Kapitel wird das Gesamtsystem der KFÜ, die Einbindung der Ausbreitungsrechnung in dieses System sowie die Bedeutung des im Rahmen dieser Arbeit entwickelten Dienstleistungskonzepts für das Gesamtsystem dargestellt.

Abbildung 6 zeigt die Basisarchitektur der KFÜ. Zentrale Komponente ist die Zentrale Datenhaltung (ZDH), welche Mess-, Prognose- und sonstige Arbeitsdaten hält und verarbeitet. Der Datenbestand der ZDH wird laufend über den Kommunikationsserver (KS) um aktuelle Daten, z.B. Wetter- und Kraftwerksdaten erweitert. Der Application Server (AS) übernimmt weitere Aufgaben, unter anderem die Replikation der ZDH-Daten aus Gründen der Ausfallsicherheit. Auf dem Ausbreitungs-Server (ABR) werden Ausbreitungsrechnungen druchgeführt. Der Client ist die Schnittstelle des Systems zum Anwender. Es handelt sich dabei um Spezialsoftware mit grafischer Benutzeroberfläche, die auf speziellen Rechnern installiert ist. Es gibt mehrere Clienten, die den verschiedenen Verantwortlichen und Entscheidungsträgern zur Verfügung stehen. Mit dem Clienten können unter anderem Ausbreitungsrechnungen

parametriert, gestartet, überwacht und ausgewertet werden. Die Ergebnisse werden dabei – wie in anderen Geografischen Informationssystemen – ebenso wie die Daten der ZDH über Landkarten dargestellt, wobei Kartenausschnitt, Maßstab und Darstellungsform der Daten wählbar sind.

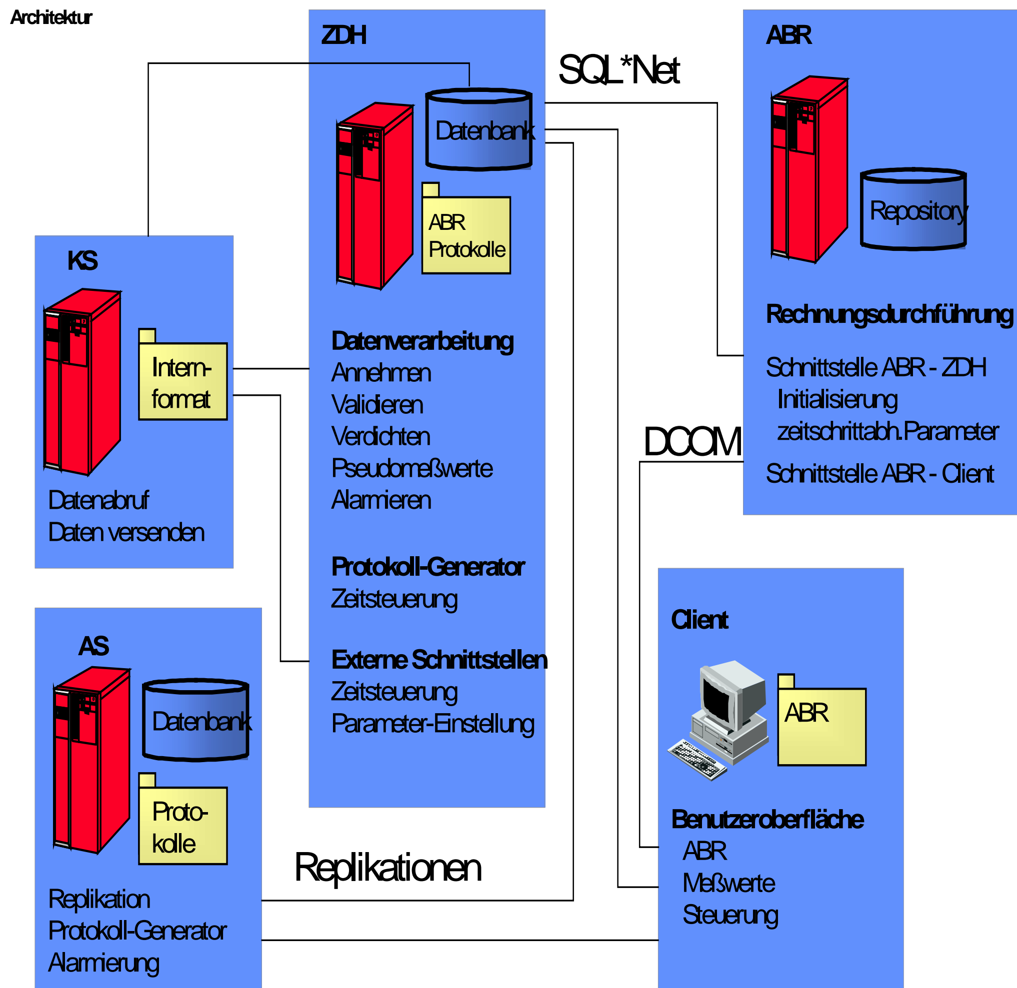


Abbildung 6: Architektur der KFÜ Baden-Württemberg (Quelle debis Systemhaus)

Für den Ausbreitungsserver wurde die weiter oben beschriebene Architektur des Logischen Klienten gewählt. Seine Schnittstellen zu den anderen Komponenten der KFÜ werden in Abbildung 7 dargestellt. Im Folgenden werden diese Schnittstellen anhand von typischen Anwendungsfällen beschrieben.

Wenn der Benutzer vom Client aus eine Verbindung zum Ausbreitungsserver herstellt, wird dort ein Client Interface erzeugt, welches die erforderlichen Daten vom Benutzer erfrägt und damit eine SAL-Dienstleistung beim Client Manager aufruft. Dieser entscheidet aufgrund der Benutzereingaben, für welchen Rechnungstyp ein Workflow vom Strategie-Service angefordert wird und gibt diesen zur Ausführung an den Workflow-Service weiter. Wenn die Ausbreitungsrechnung Daten von der Zentralen Datenhaltung benötigt, werden diese über den ZDH-Service in das System gebracht. Dabei werden alle Aktivitäten innerhalb der

Ausbreitungsrechnung als SAL-Dienstleistungen erbracht. Der ZDH-Service bietet beispielsweise Dienstleistungen an, welche die entsprechenden Datenbankabfragen bei der Zentralen Datenhaltung absetzen und die gewonnenen Daten ontologie-konform so aufbereiten, dass sie innerhalb der Ausbreitungsrechnung von allen nachfolgenden Dienstleistungen richtig interpretiert werden können. Die Ergebnisse der Ausbreitungsrechnung können vom Benutzer abgefragt und in der Benutzeroberfläche visualisiert werden. So kann man sich beispielsweise die Strahlendosis-Isoflächen farbig in die topographische Karte einzeichnen lassen, um zu sehen, in welchen Gebieten mit welchen Strahlenbelastungen für die Bevölkerung zu rechnen ist.

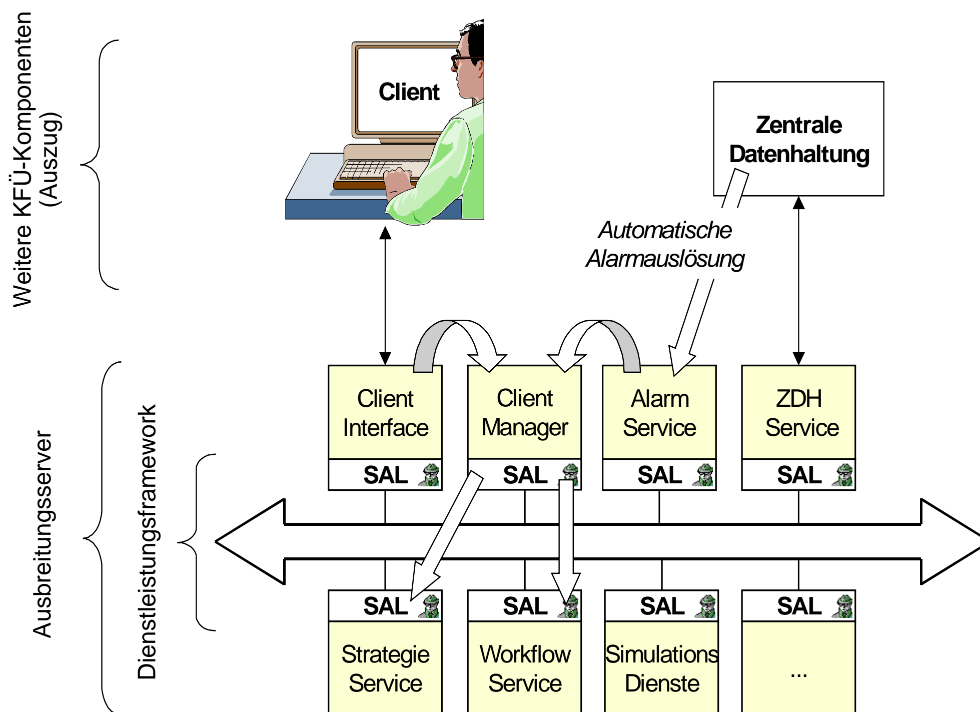


Abbildung 7: Nutzung der Ausbreitungsrechnung in der KFÜ und die Rolle des Dienstleistungsframeworks im ABR-System. Blockpfeile bezeichnen den Aufruf von SAL-Dienstleistungen.

Im Alarmfall gibt es mehrere mögliche Abläufe. Wenn bei der Zentralen Datenhaltung beim Aktualisieren von Emissionsdaten zu hohe Werte festgestellt werden, wird automatisch ein Dienstleistungsauftrag an den Alarm-Service gestellt, welcher dann für das entsprechende Kraftwerk unter bestimmten Annahmen über die Unfallkategorie eine Ausbreitungsrechnung startet. Selbstverständlich werden in einem solchen Fall alle zu diesem Zeitpunkt laufenden „normalen“ Rechnungen gestoppt, um die Rechenleistung voll für die Alarmrechnung nutzen zu können. Gleichzeitig werden die Verantwortlichen im Umwelt- und Verkehrsministerium alarmiert. Diese können beispielsweise die Alarmrechnung stoppen, um eine neue Rechnung mit realistischeren Annahmen über den Unfall zu starten. Wenn ein Störfall vorliegt, bei dem noch keine radioaktiven Nuklide freigesetzt wurden, aber eine Freisetzung unumgänglich ist (z.B. eine Druckentlastung des Reaktors durch Ablassen von bei der Kernspaltung entstandenen radioaktiven Edelgasen), dann können Benutzer eine Prognoserechnung starten, in der im Voraus

simuliert wird, wie sich die radioaktive Wolke ausbreiten würde. Eine Analyse der so gewonnenen, prognostizierten Strahlendosen kann die Auswahl von Schutzmaßnahmen für bestimmte Gebiete erleichtern und deren Wirkung erhöhen. Außerdem ist es möglich, den Zeitpunkt der Freisetzung unter Berücksichtigung von Änderungen der Wetterverhältnisse möglichst gut zu wählen. Dies könnte bei tageszeitlich bedingten Änderungen der Windverhältnisse (Thermik) oder bei prognostizierten Niederschlägen sinnvoll sein, da diese das Ausbreitungs- und Depositionsverhalten entscheidend beeinflussen.

In Abbildung 7 wird nochmals deutlich, dass das im Rahmen dieser Arbeit entstandene Dienstleistungskonzept und damit die SAL für die Ausbreitungsrechnung eine ganz zentrale Rolle spielt. Es ist das verbindende Rückgrat der Ausbreitungsrechnung. Bevor es in Kapitel 3 detailliert beschrieben wird, sollen einige alternative oder verwandte Komponenten- und Agentensysteme dargestellt und mit dem Logischen Klienten verglichen werden. Dadurch soll der Blick für das System geschärft und seine Bewertung und Einordnung erleichtert werden.

2.6 Vergleichbare Systeme

Im folgenden Kapitel werden einige Systeme beschrieben, die mit dem Dienstleistungskonzept bzw. mit dem Logischen Klienten lohnenswert verglichen werden können. Zunächst werden mit CORBA und den Enterprise Java Beans zwei Basistechnologien für Verteilung und Komposition in komplexen Systemen beschrieben. Daran anschließend werden Eigenschaften von Komponentenbasierten Systemen dargestellt, zu denen auch der Logische Klient gezählt werden kann. Ein weiteres, bei Multi-Agentensystemen weit verbreitetes Konzept ist das des Blackboards, welches aufgrund des zentralisierten Informationsbestandes in Kontrast zum Logischen Klienten steht. Abschließend wird ein Standardisierungsversuch von Agenten und Agentensystemen der Foundation of Intelligent Physical Agents (FIPA) dargestellt.

Das Ziel dieser Darstellung liegt einerseits in einer Beschreibung von Konzepten und Technologien, die Einfluß auf den Logischen Klienten und das Dienstleistungskonzept haben und verwendet wurden. Gleichzeitig sollen auch Abgrenzungen zu anderen Konzepten vorgenommen werden. Eine ausführliche Diskussion und Übersicht über weitere Agentensysteme findet man z.B. in [28].

2.6.1 CORBA

Bei der Common Object Request Broker Architecture (CORBA) handelt es sich um einen von der Object Management Group (OMG) herausgegebenen Standard [29] [30]. Die OMG ist ein firmenübergreifendes Konsortium und hat es sich im Bereich der Objektorientierung zur Aufgabe gemacht, Standards zu entwickeln. Der erfolgreichste Standard der OMG ist die Unified Modelling Language (UML), die eine grafische Notation und Austauschformate für objektorientierte Konstrukte definiert.

CORBA ist ein Standard, der die Verteilung von Objekten und entfernte Methodenaufrufe auf solche Objekte normiert. Es gibt CORBA-Implementierungen für fast alle Plattformen und für viele, nicht nur objektorientierte Programmiersprachen. Bei der Verwendung ist es für den Anwendungsentwickler relativ einfach, Objekte auch über Rechnergrenzen hinweg zu verteilen und kooperieren zu lassen. Verschiedene CORBA-Implementierungen sind interoperabel, und es

ist auch – zumindest theoretisch – leicht möglich, eine CORBA-Implementierung gegen eine andere auszutauschen.

Auf der Basis solcher verteilter Objekte definiert CORBA darüber hinaus eine Reihe von weiteren, in diesem Zusammenhang nützlichen oder auch notwendigen Diensten, die zum Großteil ebenfalls bereits als einsetzbare Produkte verfügbar sind. Allerdings zeigt sich bei manchen dieser Dienste, dass eine vollständige Umsetzung im Rahmen eines vollständig verteilten Objektsystems schwierig ist. So ist beispielsweise der SecurityService noch immer nicht vollständig implementiert [31].

Manche CORBAservices haben einen ähnlichen Aufgabenbereich wie zentrale Dienste im Logischen Klienten, doch berücksichtigt CORBA die semantische Ebene nicht. Dies wird bei der näheren Betrachtung des CORBA Trading Service deutlich, dessen Aufgabe die Vermittlung von Objekten mit spezieller Schnittstelle ist. Dabei werden Methodenaufrufe über den Namen der Schnittstellenfunktion und die Anzahl, Reihenfolge und Typen der Übergabeparameter beschrieben. Die Semantik einer Dienstleistung kann zwar als Freitext angegeben werden, doch die Interpretation dieser Beschreibung der Semantik ist offen. Das Trading-Konzept kann also nur dort funktionieren, wo bekannt ist, welche Semantik sich hinter welcher Schnittstelle verbirgt bzw. wie diese beschrieben ist. Dies trägt nur in abgeschlossenen Systemen. Deshalb kann es beim Trading eigentlich nur darum gehen, für ein bekanntes Problem und die entsprechende, ebenfalls bekannte Schnittstelle aus einem Pool von passenden Objekten eines auszuwählen. Als Veranschaulichung sei hier das Problem des Druckens in Netzwerken herangezogen. Unter der Annahme, dass die Schnittstelle bekannt ist, könnten sich Drucker beim Trading Service je nach Verfügbarkeit an- und abmelden, und Klienten könnten beim Trading Service eine Objektreferenz auf einen gerade verfügbaren Drucker bekommen.

Darüber hinaus bietet CORBA Standards bzw. Standardisierungsversuche auf einer weiteren Ebene, den sogenannten CORBAfacilities. Diese erfüllen größere, aber dennoch allgemeine Aufgaben in einem verteilten Objektsystem. Dazu gehören die User Interface Common Facilities, Information Management Common Facilities, Systems Management Common Facilities, Task Management Common Facilities und die Workflow Management Common Facilities. Diese Facilities sind zwar von ihrem Ansatz her sehr interessant, ihre Verfügbarkeit ist jedoch noch sehr gering. Manche Standards wurden aus Mangel an vorgelegten Implementierungen bereits wieder zurückgezogen. Dies betrifft beispielsweise Teile der Information Management Common Facilities (Compound Presentation and Compound Interchange Facility und Data Interchange Facility).

Eine ausgezeichnete Diskussion der CORBAservices und CORBAfacilities findet man in [31], wo der Autor Wolfgang Schulze als Mitglied der OMG Workflow Working Group auch wertvolle Insider-Informationen gibt.

Im Rahmen der SAL wird CORBA als Verteilungstechnologie verwendet: Jeder Dienst hat ein CORBA-Objekt, das Nachrichten empfängt und an den Dienst weiterleitet. Der CORBA Naming Service wird verwendet, damit sich die einzelnen Dienste unter einem eindeutigen Namen registrieren und über diesen gefunden werden können. Eine darüber hinaus gehende Verwendung von CORBA wäre im Rahmen des Logischen Klienten möglich; der Strategie-Service könnte beispielsweise als CORBA Trading Service implementiert werden. Der erwartete Gewinn ist

dadurch allerdings nicht sehr groß; das in der Folge entwickelte Dienstleistungskonzept setzt auf eine syntaktische Einheitlichkeit der Schnittstellen, und die in diesem Zusammenhang anfallenden Koordinationsaufgaben müßten auch bei einer durchgängigeren Verwendung von CORBA gelöst werden. Außerdem war eine stärkere Einbindung bzw. Abhängigkeit von CORBA im Rahmen des Logischen Klienten nicht gewünscht. Der Einsatz von CORBA beschränkt sich damit auf dessen Kernkompetenz: Methodenaufrufe (nämlich das Absetzen von Nachrichten) zwischen verteilten Objekten (Diensten).

2.6.2 Java 2 Enterprise Edition

Java 2 Enterprise Edition (J2EE) ist eine von der Firma Sun 1999 herausgegebene Spezifikation einer auf Java aufbauenden Gesamtarchitektur [32]. Es existiert eine frei verfügbare Referenzimplementierung von Sun, und weitere J2EE-Produkte sind inzwischen auf dem Markt.

J2EE ist eine Gesamtarchitektur mit dem Anspruch, komplexe Geschäftsprozesse im E-Businessbereich vom Web Server bis zur Datenbank zu unterstützen [33]. Gleichzeitig gibt J2EE ein Anwendungsprogrammiermodell vor, das für bestimmte Aufgaben bestimmte Schichten vorschlägt und so eine Leitlinie für die Entwicklung komplexer Systeme darstellt.

Für solche Gesamtsysteme stellt J2EE eine Ablaufumgebung und eine Vielzahl von unterstützenden Diensten mit spezifizierten Schnittstellen bereit. Die wichtigsten wären der Naming und Directory Service zum Auffinden von Objekten und Dateien, Datenbankzugriffsdienste nach der JDBC-Schnittstelle, Einbindung von CORBA, Applets und Java Beans, Schnittstellen zur Verwendung von elektronischer Mail, eine Schnittstelle zum Austausch von Nachrichten zwischen Komponenten, Transaktionsdienste sowie dynamische Webseiten und Java Server Pages.

Für die eigentlichen Anwendungen sind spezifische Komponenten zu entwickeln, sogenannte Java Enterprise Beans. Diese müssen eine allgemeine Schnittstelle zum Erzeugen, Zerstören, Aktivieren und Deaktivieren bereitstellen. Der Code des Enterprise Beans wird beim Kompilieren so aufbereitet, dass die J2EE-Laufzeitumgebung Komponenten bei Bedarf erzeugen und verwalten kann sowie auf die bean-spezifische Schnittstellen zugreifen kann.

Die Java Enterprise Beans fallen in zwei Klassen: Die sogenannten Session Beans existieren, solange der entsprechende Klient mit dem System verbunden ist. Ein solches Session Bean könnte beispielsweise die Abläufe beim Buchen einer Flugreise implementieren. Die zweite Klasse sind die Entity Beans, welche einen längeren Lebenszyklus haben. Ein Entity Bean könnte zum Beispiel die Daten einer Flugbuchung kapseln und damit verbundene Operationen durchführen.

Java Enterprise Beans laufen in einer kontrollierten Umgebung ab, dem Container. Der Container liest aus den beim Kompilieren des Beans aufbereiteten Eigenschaften ab, welche Laufzeitbedingungen das Bean braucht, ob sein Zustand automatisch auf einer Datenbank persistent zu machen ist, wie es erzeugt und zerstört wird und wie die spezifischen Funktionen aufgerufen werden. Eine wesentliche Stärke von J2EE liegt somit in der automatischen Code-Generierung zur Erstellungszeit. Der generierte Code übernimmt bedarfsangepaßt viele Aufgaben, um die sich der Programmierer in weiterer Folge nicht zu kümmern braucht. Im

Vergleich zu CORBA bietet J2EE zwar keine programmiersprachenunabhängige Umgebung, dafür aber stärker integrierte und einfacher verwendbare Basisfunktionalitäten.

Während beim Logischen Klienten die Kompenenten (Dienste) ständig existieren, liegt der Fokus von J2EE auf der dynamischen Erzeugung von Session und Entitiy Beans. Damit liegt J2EE in seinem Ansatz quer zum Logischen Klienten. Inwiefern die wesentlichen Elemente des Logischen Klienten gewinnbringend auf J2EE übertragen werden könnten, wäre noch zu prüfen. Zur Entwicklungszeit des Logischen Klienten war J2EE noch nicht verfügbar.

Ein Dienstleistungskonzept mit syntaktisch einheitlichen Dienstleistungen ist bei J2EE nicht vorgesehen; es müßte ebenso wie im Logischen Klienten erst entwickelt werden. Auch die damit verbundenen Probleme wie das dynamische Auffinden von Dienstleistungen, ihre Kombination und ihre semantische Beschreibung ist bei J2EE offen. So gesehen scheinen die Anforderungen des Logischen Klienten mit J2EE zwar einfacher umzusetzen zu sein, weil viele Basisdienste bereits existieren, die grundsätzlichen und auch in dieser Arbeit behandelten Aufgaben würden aber auch bei Verwendung von J2EE anfallen.

2.6.3 Komponentenbasierte Systeme

Auf der Basis von Komponententechnologien wie CORBA und Enterprise Java Beans ist es nun relativ leicht möglich, komplexere Systeme aus einzelnen Komponenten zusammenzustellen und eine Zusammenarbeit zwischen diesen zu organisieren.[9] Das Ergebnis ist eine große Zahl an sogenannten komponentenbasierten Systemen, aber auch an komponentenbezogener Literatur. Wolfgang Göbl fasst diese Situation schön zusammen: "Meiner Meinung nach wurde der heutige Komponenten-Hype durch das Reifen der Komponententechnologien ausgelöst." [36]

Dementsprechend gibt es viele Definitionsversuche von Komponenten. Hier sei die Sichtweise von Clemens Szyperski wiederholt, der folgende Charakteristik gibt [37]:

- Komponenten sind Einheiten, die unabhängig voneinander eingesetzt werden (können). Daraus resultiert die Notwendigkeit von klar definierten Schnittstellen.
- Komponenten sind Einheiten, die von Dritten in anderen Zusammenstellungen (Kompositionen) verwendet werden können.
- Komponenten haben keinen persistenten Zustand.

Für Systeme solcher Komponenten sind weitere Voraussetzungen notwendig: "Für heutige Anwendungen genügt ein entfernter Methodenaufruf jedoch nicht mehr. Es bedarf in der Regel zusätzlicher infrastruktureller Dienste, wie Transaktionen, Sicherheitsmechanismen oder Ereignisdienste" [38].

---

[9] Gewöhnlich wird bei der Diskssion von Komponententechnologien und CORBA immer auch COM bzw. DCOM angeführt. Dabei handelt es sich um eine programmiersprachenunabhängige, aber plattformabhängige Komponententechnolgie, die von Microsoft entwickelt wurde. Eine nähere Betrachtung ist an dieser Stelle unnötig, weil damit keine weiteren Einblicke in Prinzipien des Dienstleistungskonzepts bzw. des Logischen Klienten verbunden ist. Eine ausführlichere Beschreibung von COM/DCOM findet man z.B. in [34] und [35].

Im Lichte dieser Eigenschaften und Definitionen läßt sich auch der Logische Klient als komponentenbasiertes System auffassen. Die Dienste haben in der Regel keine Zustände, und erbringen ihre Dienstleistungen unabhängig von der bisherigen Geschichte. Sie können unabhängig voneinander eingesetzt werden und in verschiedenen Konstellationen auftreten. Zusätzlich gibt es "infrastrukturelle Dienste" wie den Strategie-, Workflow-, Repository- und Resource-Service.

Das Dienstleistungskonzept und der Logische Klient gehen in einem entscheidenen Aspekt über die existierenden Komponententechnologien hinaus: Es gibt für alle Dienste im System eine syntaktisch einheitliche Schnittstelle. Damit ist es auch möglich, dass intelligente Systeme, die die semantische Schnittstellenbeschreibung zur Laufzeit auswerten können, sehr flexibel und leistungsfähig werden. Die Kombinationsfähigkeit der Komponenten/Dienste wird dadurch erheblich gesteigert. Dieser Vorteil wird im weiteren Verlauf der Arbeit noch klarer herausgearbeitet.

2.6.4 Blackboard-Architekturen

Das Prinzip von Blackboard-Architekturen lässt sich schön durch folgende Metapher beschreiben:

> Imagine a group of human or agent specialists seated next to a large blackboard. The specialists are working cooperatively to solve a problem, using the blackboard as the workplace for developing the solution. Problem solving begins when the problem and initial data are written onto the blackboard. The specialists watch the blackboard, looking for an opportunity to apply their expertise to the developing solution. When a specialists finds sufficient information to make a contribution, he records the contribution on the blackboard. This additional information may enable other specialists to apply their expertise. This process of adding contributions to the blackboard continues until the problem has been solved. [39]

Diese Architektur zeichnet sich durch folgende Eigenschaften aus:

- Zentraler Informationsbestand. Alle Informationen sind an einer zentralen Stelle (Blackboard) gesammelt, und dort werden sie auch ergänzt oder verändert.
- Keine Verteilung. Alle Spezialisten (Agenten) greifen direkt auf das Blackboard zu. Es gibt zwar Versuche, die Verteilung der Agenten durch eine Art intelligentes Blackboard zu ermöglichen [40], aber dies führt zu mehrfacher Informationshaltung mit Synchronisationsaufgaben.
- Inhärent regelbasiertes Vorgehen. Da jeder Spezialist nach seinen eigenen Fähigkeiten den Informationsbestand verändert, läßt sich der Vorgang auch als die Anwendung von speziellen Regeln auf einer gemeinsamen Knowledge Base auffassen. Wie bei jedem regelbasierten System ist die Konvergenz der Einzelbeiträge zu einer Lösung nicht a priori gewährleistet. Wohl aber kann durch ein sorgfältiges Design der Regeln (bzw. der Fähigkeiten der beteiligten Spezialisten) eine Konvergenz in einer Vielzahl von Fällen erreicht werden.

Andererseits läßt sich auch der Workflow-Service als intelligentes und session-spezifisches Blackboard auffassen, das als zentrales Informationspool auf der Basis von Workflow-

Beschreibungen die Verteilung der Informationen an die Spezialisten vornimmt und deren Ergebnisse in das zentrale Informationspool wieder eingliedert.

2.6.5 Agentensysteme nach FIPA

FIPA ist ein Akronym für Foundation of Intelligent Physical Agents, eine Organisation, die Standardisierungsvorschläge aus dem Bereich der Agenten erarbeitet [41]. Diese Vorschläge sind auf einem hohen Abstraktionsniveau. Für verschiedenen Teilbereiche gibt es konkrete Standards, wie beispielsweise für die Einbindung von externer Software, persönliche Assistenten, Audio/Video-Unterhaltungssysteme, und persönliche Reiseassistenten. Die FIPA-Architektur stützt sich auf drei Säulen:

- Messaging. Die Kommunikation zwischen den Agenten erfolgt über Nachrichten.
- Agent Communication Language. FIPA schlägt ein Nachrichtenformat vor, das unabhängig von den zu übertragenen Inhalten und gleichzeitig offen für verschiedene Formatierungsvorschriften ist.
- Directory. Damit Agenten miteinander kooperieren können, gibt es Verzeichnisdienste, bei denen die Agenten ihre Leistungen und Fähigkeiten anmelden und passende Agenten finden können.

Konkrete Implementierungen konkreter Standards, die sich für die Umsetzung des Dienstleistungskonzepts eignen würden, waren nicht verfügbar. In diesem Zusammenhang ist nochmals zu betonen, dass die FIPA-Architektur sehr abstrakt ist. Ob sich das allgemeine Ziel, nämlich die Interoperabilität von konkreten, FIPA-entsprechenden Agenten unabhängig von deren Herkunft, erreichen läßt, muss vorläufig offen bleiben.

Brauchbar ist hingegen das Nachrichtenkonzept, auf welches sich die SAL wesentlich stützt. Auch sind viele Aspekte der FIPA-Architektur im Logischen Klienten enthalten und konkret umgesetzt.

3 Die SAL als Dienstleistungsframework

Dieser fachlichen Einführung und der Darstellung der Ausgangspunkte soll die Darstellung der eigentlichen Arbeit folgen. In diesem Kapitel wird die SAL als Dienstleistungsframework detailliert beschrieben. Dabei geht es um Anforderungen, Rahmenbedingungen, Entwurfsentscheidungen, konzeptionelle Entscheidungen und deren Umsetzung, die Zusammenarbeit mit der Ontologie, Unterstützung durch Skripte, Fehlerbehandlung und um allgemeine Dienstleistungen. In den darauffolgenden Kapiteln 4 und 5 werden Erfahrungen mit der SAL auf Detail- und Systemebene berichtet und analysiert, Vergleiche angestellt und mögliche Szenarien für die zukünftige Arbeit entwickelt.

Die Namensgebung der SAL und ihre grundlegenden Aufgaben lassen sich am besten anhand von Abbildung 8 veranschaulichen. Zwei Dienstleistungen – Service A und Service B – müssen in das System integriert werden. In der KFÜ waren dies unter anderem eine Vielzahl an Teilaufgaben, zu deren numerischen Lösung FORTRAN-Module existieren. Sie verfügen über die in der Dienst-Definition geforderten Schnittstelle, über die man die Dienstleistung in Anspruch nehmen kann. Diese Schnittstellen waren in der Praxis jedoch weniger klar und frei von Implementierungsentscheidungen als es wünschenswert wäre. Auch konnten die bisherigen Dienste nicht miteinander kooperieren, was für Agenten eine notwendige Eigenschaft ist. Die Erweiterung solcher Dienste bzw. Dienstleistungen zu vollwertigen Agenten erfolgt mit Hilfe der SAL.

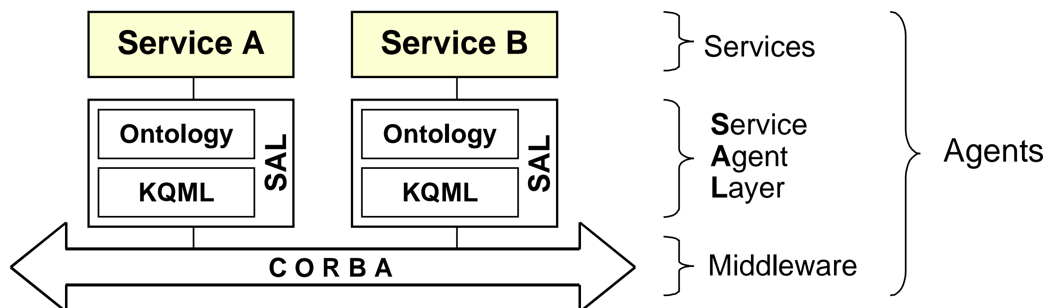


Abbildung 8: Ebenen der Kommunikation

Abbildung 8 zeigt auch die verschiedenen Ebenen der Kommunikation, welche bei der Vermittlung von Dienstleistungen eine fundamentale Bedeutung hat. Diese kann auf verschiedene Arten erfolgen. Einfache Kommunikationsmittel – zum Beispiel die Verwendung gemeinsam zugreifbarer Dateien oder Speicherbereiche – erlauben eine sehr flexible, aber auch unstrukturierte und leicht ins Unübersichtliche tendierende Art der Kommunikation. Im Interesse der Klarheit der Schnittstellen ist die Kommunikation über eigens dafür vorgesehene Kommunikationskanäle, -schnittstellen und -strukturen auf jeden Fall vorzuziehen. Für die Umsetzung einer solchen strukturierten Kommunikation fiel die Entscheidung auf Grund der geforderten Plattformunabhängigkeit auf CORBA [29] (vgl. auch Kapitel 2.6.1).

Damit steht für die physikalische Ebene der Kommunikation, also für die Übertragung von Daten, ein wohlstrukturierter, expliziter und leistungsfähiger Mechanismus zur Verfügung. Diese Ebene der Kommunikation ist in Abbildung 8 ganz unten dargestellt. In Analogie zum

Briefverkehr greift man damit auf ein System zurück, das Briefe entgegennimmt und an anderer Stelle abliefert.

Die nächste Ebene beschäftigt sich mit den Fragen, wie solchen Briefe aufgebaut sind und wie mit ihnen zu verfahren ist. Jeder Brief hat einen Empfänger und meist einen Absender. Wenn der Empfänger nicht erreichbar ist, wird der Brief zurückgeschickt. Damit der Empfänger sich bei seiner Antwort auf den Brief beziehen kann, gibt es Konventionen bezüglich des Betreffs und des Datums des Briefs. Es gibt bestimmte Briefarten für bestimmte Absichten. So genügt es beispielsweise, eine Bestellung als solche kennzuzeichnen, um beim Empfänger eine Kette von Aktionen auszulösen. Fragen dieser Art sind meist vom eigentlichen Inhalt des Briefs unabhängig. Sie zielen auf das zugrundeliegende Protokoll. Die SAL greift auf dieser Ebene stark auf die Knowledge Query and Manipulation Language (KQML) zurück, welche im anschließenden Kapitel näher beschrieben wird.

Die dritte Ebene ist die semantische und wird manchmal übersehen. Die beste Middleware und das ausgefeilteste Protokoll sind wertlos, wenn bezüglich des Inhalts zwischen Empfänger und Sender kein gemeinsames Verständnis herrscht. Dieses Verständnis wird im Alltag implizit vorausgesetzt und ist auch meist vorhanden. Im Software-Bereich muss ein solches gemeinsames Verständnis auf formaler Ebene erst erreicht werden. Dazu kann es in einfachen Systemen genügen, wenn sich die Entwickler der beteiligten Programme darauf einigen, was sie unter welchen Kommunikationsinhalten verstehen. Wesentlich mächtiger, zuverlässiger, klarer und erweiterbarer ist die Verwendung einer gemeinsamen Ontologie. Der Integration der SAL mit Ontologien ist Kapitel 3.4 gewidmet.

Der Schwerpunkt der SAL liegt auf der Protokollebene und der semantischen Ebene (vgl. Abbildung 8). Sie fordert die Verwendung einer Ontologie, ohne von ihr inhaltlich abhängig zu sein, sie benutzt ein gemeinsames Protokoll und Nachrichtenformat (KQML), sie greift auf eine gemeinsame Middleware zurück (CORBA). Auf diese Weise kann man relativ einfache Dienstleistungen zu Diensten bzw. Agenten zusammenfassen.

3.1 Nachrichtenübertragung

Zur Koordination von Dienstleistungen ist Kommunikation notwendig. Für diese Kommunikation wurde der bei Agentensystemen weit verbreitete Ansatz der Nachrichtenübertragung gewählt [42]. Die Nachrichten sind dabei prinzipiell offen für beliebige Inhalte. Hinsichtlich der Art, des Aufbaus und der Abfolge von Nachrichten orientiert sich die SAL stark an der Knowledge Query and Manipulation Language (KQML, [43]). Diese hat jedoch auch einige Schwächen beim Einsatz im Logischen Klienten und bei der Vermittlung komplexer Dienstleistungen. Trotz dieser Schwächen, die in weiterer Folge genauer analysiert werden, hat KQML eine ausgezeichnete Ausgangsbasis als Kommunikationssprache abgegeben.

KQML wurde von Tim Finin und Yannis Labrou entwickelt [43]. In Abbildung 9 wird eine KQML-Nachricht dargestellt. Dieses Beispiel ist bewusst einfach gewählt; Probleme, die bei der Verwendung realistischerer Nachrichten auftreten, werden in weiterer Folge diskutiert. Im Beispiel möchte Agent Axel von Agent Roland alle Adressbucheinträge wissen, deren Name Astrid ist. Die Angabe "AddressBook" im Feld *in-reply-to* setzt voraus, dass Agent Roland sich zuvor in einer Nachricht bereit erklärt hat, Fragen nach Adressen zu beantworten. In dieser

Nachricht hätte im *reply-with*-Feld "AddressBook" stehen müssen. Umgekehrt sollte Roland bei seiner Antwort das Feld *in-reply-to* auf "QueryAstrid" setzen, damit Axel die Antwort von Roland zuordnen kann. Die Angabe der *language* "KIF" bedeutet, dass der Inhalt syntaktisch dem Knowledge Interchange Formalism [44] entspricht. Durch den Verweis auf die *ontology* "people" wird sichergestellt, dass sich sowohl Axel als auch Roland auf dieselbe Ontologie verständigen und dass die im *content* angegebenen Begriffe dieser Ontologie entsprechen. In dieser fiktiven Ontologie ist die Struktur der *address*-Klasse festgelegt.

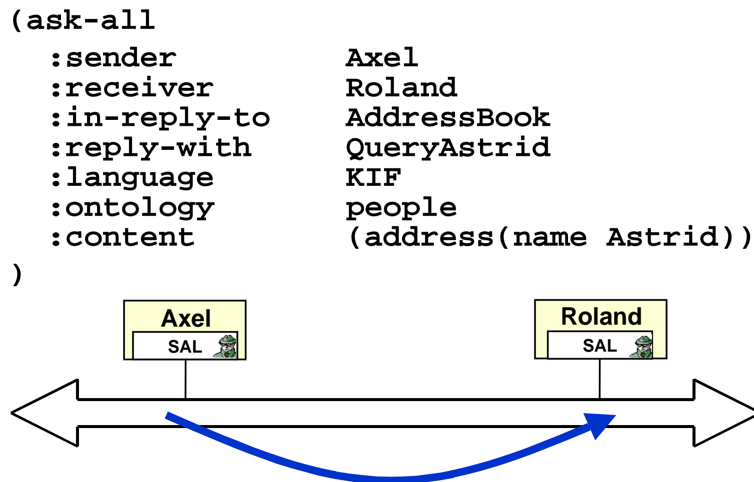


Abbildung 9: Eine typische KQML-Nachricht

Der Aufbau dieser Nachricht folgt einem strengen Schema. Das Wort *ask-all* ist ein Schlüsselwort, das die Absicht dieser Nachricht beschreibt. KQML lehnt sich hier an die Sprechakttheorie an und verwendet eine festgelegte Menge sogenannter Performativen. *ask-all* bedeutet, dass der Sender an allen Informationen der Klasse *address* interessiert ist, deren Feld *Name* auf Astrid gesetzt ist. Im Gegensatz dazu würde die Performative *ask-one* bedeuten, dass Axel mit einer einzigen solchen Information zufrieden ist. Schließlich hätte er noch die Möglichkeit, mittels *ask-if* herauszufinden, ob Roland eine solche Information hat, ohne sie jedoch konkret anzufordern. Die Performativen der KQML lassen sich folgendermaßen gruppieren[10]:

- Anfrageperformativen: ask-if, ask-all, ask-one, stream-all (die Ergebnisse kommen nach und nach in eigenen Nachrichten).
- Antwortperformativen: tell, untell (keine derartigen Informationen verfügbar), deny (es gibt eine Information, aus der auf die Nichtexistenz der gefragten Information geschlossen werden kann), eos (end of stream, die letzte Antwort auf eine stream-all-Anfrage).

[10] Diese Aufzählung ist nicht ganz vollständig. Weitere, für diese Diskussion allerdings unergiebige und im SAL-Framework nicht verwendete KQML-Performativen sind: register, unregister, transport-address, recruit-one und recruit-all.

- Aufforderungen, Informationen aufzunehmen bzw. zu löschen: insert, uninsert (Rücknahme eines vorangegangenen inserts), delete-one, delete-all, undelete (Rücknahme eines deletes).
- Aufforderungen, die Umgebung so zu beeinflussen, dass etwas wahr wird: achieve, unachieve (ein vorheriges achieve soll zurückgenommen werden).
- Fehlernachrichten: error (eine vorhergehende Nachricht wird für fehlerhaft gehalten), sorry (Nachricht verstanden, aber keine Antwort möglich).
- Zeitlich speziell angeordnete Abfragen: standby (wie eine ask-Performative, aber wenn der Empfänger etwas zu senden hat, dann soll er das signalisieren, ohne die Informationen mitzuschicken), ready (ein solches Signal), next (Aufforderung, eine Information zu schicken), rest (Aufforderung, alle Informationen auf einen Satz zu schicken), discard (der Auftraggeber teilt mit, dass ihn der Rest nicht mehr interessiert).
- Inserate: advertise (der Sender gibt bekannt, dass er eine spezielle Nachricht verarbeiten kann; eine advertise-Nachricht enthält im *content* diese spezielle Nachricht), unadvertise (Rücknahme eines advertise).
- Abonnements: subscribe (der Sender bittet den Empfänger, von allen Änderungen hinsichtlich der im *content* erwähnten Informationen in Kenntnis gesetzt zu werden).
- Vermittlungsnachrichten: forward (der Sender möchte die Nachricht an einen dritten Agenten weiterleiten lassen), broadcast (der Sender möchte die Nachricht an alle bekannten Agenten weiterleiten lassen), recommend-one (enthält im *content* eine Nachricht; der Sender fragt nach einer Empfehlung, welcher Agent diese Nachricht verarbeiten kann), recommend-all (der Sender fragt nach allen verfügbaren Empfehlungen), broker-one (der Sender bittet den Empfänger, die im *content* enthaltene Nachricht an einen geeigneten Agenten weiterzuleiten), broker-all (der Sender bittet um Weiterleitung an alle geeigneten Agenten).

Bei genauerer Betrachtung dieser Nachrichtentypen werden folgende, wesentliche Eigenschaften der KQML deutlich:

1. Eine Nachricht kann eine andere enthalten. Dies ist bei advertise, recommend und broker der Fall. Damit wird eine Rekursivität ermöglicht, die in der Tiefe prinzipiell nicht beschränkt ist.
2. Es gibt Nachrichten, die Antworten erfordern (zum Beispiel Anfragen), aber auch Nachrichten, die dies nicht tun (zum Beispiel Antworten oder Inserate). KQML legt genau fest, welche Nachrichten auf welche folgen müssen. Diese Festlegung wird auch Protokoll genannt.
3. Es gibt Nachrichten, die keinen eigenen Inhalt tragen müssen, weil sie eine konkrete Aufgabe in einem Kommunikationszusammenhang übernehmen. Hierzu gehört eos, aber auch deny, error und sorry. Bei solchen Nachrichten gibt es das in Abbildung 9 erwähnte Feld *content* nicht.
4. Es gibt Nachrichten, die über die erwähnten Felder hinaus weitere Felder benötigen. Hier ist in erster Linie forward zu nennen, welches die Angabe des Agenten benötigt, an den

weitergeleitet werden soll (Feld *to*). Außerdem wird der weitere Empfänger informiert, von wem die Nachricht kam und an wen daher Antworten gehen sollen (Feld *from*).

5. An der achieve-Performative erkennt man, dass das in der Agentendefinition geforderte Konzept der *Umgebung* auch von KQML angenommen wird.
6. An den Aufforderungen, Informationen aufzunehmen und zu löschen sowie am Namen der KQML – Knowledge Query and Manipulation Language – erkennt man den starken Zusammenhang der KQML mit dem Konzept einer Wissensbank oder Knowledge Base. Am einfachsten sind die Performativen der KQML umzusetzen, wenn man direkten Zugriff auf eine Knowledge Base hat und die Zugriffe auf diese erst zur Laufzeit ausgewertet werden müssen. Umgekehrt können die subscribe-, recommend- oder broker-Anfragen einfach umgesetzt werden, wenn aus der Knowledge Base heraus bei Vorhandensein der entsprechenden Umstände die Nachrichten erzeugt und verschickt werden.
7. Das hinter den Performativen advertise und recommend stehende Konzept der Agentenvermittlung passt gut zu den Aufgaben des Strategie-Service.

Wichtig ist über die Performativen hinaus der Umstand, dass KQML unabhängig vom *content* ist. In den Feldern *language* und *ontology* werden zwar dessen Rahmenbedingungen näher beschrieben, aber für die Übertragung einer Nachricht ist dieser *content* irrelevant – außer, er ist fehlerhaft formuliert und führt beim Aufbereiten der Nachricht zu Fehlern.

Der erste Prototyp der SAL hat KQML-Nachrichten direkt an den agentenspezifischen Code weitergegeben, welcher sich um die weitere Bearbeitung kümmern musste. Bald wurde jedoch klar, dass diese für die SAL sehr bequeme Lösung nicht sinnvoll ist. Das Dienstleistungsframework kann und soll dem Entwickler des Agenten wesentlich mehr abnehmen und vorgeben. Wenn jeder Dienst die komplexen Kommunikationsregeln selbst beherrschen müsste, würde der Gesamtaufwand zur Entwicklung eines System steigen, und gleichzeitig würde das Risiko wachsen, dass bei unvollständiger Spezifikation des Transaktionsprotokolls oder bei der Umsetzung Missverständnisse wachsen, die den Erfolg der Kommunikation gefährden.

Die SAL geht vor allem in folgenden drei Bereichen über die KQML hinaus bzw. weicht von ihr ab:

- In einem von dienstebasierten System geht es vor allem um die Erbringung von Dienstleistungen. Alle Nachrichten dienen dazu, die Erbringung einer Dienstleistung anzubahnen und abzuwickeln. Die SAL unterstützt ein durchgängiges Dienstleistungskonzept. Der Agentenentwickler muss von der im Hintergrund ablaufenden Nachrichtenübertragung nichts wissen.
- Durch die durchgängige Verwendung des Dienstleistungskonzepts und die Verwendung einer höher strukturierten Middleware ist es möglich, das Transaktionsprotokoll zu vereinfachen.
- KQML ist vom Inhalt der Nachricht unabhängig. Dies ist zunächst weder ein Vorteil noch ein Nachteil, sondern eine Abgrenzung: Weil der Inhalt offen für die Interpretation ist, muss

an anderer Stelle im System festgelegt werden, wie diese Interpretation zu erfolgen hat. Hier springt die SAL ein, macht Vorgaben und unterstützt deren Erfüllung.

Diese drei Bereiche werden in den folgenden drei Unterkapiteln 3.2, 3.3 und 3.4 detailliert beschrieben.

3.2 Das Dienstleistungskonzept in der SAL

Die SAL hat aus dem Alltag bzw. von dienstebasierten Systemen (vgl. Kap. 2.1) das Konzept der Dienstleistung entlehnt. Synonym dazu werden die Begriffe Job und Auftrag verwendet. Bei jeder Erbringung einer Dienstleistung gibt es einen Auftraggeber und einen Auftragnehmer.

Die SAL stellt allgemeine Mechanismen zum Nachfragen nach einer Dienstleistung für den Auftraggeber zur Verfügung. Auf der Auftragnehmerseite muss die jeweilige Dienstleistung erbracht werden. Dazu bringt der Agent bzw. dessen Entwickler eine Implementierung. Die SAL sorgt bei einer Nachfrage – auch Request genannt – dafür, dass der Auftragnehmer den richtigen Job beginnt. Request und Job stehen dabei in einer Eins-Zu-Eins-Beziehung, und die SAL verwaltet diese Beziehungen.

Im folgenden werden fünf unterschiedlich komplexe Beispiele für solche Dienstleistungen genannt, anhand derer die weiteren Eigenschaften des von der SAL umgesetzten Dienstleistungskonzepts erläutert werden. Den Abschluß dieses Unterkapitels bildet eine Bewertung des Dienstleistungskonzepts.

3.2.1 Beispiele für Dienstleistungen

Hier sollen fünf unterschiedlich komplexe Dienstleistungen als Ausgangsbasis für die Diskussion der wichtigsten Anforderungen skizziert werden:

1. Auskunftsdienstleistung: Im Zusammenhang mit der Einführung der KQML hat der fiktive Agent Roland eine Auskunftsdienstleistung angeboten: Unter Angabe eines Namens konnte man eine Adresse erfragen, sofern diese in seinem – virtuellen oder realen, verteilten oder zentralen – Adressbuch eingetragen ist.

2. Auskunftsdienstleistung mit Nebenbedingungen: In der KFÜ gibt es einen Dienst, der unter der Angabe eines Zeitraums und eines Raumes Windmesswerte liefert. Dabei erfolgt die Angabe des Raumes durch Nennung des Modellgebiets, welches als ein geodätisches Rechteck um ein Kernkraftwerk definiert ist. Dieser Dienst greift auf eine zentrale Datenbank zurück, die im 10-Minutentakt mit Messwerten gespeist wird. Für Simulationszwecke gibt es eine von der Standard-Datenbank abweichende Simulationsdatenbank, weshalb der Dienst als Eingabe auch noch die Nennung der relevanten Datenbank benötigt.

3. Alternative Auskunftsdienstleistung: Für den Fall, dass keine Messwerte vorliegen, gibt es in der KFÜ den Ersatzwert-Service, welcher unter Nennung des Zeitraums und des Modellgebiets ebenfalls Windwerte liefert.

4. Auskunftsdienstleistung mit numerischer Berechnung für einzelne Zeitschritte: Es gibt eine Dienstleistung, die aus dem Modellgebiet, zugehörigen Topographiedaten und Windwerten ein für diese Topographie gültiges Windfeld berechnet. Dies ist eine numerisch sehr

aufwendige Rechnung. Als Eingabe benötigt sie noch eine Z-Achsen-Aufteilung, anhand derer die Höhenangaben im dreidimensionalen Windfeld erfolgen sollen und einen Zeitraum, auf den sich die Eingabe-Windwerte und das Ergebnis-Windfeld beziehen.

5. Auskunftsdienstleistung mit numerischer Berechnung für den Gesamtzeitraum: Für die Transportrechnung – also die Berechnung, welche Nuklide sich zu welcher Zeit an welchem Ort befinden – wird ein FORTRAN-Modul für den gesamten Berechnungszeitraum gestartet. Zu jedem Zeitschritt erhält es die Angaben, welche Nuklide beim Kraftwerk emittiert werden. Es bringt diese Nuklide in das Windfeld ein und ermittelt unter Berücksichtigung des Transports im Windfeld und bereits im Windfeld aus früheren Zeitschritten vorhandener Nuklide die jeweiligen Nuklidkonzentrationen. Diese Nuklidkonzentrationen liegen in einem dreidimensionalen Maschennetz vor und werden über den gesamten Berechnungszeitraum im Hauptspeicher gehalten.

Anhand dieser Beispiele wollen wir nun die wichtigsten Eigenschaften des Dienstleistungskonzepts in der SAL beschreiben. Als Vergleich wird immer wieder die KQML dienen, zum einen um die Abweichungen von ihr zu rechtfertigen, zum anderen um zu veranschaulichen, dass mit dem Dienstleistungskonzept eine wesentliche Abstraktionsstufe genommen wurde, die die Implementierungen der Agenten deutlich vereinfacht.

3.2.2 Ein- und Ausgabe

Dienstleistung 1 konnte als aussagekräftiges Beispiel für die Formulierung einer KQML-Abfrage dienen. Dazu musste die gesuchte Information referenziert werden. Im *content*-Feld stand dem entsprechend eine Beschreibung des gewünschten Ergebnisses. Dies wurde in Übereinstimmung mit der Sprache KIF als Schablone angegeben, und die Nennung des Attributs *name* mit dem zugehörigen Wert *Astrid* hatte die Bedeutung einer Einschränkung. Alle Attribute, die in der Anfrage nicht vorgekommen sind, waren frei. Ein Eintrag in Stuttgart wäre genauso akzeptabel gewesen wie ein Eintrag in Rom.

Nun bietet KIF mächtige Konstrukte zur Formulierung komplexer Zusammenhänge zwischen Datenobjekten. Prinzipiell wären so auch die Anfragen 2 bis 5 formulierbar. In der Praxis ergeben sich jedoch enorme Probleme. In Anfrage 2 müssten die gewünschten Windmesswerte gemäß KQML als Schablone angegeben werden, und mit der Performative ask-all gibt man bekannt, dass man an allen geeigneten Windmesswerten interessiert ist. In diese Schablone müssten der konkrete Zeitraum und das konkrete Modellgebiet eingetragen werden, so wie im Beispiel der Name *Astrid*. Doch während ein Objekt der Klasse *address* noch relativ einfach strukturiert ist und *name* ein ganz normales Attribut dieser Klasse war, sind die Verhältnisse zwischen Windmesswerten, Zeiträumen und Modellgebiet bereits wesentlich komplizierter. Außerdem ist z.B. das Modellgebiet selbst ein strukturiertes Objekt, das – anders als *name* bei der Klasse *address* – als gewöhnliches Attribut der Klasse Windmesswert kaum mehr in Frage kommt: Das Datenvolumen würde vervielfacht, wenn jeder Windmesswert das gesamte Modellgebiet als Attribut enthalten müßte.

Wie auch immer die Beziehungen zwischen Windmesswert, Zeitraum und Modellgebiet in der Ontologie modelliert werden, eine KQML-Abfrage müßte die Beziehungen zwischen ihnen abbilden und so komplizierter werden. So würde sich die Formulierung von Anfrage 2

wahrscheinlich bereits über eine halbe Seite erstrecken. Dies ist überschaubar, aber es ist erst die Spitze eines Eisbergs. Die Spezifikation der Dienstleistung 5 benötigt in der KFÜ viereinhalb Seiten. Eine KQML-konforme Inanspruchnahme der Dienstleistung würde kaum weniger Platz brauchen. Wenn es sein muss, dann kann man als Mensch solche Texte erstellen, und wenn es sein muss, kann man auch Programme schreiben, die eine KQML-Nachricht solchen Umfangs zusammenstellen. Spätestens wenn man jedoch bedenkt, dass KQML seine Stärke aus einer direkten Übertragbarkeit von Inhalten der Wissensbasis in Nachrichten und umgekehrt zieht und dass es in einem Agentensystem erwünscht wäre, einen erst zur Laufzeit bekannten Agenten (welcher eine advertise-Nachricht geschickt hat) in Anspruch zu nehmen, wird klar, dass ein solches System nicht mehr handhabbar wird. Die Entwicklungskosten für hart verdrahtete Abfragen sind zu hoch, und die Programmierung von Regeln zur Erstellung von ask-all-Nachrichten der geforderten Komplexität zur Laufzeit auf der Basis von nicht minder komplexen advertise-Nachrichten ist enorm schwierig. Ein weiterer Rückschlag kommt aus den Anforderungen, die eine solche Nachrichtenorganisation auf die Modellierung der Ontologie stellt. Es ist schwierig genug, eine Ontologie zu erstellen. Wenn man bei allen Datenobjekten zusätzlich berücksichtigen muss, in welchen Anfragen sie mit welchen anderen Datenobjekten in Beziehung stehen und wie man daher die Beziehung modellieren muss, dann wird es sehr schwer, komplexere Sachverhalte zu modellieren und es gleichzeitig allen vorgesehenen oder möglichen Dienstleistungen Recht zu machen.

KQML bleibt in der Praxis auf einfache Nachrichten beschränkt, und Abhängigkeiten und Beziehungen zwischen den Datenobjekten dürfen kaum auftreten. Die Lösung dieses Problems bei komplexeren Dienstleistungen liegt in der konsequenten Unterstützung des Dienstleistungskonzepts. Man schickt einem Auftragnehmer eine Nachricht, in der man nicht das gewünschte Ergebnis beschreibt, sondern sich auf eine Dienstleistung bezieht, die dieser Agent anbietet: Das *in-reply-to*-Feld enthält den Namen der Dienstleistung. Die Eingabedaten gehen im *content* als konkrete Datenobjekte mit. Eine Beschreibung der gewünschten Ergebnisse ist damit nicht mehr notwendig, und die Implementierung des Agenten entscheidet aufgrund des Namens der Dienstleistung, was zu tun ist.

3.2.3 Zeitmanagement bei Dienstleistungen

KQML geht davon aus, dass zwischen einer ask-Anfrage und der zugehörigen Antwort eine vernachlässigbare Zeitspanne vergeht. In der KFÜ gibt es jedoch zahlreiche Dienstleistungen, die 10 Minuten benötigen, und die in Beispiel 5 genannte Dienstleistung erstreckt sich sogar über den gesamten Rechnungszeitraum von maximal 72 Stunden. Wenn jedoch ein Alarm ausgelöst wird, dann müssen sofort alle weniger wichtigen Rechnungen unterbrochen und die Ressourcen freigegeben werden. Aus der erhöhten Dauer von Dienstleistungen ergeben sich damit folgende Anforderungen an die SAL:

- Auftragsbestätigung: Sobald eine Nachricht zur Inanspruchnahme einer Dienstleistung akzeptiert wird, erhält der Auftraggeber eine Auftragsbestätigung.
- Suspend: Laufende Dienstleistungen müssen angehalten werden können.
- Resume: Angehaltene Dienstleistungen müssen wiederaufgenommen werden können.

- Terminate: Dienstleistungen – egal ob angehalten oder nicht – müssen auch abgebrochen werden können, entweder um Ressourcen ganz freizugeben oder weil der Benutzer die Ergebnisse einer Rechnung nicht mehr benötigt.
- Persistenz: Sollte es während einer Dienstleistung zu einem Absturz kommen, dann soll es möglich sein, unter Minimierung von Daten- und Rechenzeitverlusten weitermachen zu können. Persistenz wird in Kapitel 3.8 näher beschrieben.
- Handhabung der Asynchronizität: Wen ein Programm eine Dienstleistung in Auftrag gibt, kann zwischen drei Möglichkeiten gewählt werden: Entweder wird der Programmablauf blockiert, bis die Dienstleistung erbracht ist, oder es werden spezielle Call-Back-Methoden implementiert, die von der SAL beim Ende der Dienstleistung aufgerufen werden und die es dem Auftraggeber ermöglichen, die Ergebnisse weiterzuverarbeiten. Als letzte Möglichkeit bleibt, dass das Programm immer wieder nachfragt, ob die Dienstleistung bereits beendet ist (Polling). Die SAL unterstützt alle drei Möglichkeiten.

Für die Umsetzung dieser Anforderungen mussten über KQML hinausgehend weitere Nachrichtentypen eingeführt werden. Diese werden in Kapitel 3.3 genauer beschrieben.

Bei den Suspend-, Resume-, Terminate- und Persistenzaufgaben ist es notwendig, dass die Implementierung des Agenten mit der SAL zusammenarbeitet. Die Dienstleistung selbst muss das Anhalten, Wiederaufnehmen und Abbrechen erlauben und unterstützen, und die SAL muss die Verbindung zwischen Auftraggeber und Auftragnehmer dabei herstellen. Praktisch wurde dies im Sinne des Framework-Ansatzes dadurch gelöst, dass die SAL eine abstrakte Job-Klasse definiert, die virtuelle Methoden zur Durchführung, zum Anhalten, Wiederaufnehmen und Abbrechen von Dienstleistungen definiert. Diese werden durch die Dienstleistungs-implementierung überschrieben. Dadurch wird das Zusammenspiel zwischen SAL und Agent ermöglicht. Grundsätzlich kann es jedoch auch sein, dass das Kommando, eine Dienstleistung abzubrechen, gar nicht durchgeführt werden kann, sei es, weil die Dienstimplementierung dies nicht unterstützt, sei es, weil die Erbringung der Dienstleistung in einem Zustand ist, der ein Abbrechen nicht erlaubt oder nicht mehr sinnvoll erscheinen lässt.

Somit können Nachrichten zur Inanspruchnahme einer Dienstleistung, zum Anhalten, Wiederaufnehmen oder Abbrechen auch scheitern. Dieses Scheitern muss wiederum dem Auftraggeber mitgeteilt werden, und die SAL muss die Zustände auf eine konsistente Weise verwalten.

3.2.4 Geschachtelte Dienstleistungen

Die Dienstleistung 5 unterscheidet sich von den anderen dadurch, dass sie sich über den gesamten Berechnungszeitraum erstreckt, aber gleichzeitig zu jedem Zeitschritt zusätzliche Eingaben benötigt und zusätzliche Ausgaben produziert. Diese Sonderstellung ist nicht einfach zu handhaben. Wenn man das Ganze als eine Dienstleistung aufsetzt, dann muss man die zeitschrittabhängigen Ein- und Ausgaben auf einem anderen Kanal transportieren, und die Auftraggeber müssen diesen korrekt bedienen. Wenn man für jeden Zeitschritt eine eigene Dienstleistung modelliert, und der Dienst selbst entscheidet, welche Dienstleistungen zu welcher Berechnung und damit zu welchem Lauf des FORTRAN-Moduls gehören, dann täuscht die Schnittstelle eine Unabhängigkeit der Dienstleistungen vor, die nicht existiert.

In diesem Dilemma fiel die Entscheidung für einen Mittelweg: Es ist möglich, eine Dienstleistung als Teildienstleistung einer anderen anzubieten und in Anspruch zu nehmen. Damit wird ein expliziter Verweis auf Zusammenhänge gegeben, während gleichzeitig der enge Zusammenhang zwischen Ein- und Ausgabedaten für einen Zeitschritt in Form von Teildienstleistungen erhalten bleibt. Auftraggeber müssen dazu wissen, welche Dienstleistungen Teildienstleistungen und welche Rahmendienstleistungen sind. So werden die Abhängigkeiten deutlich und explizit, während die Handhabung für den Auftraggeber relativ einfach bleibt.

Die SAL unterstützt solche Rahmen- und Teildienstleistungen. Die dabei entstehenden Abhängigkeiten zwischen Dienstleistungen haben in der Praxis zu erstaunlich wenig Problemen geführt.

3.2.5 Bewertung

Das Konzept der Dienstleistungen ist sehr leistungsfähig. Seine durchgehende Unterstützung durch die SAL vereinfacht die Integration von Aktivitäten in komplexen Systemen wesentlich. Das Dienstleistungskonzept wurde notwendig, weil die in der KFÜ vorkommenden Dienstleistungen und die dabei angefragten Datenobjekte und ihre Abhängigkeiten so komplex strukturiert sind, dass man mit reinen KQML-Nachrichten nicht mehr durchgekommen wäre.

Andererseits hat dieses Dienstleistungskonzept nicht nur in komplexen Simulationssystemen seine Berechtigung. Es hat sich so sehr bewährt, dass alle Nachrichten im System im Rahmen von Dienstleistungen ausgetauscht werden. Diese allgemeine Anwendbarkeit des Dienstleistungskonzepts findet seine Parallele im täglichen Leben, wo ebenfalls der überragenden Teil unserer menschlichen Aktivitäten in Form von Dienstleistungen organisiert ist.

Gleichzeitig wurde damit auch eine neue Abstraktionsebene erreicht. Agenten beschreiben ihre Wünsche nicht mehr durch die Nennung der gewünschten Datenobjekte sondern durch Bezugnahme auf eine Dienstleistung. Dieser Übergang von Datenobjekten zu Dienstleistungen stellt einen logischen weiteren Schritt in der Strukturierung komplexer werdender Agentenaktivitäten dar.

Damit verbunden ist jedoch auch die Notwendigkeit, über Dienstleistungen an sich zu kommunizieren. Konnte dies in KQML durch die Schachtelung von Nachrichten geschehen, muss dies nunmehr explizit erfolgen. In offenen Systemen müssen solche Dienstleistungsbeschreibungen durch Software-Agenten automatisch gelesen und ausgewertet werden. Hierzu muss die Ontologie Sprachmittel zur Verfügung stellen, um eine Dienstleistung hinsichtlich ihrer Ein- und Ausgabe und ihrer Semantik zu beschreiben. Im abgeschlossenen System der KFÜ reicht zur Beschreibung eine in natürlicher Sprache formulierte Dienstleistungsbeschreibung. Der Entwickler eines Auftraggebers entscheidet dann auf der Basis dieser Dienstleistungsbeschreibung, ob die Dienstleistung verwendet werden kann, welche Eingabeobjekte notwendig sind, welchen Nebenbedingungen sie genügen müssen und welche Ausgabeobjekte die Dienstleistung liefern wird.

Dass diese Beschreibung der Semantik sinnvoll ist, wird beim Vergleich von Dienstleistung 2 und 3 klar. Die Ein- und Ausgaberäume decken sich weitgehend, aber der wesentliche Unterschied liegt in der Semantik der Dienstleistungen. Natürlich könnte man versuchen, für die

Semantik leere Datenobjekte zu definieren, deren An- oder Abwesenheit im Ein- oder Ausgaberaum eine Dienstleistung charakterisiert. Alternativ könnte man auch die Ontologie so feingranular machen, dass sich Ein- und Ausgaberäume deutlicher voneinander unterscheiden. Doch beide Ansätze führen zu einer Vermischung der Ebenen und haben den Beigeschmack, die Notwendigkeit einer neuen Stufe durch immer aufwendigere Konstrukte aus herkömmlichen Mitteln zu verzögern. Die Semantik einer Dienstleistung hat mit ihren Ein- und Ausgaberäumen nur sehr mittelbar tun. Es ist sinnvoller, sie getrennt zu modellieren und zu beschreiben.

3.3 Transaktionsprotokoll und Dienstleistungszustände

In diesem Kapitel wird beschrieben, welche Nachrichtentypen innerhalb der SAL verwendet werden, mit welchen Zustandsänderungen sie einhergehen und wie sie aufeinanderfolgen (müssen). Diesem Protokoll kommt eine zentrale Bedeutung zu: Die SAL muss gewährleisten, dass die Auftragsabwicklung geordnet abläuft, dass es keine undefinierten Zwischenzustände gibt und dass Auftraggeber und Auftragnehmer ein übereinstimmendes Bild von dem gerade gültigen Zustand haben. Außerdem verhindert die SAL, dass Kommandos, die im gerade gültigen Zustand unzulässig sind, Schaden anrichten. Andererseits ist das Transaktionsprotokoll eine interne Angelegenheit der SAL. Es verbirgt sich hinter der Handhabung von Dienstleistungen, ermöglicht aber erst deren Vermittlung und Abwicklung.

Wie in Kapitel 3.2.3 beschrieben, gibt es viele Nachrichten, deren Erfolg oder Mißerfolg sofort zurückgemeldet werden muss. Dazu gehört die Auftragsvergabe, die bestätigt oder abgelehnt wird, aber auch das Kommando zum Anhalten, Wiederaufnehmen und Abbrechen von Dienstleistungen. Nun könnte man für solche Erfolgsrückmeldungen eigene Rücknachrichten schicken; die von KQML dafür vorgesehenen Nachrichtentypen reichen allerdings für die Anforderungen der SAL nicht aus. Die Verwendung einer höher strukturierten Middleware ermöglicht es jedoch, dass jede Nachricht einen Rückgabewert hat, welcher den Erfolg oder Misserfolg signalisiert. Damit fallen eine Reihe von Kurzzeit-Synchronisationsproblemen weg. Auf der Implementierungsebene bedeutet dies, dass jeder Agent durch die SAL über ein Objekt verfügt, das folgende Methode implementiert:

```
unsigned short receiveMessage (CSAL_Message message);
```

Eine CSAL_Message ist dabei wie eine KQML-Nachricht aufgebaut; das Feld content ist jedoch genauer strukturiert, was im folgenden Kapitel beschreiben wird, und zu den Feldern sender und receiver tritt eine logische Adresse, die sich von der physikalischen (CORBA-)Adresse durch ihre lange Gültigkeit auszeichnet: Wenn der Agent beendet und neu gestartet wird, ist die logische Adresse immer noch gültig, während sich die physikalische Adresse, in der unter anderem die Prozess-ID enthalten ist, geändert hat.

Über den Rückgabewert der Methode wird der Erfolg oder Mißerfolg der Nachricht detailliert zurückgemeldet. Dabei werden Übertragungsprobleme (z.B. dass der angesprochene Agent momentan nicht verfügbar ist oder nicht lokalisiert werden kann) ebenso dokumentiert wie ein Mißerfolg aus inhaltlichen Gründen (z.B. dass der Dienstleistungserbringer in einem Zustand ist, der einen Abbruch nicht erlaubt oder sinnvoll erscheinen lässt). Durch die Rückgabewerte der Nachrichten wird allerdings auch eine Anforderung an die Middleware gestellt. Es ist nicht mehr möglich, einfache Mechanismen wie z.B. Sockets zu verwenden, da sie bloß in einer Richtung arbeiten. Andererseits ist die direkte Rückmeldung für die genannten Nachrichten so wichtig,

dass ihre Umsetzung im Bereich der Nachrichtenübertragung auch bei Sockets sinnvoll wäre. Der Rest der SAL ist dann von dieser Aufgabe unabhängig und kann auf einem höheren Niveau operieren.

Eine vollständige Übersicht der von der SAL verwendeten Nachrichtentypen wird in Tabelle 1 gegeben. Betrachten wir zunächst den Standardfall, die einfache Auftragsvergabe. Sie wird in Abbildung 10 links dargestellt. Der Auftraggeber verschickt eine Nachricht vom Typ Aa. Wenn diese Nachricht mit einem Misserfolg zurückkommt, folgt vom Auftragnehmer keine weitere Nachricht. Wenn sie jedoch Erfolg hat, dann kommt irgendwann genau eine Nachricht mit den Ergebnissen zurück (Typ B) oder[11] genau eine Nachricht, die Ergebnisse unter Vorbehalt enthält oder das Scheitern der Dienstleistung (Typ D) bekanntgibt. Ob beim Nachrichtentyp Aa die Performative ask-all oder ask-one ist, ist in den meisten Fällen irrelevant. Es gibt jedoch Dienstleistungen, die mehrere gleichartige Informationen liefern können. Diese entscheiden auf Grund der Performativen, ob sie bloß eine Information liefern oder alle.

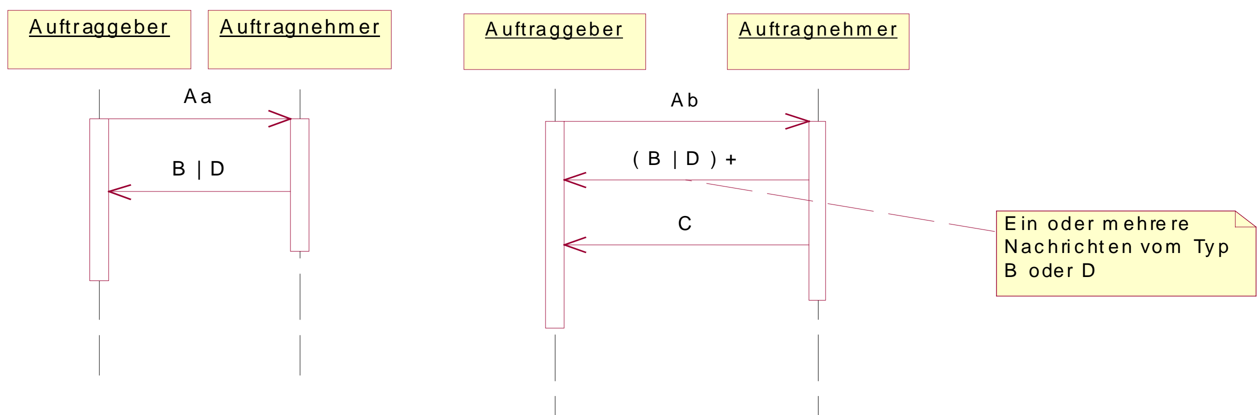


Abbildung 10: Nachrichtenfolgen bei der Verwendung von ask-all und ask-one (links) bzw. bei stream-all (rechts). Nachrichtentypen nach Tabelle 1, Darstellung nach UML ([24] [27]). Jeder Pfeil symbolisisert das Senden einer Nachricht.

Die SAL bietet in Anlehnung an KQML auch die Möglichkeit, einen Auftrag im sogenannten Stream-Modus zu vergeben. Dies ist in der Abbildung rechts dargestellt. Die Auftragsvergabe erfolgt in diesem Fall mit einer Nachricht vom Typ Ab. Die Ergebnisse werden geschickt, sobald sie auf der Auftragnehmerseite verfügbar sind. Dazu dienen Nachrichten vom Typ B (bei Erfolg) oder D (bei Miss- oder Teilerfolg), von denen mindestens eine geschickt werden muss. Ist die Dienstleistung erbracht und sind alle Ergebnisse mit B- oder D-Nachrichten verschickt, dann informiert der Auftragnehmer den Auftraggeber über das Ende der Dienstleistung mit einer end-of-stream-Nachricht (Typ C). Nachrichtentyp C trägt also keinen eigenen Inhalt.

Die Auftragsvergabe im Stream-Modus ist komplizierter zu handhaben als im einfachen Fall. Der Auftraggeber vergibt einen Auftrag in diesem Modus, weil er mit den Informationen schon vor dem Ende der Dienstleistung etwas anfangen möchte. Konsequenterweise muss er dafür Abläufe

[11] Dieses *oder* ist exklusiv.

in einer Call-Back-Methode implementieren. Dies geschieht durch die Verwendung der Polymorphie: Die SAL definiert in einer Basisklasse eine abstrakte Methode. Der Auftraggeber gibt in einer abgeleiteten Klasse für die abstrakte Methode eine konkrete Implementierung. Wenn nun Ergebnisse vorliegen, dann ruft die SAL über die Polymorphie diese Methode auf. In der Implementierung der Methode legt der Auftraggeber fest, was mit den Informationen geschehen soll.

Typ	**Zweck**	**Umsetzung in KQML**	**Erläuterung**
Aa	Auftragsvergabe im normalen Modus	*performative* ::= ask-all | ask-one *in-reply-to* ::= <Name der Dienstleistung>	Ergebnisse werden am Ende der Dienstleistung geschickt. Der Dienstleistungserbringer kann auf Grund der Performative entscheiden, wie viele Ergebnisse er schicken soll.
Ab	Auftragsvergabe im Stream-Modus	*performative* ::= stream-all *in-reply-to* ::= <Name der Dienstleistung>	Vorliegende Ergebnisse werden sofort geschickt
B	Zurückschicken von Ergebnissen	*performative* ::= tell *in-reply-to* ::= <*reply-with* der Auftragsvergabe>	
C	Melden, dass ein Auftrag abgeschlossen ist	*performative* ::= eos *in-reply-to* ::= <*reply-with* der Auftragsvergabe>	Nur, falls die Auftragsvergabe durch Typ Ab erfolgt ist.
D	Melden von Warnungen und Fehlern	*performative* ::= untell | error | sorry | deny | terminated *in-reply-to* ::= <*reply-with* der Auftragsvergabe>	Die Performative richtet sich danach, ob Ergebnisse nicht oder nur unvollständig vorhanden sind bzw. eine Warnung aufgetreten ist (tell), ob ein prinzipieller Fehler aufgetreten ist (error), ob ein temporärer Fehler aufgetreten ist und ein späterer Versuch Erfolg haben könnte (sorry) oder ob die angefragten Ergebnisse prinzipiell nicht geliefert werden können. Wurde die Dienstleistung durch ein Terminate beendet (Typ G), dann ist die Performative terminated.
E	Anhalten einer Dienst-leistung	*in-reply-t*o ::= suspend@<*reply-with* der Auftragsvergabe>	
F	Wiederaufnahme einer angehaltenen Dienstleistung	*in-reply-to* ::= resume@<*reply-with* der Auftragsvergabe>	
G	Abbrechen einer Dienstleistung	*in-reply-to* ::= terminate@<*reply-with* der Auftragsvergabe>	
H	Statusabfrage einer Dienstleistung	*in-reply-to* ::= status@<*reply-with* der Auftragsvergabe>	Eine Statusabfrage hat keine Auswirkungen auf den Zustand. Sie ist ein Service für den Dienstleistungsnachfrager.
I	Aufforderung zum Nachsenden von Ergebnissen, die noch nicht angekommen sind	*in-reply-to* ::= resendResults@<*reply-with* der Auftragsvergabe>	Dieser Nachrichtentyp wird intern verwendet, wenn im Zuge des Wiederaufsetzens nach einem Absturz die Ergebnisse zwischen Dienstleistungserbringer und Dienstleistungsnachfrager abgeglichen werden müssen.

Tabelle 1: Von der SAL verwendete Nachrichtentypen

Auf der Auftragnehmerseite ist die Sache einfacher: Die Implementierung der Dienstleistung gibt der SAL bekannt, welche Informationen als Ergebnisse zurückzusenden sind. Die SAL entscheidet dann, ob diese am Ende gesendet werden sollen (einfacher Fall, Auftragsvergabe vom Typ Aa) oder sofort (Typ Ab). Somit braucht der Dienstleistungserbringer nichts davon zu wissen, in welchem Modus die Dienstleistung angefragt wurde.

Zur Veranschaulichung der Nachrichtenfolgen und Zustandsübergänge bei der Auftragsvergabe, Suspend, Resume und Terminate dient Abbildung 11. In dieser Abbildung stehen durchgezogene Pfeile für Nachrichten. Der jeweilige Nachrichtentyp ist beim Pfeil angegeben. Dabei gehen alle Nachrichten vom Auftraggeber zum Auftragnehmer, mit Ausnahme der Nachrichten vom Typ B, C und D. Wenn eine Nachricht vom Auftraggeber einen Erfolg zurückmeldet, dann ist damit ein Zustandsübergang verbunden.

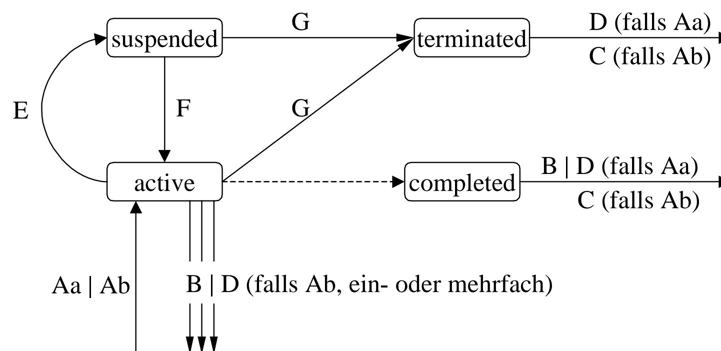


Abbildung 11: Zustandsübergänge und Nachrichtenfolgen. Nachrichtentypen nach Tab. 1.

Wenn also ein Auftragnehmer einen Job annimmt, dann geht dieser in den Zustand active über (links unten im Bild, Zustandsübergang auf Grund einer Auftragsvergabe durch eine Nachricht vom Typ Aa oder Ab). In diesem Zustand können Ergebnisse geschickt werden (Nachrichten vom Typ B oder D, falls der Auftrag im Stream-Modus vergeben wurde). Der Zustand active kann auf drei Arten verlassen werden:

- Durch eine erfolgreiche Suspend-Nachricht (Typ E). Der neue Zustand ist dann suspended.
- Durch eine erfolgreiche Terminate-Nachricht (Typ G). Der neue Zustand ist terminated.
- Auf natürliche Weise, d.h. wenn die Dienstleistung erbracht wurde und beendet ist. In diesem Fall ist der neue Zustand completed. Dieser Zustandsübergang ist der einzige, der nicht durch eine Nachricht hervorgerufen wird. In der Abbildung ist dies durch den gestrichelten Pfeil dargestellt.

Der Zustand suspended kann entweder durch eine Resume-Nachricht verlassen werden (Typ F; neuer Zustand active) oder durch eine Terminate-Nachricht (Typ G; neuer Zustand terminated). Die Anzahl der Suspend-Resume-Zyklen ist nicht begrenzt.

Die Zustände terminated und completed sind von relativ kurzer Dauer. Falls der Auftrag im Normalmodus vergeben wurde (also durch eine Nachricht vom Typ Aa), werden die Ergebnisse

mit einer B- oder D-Nachricht zurückgeschickt. Falls der Auftrag im Stream-Modus (Typ Ab) vergeben wurde, dann wird dem Auftraggeber mit einer C-Nachricht (end-of-stream) signalisiert, dass die Abarbeitung der Dienstleistung beendet ist. Danach wird das Job-Objekt beim Auftragnehmer zerstört. Im terminate-Fall schickt die SAL keine B-Nachricht, sondern eine D-Nachricht mit der Performative terminated.

Bisher sind wir immer von erfolgreichen Nachrichten ausgegangen. Die Reaktionen auf nicht erfolgreiche Nachrichten ist einfacher als der erste Blick vermuten lässt: Die meisten Nachrichten gehen vom Auftraggeber zum Auftragnehmer. Kann eine solche Nachricht nicht zugestellt oder erfolgreich verarbeitet werden, dann ist damit keine Zustandsveränderung verbunden. Dies kann entweder die Auftragsvergabe an sich betreffen, oder aber ein Suspend, Resume oder Terminate. Die weiteren Reaktionen obliegen dem Auftraggeber. Wenn jedoch der Auftragnehmer eine Nachricht nicht schicken kann, dann wird sie solange aufgehoben, bis der Auftraggeber sie mit einer Resend-Results-Nachricht (Typ I) abfragt. Hat der Auftraggeber allerdings durch ein vorangegangenes Terminate signalisiert, dass er an den Ergebnissen nicht mehr interessiert ist, dann wird ein erfolgloser Zustellversuch nicht weiter ernst genommen, und die Daten werden gelöscht. Im Zusammenhang mit dem Wiederaufsetzen von Agenten in Kapitel 3.8 wird auf das erneute Schicken von Ergebnissen etwas genauer eingegangen.

Zur Vermeidung von Synchronisationsproblemen bei Zuständen und Nachrichtenübertragungen verwendet die SAL folgende Schutzmechanismen: Die Zustandsverwaltung liegt allein beim Auftragnehmer. Die SAL hält und verwendet auf Auftraggeberseite keine Zustandsinformationen. Damit ist ein wichtiges Synchronisationsproblem vermieden. Andererseits wird durch die Verwendung sogenannter Mutex-Variablen[12] verhindert, dass der Aufterraggeber parallel einen Auftrag senden, anhalten, wieder aufnehmen oder abbrechen kann. Es ist also beispielsweise unmöglich, dass ein Auftrag angehalten wird, noch bevor der Dienstleistungserbringer den Erfolg der Auftragsvergabe zurückgemeldet hat. So können die sogenannten Race-Conditions nicht auftreten, und die Zustandsverwaltung wird sicher.

Damit soll der wichtige und zentrale, aber SAL-interne Bereich der Jobzustandsverwaltung und des Transaktionsprotokolls abgeschlossen werden. Mit der folgenden Informationsübertragung wird ein Bereich dargestellt, mit dem Benutzer der SAL ständig konfrontiert sind.

3.4 Übertragung von Informationen

Was den Inhalt einer Nachricht betrifft, die Übertragung von Informationen, ist KQML völlig offen. Das hat den Vorteil, dass sich KQML grundsätzlich mit verschiedenen Informations-Kodierungsmöglichkeiten kombinieren läßt. Auf der anderen Seite ist es unabdingbar, in einem Agentensystem die Kodierung der Informationen zu standardisieren, damit die Informationen

---

[12] Mutex steht für "mutual exclusion", also die gegenseitige Exklusivität. Wenn Thread A eine Mutex-Variable beansprucht, kann man sicher sein, dass ein anderer Thread diese Mutex-Variable nicht beanspruchen kann und vom Betriebssystem solange blockiert wird, bis A die Mutex-Variable freigibt. Code, der nicht gleichzeitig durchlaufen werden darf (wie hier Auftragsvergabe, Anhalten, Wiederaufnehmen und Abbrechen) wird geschützt, indem jeweils am Anfang das Mutex beansprucht und am Ende wieder freigegeben wird.

tatsächlich ausgetauscht und weiterverarbeitet werden können. Diese Aufgabe der Standardisierung wird von KQML nicht wahrgenommen. Dies könnte daran liegen, dass KQML ein Vorschlag für einen Nachrichtenstandard ist, und eine Festlegung auf eine Art der Informationskodierung würde die Chancen für KQML reduzieren, als allgemeiner Standard akzeptiert zu werden.

Wie dem auch sei, im Logischen Klienten ist es notwendig, die Informationskodierung festzulegen. Eine gute Festlegung bietet allen Beteiligten eine enorme Hilfe und Vereinfachung, während eine schlechte zu vermehrtem und dupliziertem Aufwand führt – zur Implementierungszeit ebenso wie zur Laufzeit. Bei der Wahl einer geeigneten Informationskodierung gab es folgende Ausgangspunkte:

- Die in KQML als Beispiele immer wieder auftretende Form (vgl. Abbildung 9) als Zeichenkette mit Klammern zur Strukturierung der Objekte mag zwar für interpretierte Sprachen aus dem Bereich der Künstlichen Intelligenz, der logischen Programmierung und der Expertensysteme geeignet sein, aber da diese Zeichenketten jedesmal geparst werden müssen, ist diese Variante zu langsam.

- Informationen werden in Form von Objekten ausgetauscht. Damit ergibt sich ein nahtloser Übergang zur objektorientierten Ontologie und zu den Implementierungen der einzelnen Dienste, die in der objektorientierten Sprache C++ geschrieben sind.

- Da die Nachrichtenübertragung auf CORBA basiert, ist es naheliegend, bei der Übertragung der Informationsobjekte ebenfalls auf Mechanismen von CORBA zurückzugreifen. Dabei ist es jedoch wichtig, die Abhängigkeit von CORBA so gering wie möglich zu halten, um die Middleware gegebenenfalls austauschen zu können.

- Die SAL soll von den zu übertragenden Datenobjekten unabhängig sein. Sie soll diese zwar verschicken, aber über die interne Struktur der Datenobjekte nichts wissen müssen. Programmiertechnisch bedeutet dies, dass die SAL kompiliert werden können muss, ohne dass die Klassendefinitionen der Informationsobjekte bekannt sind.

CORBA ist ein objektorientierter Standard. Er erlaubt den Zugriff auf verteilte Objekte und deren Methoden. Was solche entfernten Methodenzugriffe betrifft, unterstützt CORBA die wichtigen Konzepte der Vererbung und Polymorphie. Vererbung tritt auch in der Ontologie auf und ist dort ein essentielles Strukturierungsmittel. Polymorphie ist in der Ontologie jedoch nicht wichtig – die Informationsobjekte enthalten Daten, aber sie bieten keine Methoden an. Auch was die Aufgabe der Übertragung von Informationsobjekten durch die SAL betrifft, passt CORBA nicht hundertprozentig: Es geht hier nicht darum, den Zugriff auf verteilte, aber ortskonstante Objekte zu ermöglichen, sondern darum, Objekte zwischen verschiedenen Agenten (Orten) zu verschicken. Dieser Unterschied hat weitreichende Konsequenzen.

Die SAL verwendet CORBA zur Übertragung von Nachrichten. Jeder Agent verfügt über ein CORBA-Objekt, das Nachrichten empfängt und auf das von außerhalb, also von anderen Agenten, zugegriffen werden kann. Gleichzeitig stellt die SAL Mittel bereit, um solche Nachrichten zu schicken. Die Signatur der Methode zum Empfangen von Nachrichten ist:

```
unsigned short receiveMessage (CSAL_Message message);
```

Die Klasse CSAL_Message ist weiter strukturiert, sie enthält Attribute zum Bestimmen des Empfängers, des Senders, des Bezugs, der Performative und eine Liste von Datenobjekten. Der entscheidende Punkt hierbei ist, dass CORBA bei dieser Liste von Datenobjekten Vererbung *nicht* unterstützt. Objekte, die als Parameter einer CORBA-Methode übergeben werden, müssen genau von der Klasse sein, die in der Schnittstelle dieser Methode angegeben ist. Es ist hier nicht möglich, dass ein Objekt einer spezialisierteren Klasse anstelle eines Objektes einer allgemeineren Basisklasse eingesetzt wird. Diese Einschränkung von CORBA hat gute Gründe: Die Datenobjekte werden von CORBA beim Sender in eine Kette von Bytes verwandelt, als solche zum Empfänger geschickt und dort wieder auf Grund der Bytekette rekonstruiert. Damit das funktioniert, ist es unerläßlich, dass beide Seiten genau wissen, was in dieser Kette steht. Diese Festlegung muss bereits zur Kompilierungs-Zeit getroffen werden, wenn auf aufwendige Laufzeit-Interpretationen wie bei Sprachen der Künstlichen Intelligenz verzichtet werden soll.

Nun musste aber die SAL von der Klassenzugehörigkeit und Anzahl der ausgetauschten Datenobjekte unabhängig sein. Zusätzlich verhindert die dabei auftretende große Anzahl an Kombinationsmöglichkeiten von Datenobjekten, dass die SAL zur Erstellungszeit brauchbare Nachrichten festlegt.

Aus diesem Dilemma führen zwei Wege. Der erste Weg überträgt nicht die Datenobjekte an sich, sondern bloß Referenzen auf sie. Deshalb müssen die Datenobjekte nicht in Byteketten umgewandelt und rekonstruiert werden. Dieser Weg bedeutet in herkömmlichen Programmiersprachen die Übertragung von Zeigern. Wenn man einen Zeiger auf eine Basisklasse erhält, kann man zunächst nur auf die Attribute der Basisklasse zugreifen, auch wenn sich dahinter ein Objekt einer abgeleiteten Klasse mit weiteren Attributen verbirgt. Hat man jedoch die Information, welches Objekt sich tatsächlich hinter einem Zeiger verbirgt, kann man zur Laufzeit versuchen, den Typ des Zeigers von der Basisklasse auf die abgeleitete Klasse zu "casten"[13]. Wenn dies erfolgreich ist und der angenommene Typ mit dem tatsächlichen Typ übereinstimmt, kann man so auch auf die zusätzlichen Attribute der abgeleiteten Klasse zugreifen. Dieser Vorgang würde auch mit CORBA-Objekten funktionieren. Agent A erzeugt ein Datenobjekt, und zwar als ein CORBA-Objekt, auf das entfernt zugegriffen werden kann. Er gibt Agenten B die Adresse und den Typ dieses Objekts bekannt. Agent B kann nun versuchen, auf dieses Objekt zuzugreifen. Dieser Weg wird zum Beispiel vom "Integrated Collaborative Decision Making"-Framework beschritten, hat aber zwei Nachteile: Es ist unklar, wann ein Objekt nicht mehr benötigt wird und gelöscht werden kann, und die Anzahl der entfernten und teuren Objektzugriffe steigt rapide an.

Der zweite Weg wird von der SAL beschritten. Er stützt sich dabei auf die Tatsache, dass der Empfänger einer Nachricht weiß, welche Datenobjekte darin enthalten sein sollen. Dieses Wissen kommt aus der Offenlegung der Dienstleistungsschnittstellen: Der Dienstleistungsanbieter definiert den benötigten Input, und der Auftraggeber weiß aus der Schnittstellendefinition, welche Ergebnisse er zu erwarten hat. Programmiertechnisch bedeutet dies, dass beide Kommunikationspartner über die Klassendefinitionen der Ontologie-Objekte verfügen. Während also die SAL unabhängig von der Art der Informationsobjekte ist, sind die

---

[13] cast (engl.) bedeutet eigentlich umgießen. In C++ werden diese Zeiger-Reinterpretationen cast-Operationen genannt.

Kommunikationspartner dies nicht. Mehr noch – deren Abhängigkeit von den entsprechenden Informationsklassen ist kein Nachteil, sondern erforderlich.

Die Datenobjekte werden von der SAL beim Verschicken ohne Informationsverlust in eine allgemeine, generische Form überführt, ein sogenanntes serialisiertes Datenobjekt. Die SAL überträgt mit jeder Nachricht eine beliebig lange Liste solcher serialisierter Datenobjekte. Der Empfänger erzeugt für das in der Nachricht erwartete Objekt ein leeres Objekt. Die SAL sucht in der Liste der serialisierten Objekte das passende aus und entpackt es. Zur Veranschaulichung dieses Vorgangs und zur Erklärung, dass damit keine Abhängigkeit der SAL von den ausgetauschten Informationsobjekten verbunden ist, soll der Ablauf an einem konkreten Beispiel erläutert werden.

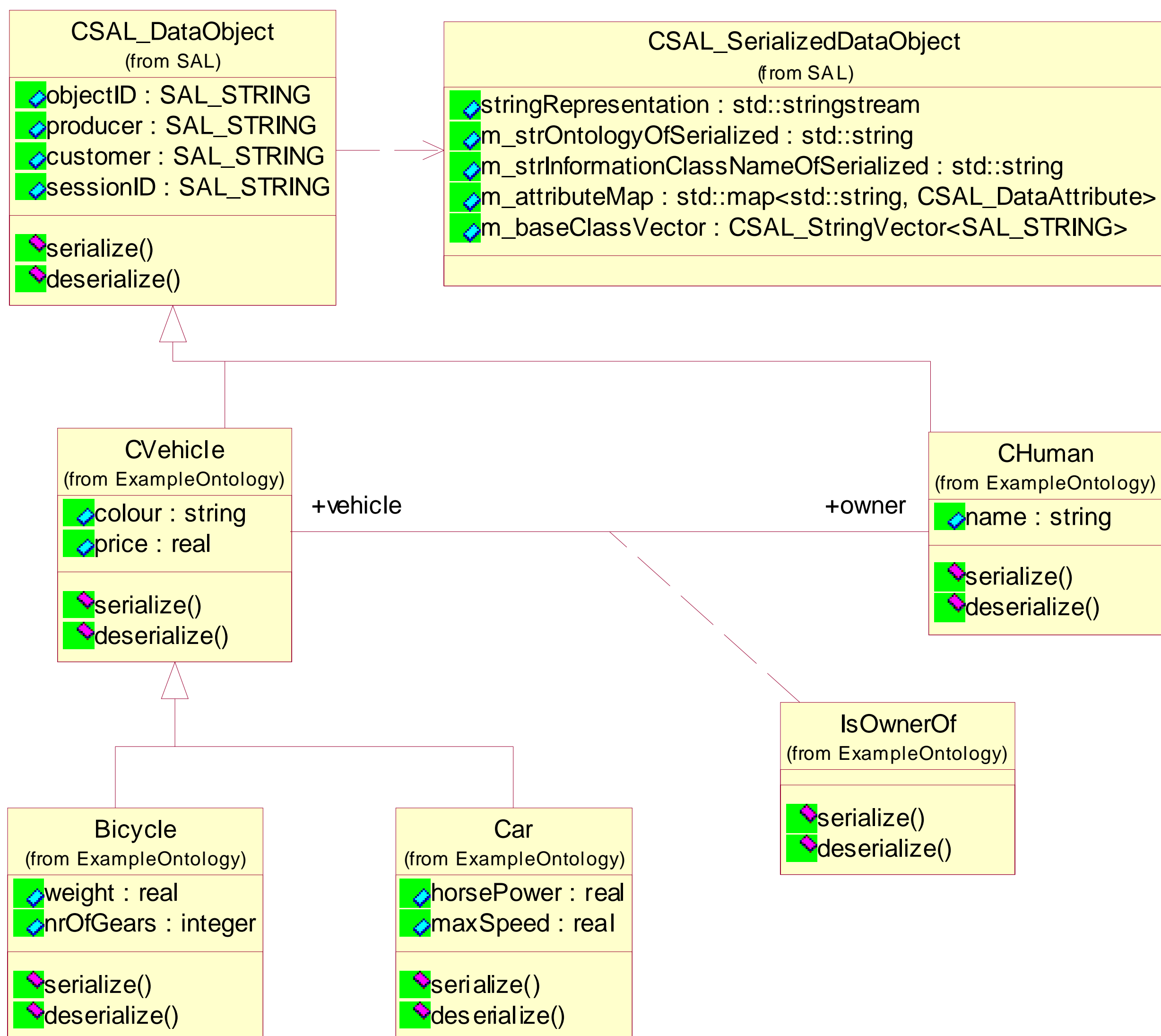


Abbildung 12: Zusammenspiel zwischen Ontologie und SAL beim Serialisieren und Deserialisieren. Darstellung nach UML ([24] [27]).

In Abbildung 12 ist ein Ausschnitt aus einer (fiktiven) Beispielontologie gezeigt. Oben im Bild ist eine abstrakte, von der SAL vorgegebene Klasse, CSAL_DataObject zu sehen. Alle Informationsklassen, deren Objekte über die SAL verschickt werden, müssen von CSAL_DataObject erben. Dadurch erben sie an allgemeinen Attributen eine eindeutige Objektkennung, den Namen des Erzeugers und des Kunden, und die Kennung der Session, innerhalb derer das Datenobjekt erzeugt wurde. Außerdem definiert CSAL_DataObject zwei abstrakte Methoden zum Serialisieren und Deserialisieren von Datenobjekten. Die Schnittstellen dieser Methoden sind in der Abbildung stark vereinfacht. Wesentlich ist, dass beim Serialisieren ein serialisiertes Datenobjekt entsteht. Diese Objekte werden von der SAL übertragen.

Objekte, die von CSAL_DataObject erben, müssen die serialize()- und deserialize()-Methode überschreiben, und zwar so, dass alle Attribute beim Serialisieren in die StringRepresentation des serialisierten Datenobjekts geschrieben werden und beim Deserialisieren von dort rekonstruiert werden können.

Wenn nun ein Auftraggeber einen Auftrag absetzen möchte, muss er die Eingabeobjekte der SAL bekanntgeben. Die SAL weiß nur, dass die Eingabeobjekte von der Klasse CSAL_DataObject abgeleitet sind und deshalb eine serialize()-Methode haben. Diese wird von der SAL aufgerufen. Beim Linken sorgt der Compiler dafür, dass die richtige, konkrete Methode ausgewählt wird. Umgekehrt wird jemand, der z.B. ein CVehicle-Objekt in einer Nachricht erwartet, zuerst ein solches Objekt erzeugen. Dessen Attribute sind noch nicht gesetzt. Sodann ruft der Code dieses Agenten die für diese Klasse gültige deserialize()-Methode auf. Diese sucht in der mit der Nachricht gekommenen Liste von CSAL_SerializedDataObjects, dasjenige heraus, das der gewünschten Ontologie und der gewünschten Klasse entspricht. Die Attribute des Vehicle-Objekts werden dann von der StringRepresentation des serialisierten Datenobjekts rekonstruiert. Auf diese Weise kann die SAL von der Anwendungsontologie vollständig unabhängig bleiben; die Agenten übernehmen die Erzeugung der jeweiligen Objekte, und der Compiler sorgt durch die Polymorphie dafür, dass die richtigen serialize()/deserialize()-Methoden aufgerufen werden.

Es wäre natürlich mühsam und auch fehleranfällig, wenn für jedes Ontologie-Objekt die serialize()- und deserialize()-Methoden händisch kodiert werden müßten. Wie bereits kurz in Kapitel 2.3.9 angedeutet, wurde die Ontologie mit Hilfe des Programms Rational Rose entworfen. Dieses bietet eine grafische Visualisierung der Klassen und ihrer Beziehungen an, kann sehr detaillierte Klasseninformationen aufnehmen und daraus zum Beispiel C++ - Dateien erzeugen. Die von Rational Rose standardmäßig erzeugten C++ - Klassen sind jedoch für die Datenübertragung nicht geeignet. Statt dessen wurde im Rahmen der Implementierung der SAL und ihrer Integration mit der Ontologie auf die in Rational Rose verfügbare Skriptsprache zurück gegriffen und damit ein Skript geschrieben, welches durch das Klassenmodell iteriert und für alle Klassen Definitionen und Implementierungen in C++ erzeugt. Diese erben von CSAL_DataObject und verfügen über durch das Skript generierte serialize()- und deserialize()-Methoden. Dadurch ist gewährleistet, dass die in der Ontologie vorhandenen und mit Rational Rose dargestellten Klassen mit den bei der Implementierung und bei der Übertragung durch die SAL verwendeten Klassen übereinstimmen. Außerdem ist sichergestellt, dass die serialize()- und deserialize()-Methoden gleichmäßig fehlerfrei funktionieren und einwandfrei zusammenarbeiten.

Dieses Skript zur Generierung der C++ - Klassendefinitionen und –implementierungen war der erste praktische Nutzen, der aus der formalen Modellierung der Ontologie gezogen wurde. Später

wurden noch Skripte erstellt, die eine Dokumentation der Ontologie-Klassen im HTML-Format erstellen und Klassendefinitionen für die regelbasierte Sprache CLIPS bereitstellen. Der praktische Nutzen der formalen Modellierung der Ontologie mit Hilfe von grafischen Werkzeugen ist also sehr groß und gleichzeitig um so größer, je vielfältiger die Bereiche sind, in denen Ontologie-Klassen verwendet werden.

Die Implementierungen der serialize- und deserialize-Methoden stützt sich stark auf die vom Compiler direkt zur Verfügung gestellten stream-Operatoren. Dadurch ist eine effiziente und direkt auf dem C++ - Standard aufbauende Serialisierung und Deserialisierung gewährleistet.

Während die bisherige Darstellung sich vor allem auf das Serialisiserungs- und Deserialisierungskonzept konzentriert, sollen im folgenden einige kleinere, aber durchaus interessante und wichtige Aspekte der Übertragung von Datenobjekten betrachtet werden:

Namespaces[14]. In einer Ontologie gibt es normalerweise eine große Anzahl an Klassen, und die Gefahr von Namenskonflikten steigt an. Um dem entgegenzusteuern, verwenden Ontologie und SAL konsequent die sogenannten Namespaces. Dies ist ein relativ spät zum C++ - Standard hinzugekommenes Sprachmittel, und es war nicht immer ganz einfach, bei den generierten Klassen die Schwächen des in der KFÜ vorgeschriebenen Microsoft – C++ - Compilers bei der Handhabung von Namespaces zu umgehen. Diese sind bei einer Kombination von Namespaces, Vererbung über Namespace-Grenzen hinweg und mehrfacher, virtueller Vererbung aufgetreten. Für jede Ontologie und Teilontologie wird ein eigener Namespace eingerichtet.

Klassenzugehörigkeit. Bei der Überprüfung der richtigen Klassenzugehörigkeit greift die SAL auf ein in C++ verfügbares Sprachmittel zurück. Es handelt sich dabei um die Run-Time Type Identification (RTTI), die der Compiler bei den entsprechenden Voreinstellungen zur Verfügung stellt. Dadurch konnten die generierten serialize- und deserialize-Methoden von dieser Aufgabe entlastet werden, und der entstehende Code ist effizienter. Mit RTTI wird beim Serialisieren die Klassen- und Ontologiezugehörigkeit ermittelt und in die entsprechenden Attribute des serialisierten Datenobjekts (m_strOntologyOfSerialized und m_strInformationClassNameOfSerialized) eingetragen. Dabei wird der Name der Ontologie aus den eventuell auch geschachtelten Namespace abgeleitet.

Beziehungen zwischen Objekten. Eine Sonderstellung unter den Informationsobjekten nehmen Beziehungen ein. In der Ontologie wurden Beziehungen zwischen Klassen extern modelliert. Das ist in Abbildung 12 durch die Assoziation zwischen CHuman und CVehicle dargestellt. Weder CHuman noch CVehicle werden durch die mögliche Beziehung zwischen ihnen verändert, aber

---

[14] In der deutschsprachigen Literatur wird für Namespaces oft der Begriff "Namensraum" oder "Namensbereich" verwendet. In dieser Arbeit wird jedoch dem englischen Wort der Vorzug gegeben, da es im Softwarebereich inzwischen zu einem stehenden Begriff geworden ist. Gemeint ist die Einteilung der vom Programmierer verwendeten Klassen, Funktionen, Typen und Variablen in verschiedene Namensbereiche, eben Namespaces, innerhalb derer ein Name für eine Klasse, Funktion, Typ oder eine Variable nur einmal definiert werden kann. Ein- und derselbe Name kann in verschiedenen Namespaces jedoch ohne Probleme verwendet werden.

es gibt eine zusätzliche Klasse – IsOwnerOf –, die erzeugt und mit einem CVehicle und einem CHuman verknüpft werden kann. Die SAL unterstützt den Programmierer dadurch, dass sie ein typsicheres Verknüpfen und die Navigation über Zeiger erlaubt. Die Beziehungspartner können entweder als Referenz oder über ihre Objekt-ID angegeben werden. Beim Deserialisieren stellt die SAL die Beziehung wieder voll her.

Container. Ein Schlüsselkonzept des Logischen Klienten ist die Trennung von Daten und Metadaten (vgl. Kapitel 2.3.5). Dies wird auf Dateiebene durch den Repository-Service ermöglicht. Für kleinere Datenmengen bietet die SAL eine Kontainer-Klasse an, in die beliebige Daten in Tabellenform eingetragen und übertragen werden können. Zusätzlich unterstützt die SAL den Programmierer durch spezielle Methoden, die das Auffinden eines Kontainer-Objekts auf Grund des zugehörigen und über ein Beziehungsobjekt verknüpften Metadatenobjekts.

Bedingungsrelevante Attribute. Bei einem Post-Paket weiß man von außen nicht, was sich innen befindet. Es gibt aber Fälle, wo es sinnvoll ist, außen den Inhalt zu kennzeichnen. Dies kann beispielsweise als Warnung vor zerbrechlichen Gütern gedacht sein oder dem Empfänger die Weiterleitung des Pakets auf Grund seines Inhalts an die richtige Stelle ermöglichen. In Analogie dazu kann es wichtig sein, dass Agenten mehr Informationen über den Inhalt eines speziellen Datenobjekts verarbeiten müssen, ohne das Datenobjekt zu entpacken. Dies ist etwa bei Dateien der Fall, die von einem Simulationsmodul erzeugt werden. Es ist dabei wichtig, ob es sich um Protokolldateien oder um Ergebnisdateien handelt, denn auf Grund dieser Unterscheidung gibt der Workflow-Service diese Objekte an verschiedene, nachgeordnete Dienstleistungen weiter. Der Workflow-Service soll aber als Systemdienst diese Datenobjekte nicht entpacken, d.h. deserialisieren. Als Lösung wurden die bedingungsrelevanten Attribute eingeführt. Es handelt sich dabei um Attribute einer Klasse, die in der Ontologie als bedingungsrelevant gekennzeichnet werden. Beim Serialisieren werden diese Attribute in eine Attributsliste des CSAL_SerializedDataObject kopiert und können dort gelesen werden, und zwar in völliger Analogie zu Informationen, die auf einem Paket außen angebracht sind. Strenggenommen wäre es möglich, auf bedingungsrelevante Attribute zu verzichten und dafür die Ontologie feiner zu strukturieren. Dies hätte aber den Nachteil, dass die Zahl der Klassen erheblich erhöht würde. Deshalb sind bedingungsrelevante Attribute eine sinnvolle Einrichtung der Ontologie. Sie werden durch das Skript in entsprechende Anweisungen in den serialize-Methoden umgesetzt; zur Laufzeit werden bedingungsrelevante Attribute in die m_attributeMap des serialisierten Objekts eingetragen.

Polymorphie. Im Kontext der SAL und der Übertragung von Datenobjekten bedeutet Polymorphie, dass ein spezialisiertes Objekt (etwa von der Klasse Car in Abbildung 12) von einem anderen Agenten als allgemeineres Objekt, etwa als CVehicle entpackt werden kann. Es gibt zwar nur wenige Situationen, wo dergleichen sinnvoll ist, aber gerade in diesen Situationen gibt es oft keine Alternative. Ein Agent soll nicht wissen müssen, als was die von ihm erzeugten Datenobjekte später benutzt werden sollen, und es gibt auch Fälle, in denen ein Datenobjekt später einmal als spezialisiertes Objekt und von einem anderen Agenten als allgemeineres Objekt verwendet werden soll. Die Umsetzung der Polymorphie war nicht ganz einfach, nicht zuletzt deshalb, weil die Attribute mit einer serialize-Methode serialisiert werden, aber mit einer deserialize-Methode einer anderen Klasse deserialisiert werden. Dieses klassenübergreifende Zusammenspiel der serialize()- und deserialize()-Methode ist nur innerhalb von Klassenhierarchien ohne Mehrfachvererbung möglich. Um Fehler zu vermeiden, werden bei der

Generierung des C++ - Codes die Klassenhierarchien geprüft und über das Attribut m_baseClassVector für die Klasse CSAL_SerializedDataObject verfügbar gemacht.

3.5 Interne Struktur der SAL

Im folgenden Kapitel wird die interne Struktur der SAL detailliert beschrieben. Dabei geht es nicht nur um die in den Diagrammen dargestellten Klassenstrukturen und -hierarchien, sondern auch um die Gesichtspunkte, nach denen bestimmte Aufgabenverteilungen vorgenommen wurden. Die Klassendiagramme folgen der grafischen Notation der Unified Modelling Language (UML, [24] [27]), sie sind ein "Abfallprodukt" des Entwurfs der SAL, welcher mit Hilfe des UML-Werkzeugs Rational Rose erfolgte. Die Diagramme zeigen die Struktur der SAL so wie sie ist, aber in den einzelnen Diagrammen wurden aus Gründen der Übersichtlichkeit Details weggelassen.

Kenntnisse der UML und der Objektorientierung werden hier vorausgesetzt. Allerdings ist dieses Kapitel für das Verständnis der weiteren nicht unbedingt erforderlich und kann vom technisch weniger interessierten Leser ohne weiteres überblättert werden.

3.5.1 Übersicht

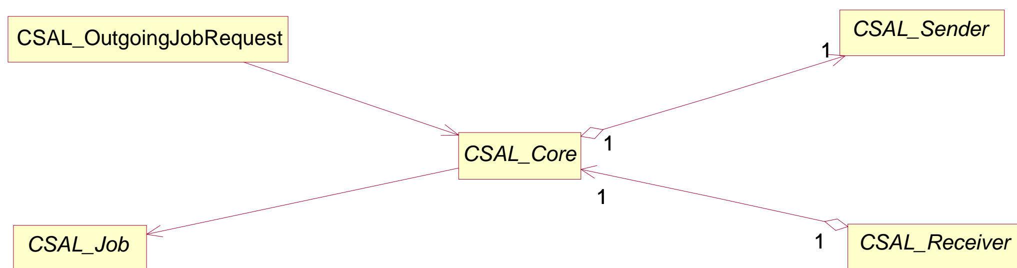


Abbildung 13: Basisklassen der SAL

Die Grundstruktur der SAL ist relativ einfach. Von zentraler Bedeutung ist die Klasse CSAL_Core (Abbildung 13). Sie kennt eine Klasse zum Senden von Nachrichten, CSAL_Sender, und umgekehrt kennt die Klasse zum Empfangen von Nachrichten, CSAL_Receiver den CSAL_Core. Diese drei Klassen gehören zum ständigen Laufzeit-Inventar eines Dienstes, der die SAL verwendet.

Objekte der beiden Klassen links, CSAL_Job und CSAL_OutgoingJobRequest werden zur Laufzeit erzeugt und gelöscht: Wenn der Dienst mit der Erbringung einer Dienstleistung beauftragt wird, wird ein Objekt der Klasse CSAL_Job angelegt und vom CSAL_Core angesteuert. Wenn jedoch der dienstespezifische Code einen Dienstleistungsauftrag vergeben will, erzeugt er einen CSAL_OutgoingJobRequest, welcher mit dem CSAL_Core kommuniziert.

Alle in Abbildung 13 dargestellten Klassen sind abstrakt. Sie stellen die Basisinfrastruktur zur Verfügung. Bei der Erzeugung der SAL bzw. der Erzeugung von Jobs und Requests werden konkrete Klassen angelegt, welche die durch die Basisklassen bereitgestellte Infrastruktur erben und benutzen, aber um spezielle Fähigkeiten und Aufgaben erweitern.

3.5.2 Jobs und Requests

CSAL_Jobs und CSAL_OutgoingJobRequests haben einige Gemeinsamkeiten, was daran liegt, dass ein Request auf Auftraggeberseite einem Job auf der Auftragnehmerseite entspricht. Die SAL koordiniert dabei den Daten- und Kontrollfluss zwischen diesen beiden Klassen.

Die Gemeinsamkeiten von Jobs und Requests wurden in der gemeinsamen Basisklasse CSAL_JobClassesBase vereint (vgl. Abbildung 14). Beide können Datenobjekte verschicken und empfangen: Der Auftraggeber stellt die Inputobjekte für die Dienstleistung bereit. Dazu erzeugt er Ontologie-Objekte und ruft die Methode serialize() der Klasse CSAL_JobClassesBase auf. Dieser Methode wird das Ontologie-Objekt übergeben. Serialize() serialisiert das Ontologie-Objekt und stellt es in die Liste der ausgehenden Datenobjekte (m_outgoingDataList).

Auf der Auftragnehmerseite empfängt die SAL die serialisierten Datenobjekte, erzeugt einen Job und übergibt diesem die serialisierten Datenobjekte, indem sie sie in die Liste der eingehenden Datenobjekte (m_incomingDataList) stellt. Beim Entpacken der serialisierten Datenobjekte wird zunächst ein Objekt der gewünschten Klasse erzeugt und der Methode deserialize() übergeben. Das entsprechende Objekt wird aus der Liste der eingehenden Objekte entnommen und deserialisiert. Die gleichen Abläufe finden beim Zurückschicken der Ergebnisse statt, mit dem Unterschied, dass der Informationsfluss diesmal in die andere Richtung vom Job zum Request verläuft und deshalb die Rollen vertauscht sind: Der Job serialisiert die Objekte in die m_outgoingDataList, und der Request deserialisiert sie aus der m_incomingDataList.

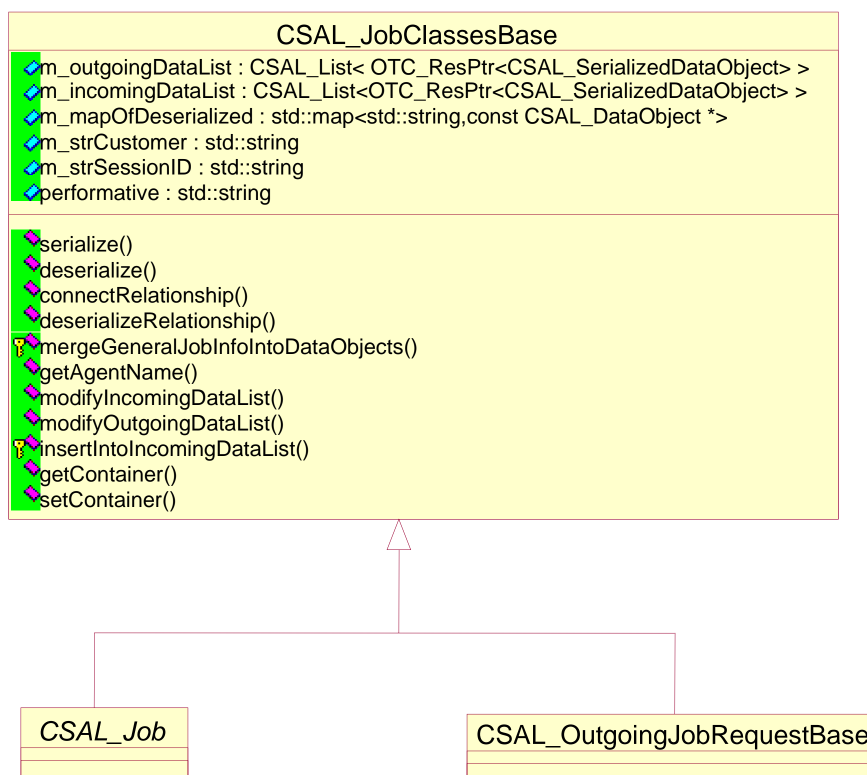


Abbildung 14: Gemeinsamkeiten von Jobs und Requests

Die Implementierung der beiden Listen von serialisierten Datenobjekten stützt sich auf zwei Software-Technologien: Zum einen werden die serialisierten Datenobjekte aus Performanzgründen bei ihrem Transfer durch die SAL zum Job oder zum Request nicht bei jedem Schritt kopiert, sondern als Referenz übergeben. Dies erfordert jedoch auch, dass Datenobjekt gelöscht werden, wenn sie nicht mehr benötigt werden. Im Multi-Threading-Betrieb der SAL hat es sich bewährt, das Löschen der Datenobjekte nach Bedarf durchzuführen, wozu das Prinzip des Reference Countings verwendet wurde [46]. Dabei wurde auf die Implementierung der Softwarebibliothek OSE zurückgegriffen [47], welche gut, gut dokumentiert und frei verfügbar ist. Deshalb verwenden sowohl m_outgoingDataList als auch m_incomingDataList die Klasse OTC_ResPtr (Resource Pointer der OSE, der ein Reference Counting ermöglicht). Die zweite Software-Technologie ist die durch die Standard Template Library [48] zur Verfügung gestellte Listen-Klasse, welche von CSAL_List um die Fähigkeit erweitert wurde, selbst serialisiert und deserialisiert zu werden. Dadurch ist es möglich, auch Jobs und Requests zu serialisieren und zu deserialiseren, was der Rekursivität von KQML-Botschaften entspricht (vgl. Kapitel 3.1).

Darüber hinaus stellt CSAL_JobClassesBase die Namen von Auftraggeber, die Kennung der Session und der für das Senden des Auftrags gewählten Performative zur Verfügung. Außerdem wird das Deserialisieren von Beziehungen dadurch unterstützt, dass auf der Basis der bisher deserialisierten Datenobjekte (für die Referenzen in der m_mapOfDeserialized angelegt werden) beim Deserialisieren von Beziehungen auch die Beziehungen zwischen den deserialisierten Datenobjekten korrekt und als Zeiger wiederhergestellt werden. Außerdem wird das Serialisieren und Deserialisieren von Containern durch eigene Methoden (getContainer() und setContainer()) unterstützt. Schließlich sei noch erwähnt, dass es gelegentlich Dienstleistungen gibt, die serialisierte Datenobjekte nicht deserialisieren, sondern in serialisierter Form zwischenspeichern und weiterschicken. Dazu gehört beispielsweise der Workflow-Service, der Ergebnisobjekte einer Dienstleistung einer anderen als Input mitgibt. Solche Dienstleistungen können an der serialize()/deserialize()-Schnittstelle vorbei mit den Methoden modifyIncomingDataList() und modifyOutgoingDataList() direkt auf die Listen der ein- und ausgehenden Datenobjekte zugreifen.

3.5.3 Jobs im Detail

Die Klasse CSAL_Job ist eine abstrakte Klasse. Sie erbt von CSAL_JobClassesBase die Mechanismen zum Übertragen von Datenobjekten. Gleichzeitig definiert sie eine abstrakte Methode, doJob(). Implementierungen dieser Methode in abgeleiteten Klassen enthalten den spezifischen Code zur Durchführung einer Dienstleistung.

Jobs können die virtuellen Methoden suspend(), resume() und terminate() implementieren, wenn es sinnvoll ist, längerdauernde Dienstleistungen anzuhalten, wiederaufzunehmen oder abzubrechen. Eine Implementierung ist aber nicht verpflichtet, diese Methoden immer erfolgreich zu Ende zu führen: Es kann Situationen geben, wo ein Abbrechen beispielsweise nicht mehr möglich oder sinnvoll ist. Liefern suspend(), oder terminate() einen Misserfolg zurück, verändert die SAL den internen Status der Dienstleistung nicht. Ausgenommen ist hiervon resume(): War ein suspend() erfolgreich, dann muss ein darauffolgendes resume() ebenfalls erfolgreich sein. Die Methode furtherMessage() ist für den Fall angelegt, dass einer laufenden Dienstleistung im Nachhinein weitere Informationen zur Verfügung gestellt werden

müssen. Dieser Fall ist jedoch schwierig, da über diese Hintertür auch Schaden angerichtet werden kann. Ein genaueres Protokoll wäre noch erforderlich. In der KFÜ ist die Methode furtherMessage() nicht erforderlich.

Wenn eine Dienstleistung im Zusammenhang mit der Persistenz nach einem Absturz gerettet wird, wird von der SAL nicht die doJob()-Methode aufgerufen, sondern doRecoveredJob(). Dadurch können im Normal- und Recovery-Betrieb voneinander abweichende Vorgänge getrennt werden.

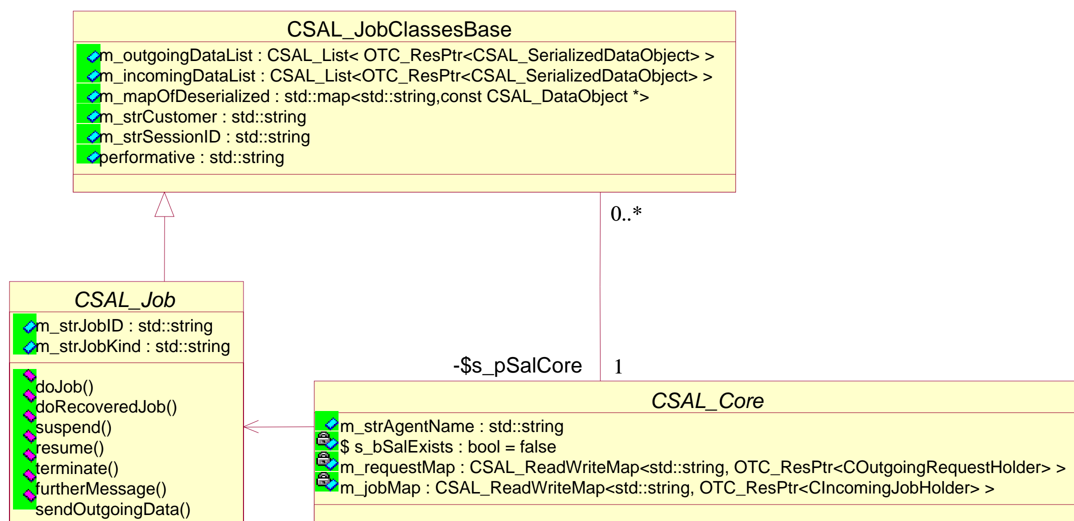


Abbildung 15: CSAL_Job und das Zusammenspiel mit dem CSAL_Core

Das Zusammenspiel von Job und Core ist bidirektional: CSAL_Core veranlaßt die Erzeugung von Jobs, verwaltet diese, stellt die Datenobjekte in die m_incomingDataList und ruft die Methoden doJob(), suspend(), resume(), terminate() oder doRecoveredJob() auf. Wenn die Dienstleistung jedoch Zwischenergebnisse erbracht hat, ruft sie die Methode sendOutgoingData() auf. Falls der Job im stream-all-Modus aufgerufen wurde, werden über den statischen Zeiger von CSAL_JobClassesBase dem Core die Zwischenergebnisse zum Verschicken übergeben.

3.5.4 Requests im Detail

Im Gegensatz zu Jobs, die ihre Arbeit nur in dienstspezifischen Klassen sinnvoll tun können, stellt die SAL voll funktionsfähige Request-Klassen zur Verfügung. Allerdings gibt es auch bei Requests mehrere Varianten innerhalb einer Klassenhierarchie.

Die Klasse CSAL_OutgoingJobRequestBase erbt von CSAL_JobClassesBase die Mechanismen zum Serialisieren und Deserialisieren von Datenobjekten. Nach dem Setzen der Input-Objekte und der Angabe des Empfängers (receiver) sowie der Dienstleistung, die man in Auftrag gibt (inReplyTo) kann man die Methode sendRequest() aufrufen. Diese teilt über den statischen Zeiger s_pSalCore der Basisklasse dem Core der SAL mit, dass ein Request zu schicken ist,

welcher dann die weitere Koordination und das Senden der entsprechenden Botschaften übernimmt.

Ein wichtiger Unterschied zwischen den verschiedenen Request-Varianten ist die Synchronisierung. Der einfachste Fall ist der, dass der Auftraggeber solange warten möchte, bis die Dienstleistung erbracht oder gescheitert ist. Dieser Fall wird durch die Unterklasse CSAL_BlockingRequest abgedeckt. Die andere Möglichkeit besteht darin, dass beim Aufrufen die sendRequest()-Methode nicht blockiert. Dies wird durch die Verwendung der Klasse CSAL_NonBlockingRequest erreicht. In diesem Fall kann der Auftraggeber weiterarbeiten, doch die Synchronisierung mit den Ergebnissen muss anders erfolgen. Die einfachere Möglichkeit wäre das klassische Polling: Der Auftraggeber fragt zyklisch nach, ob die Ergebnisse vorhanden sind (indem er beispielsweise die deserialize()-Methode versucht). Eleganter ist die Verwendung von Call-Back-Mechanismen: Der Auftraggeber erbt von CSAL_NonBlockingRequest und überschreibt die Methode onDataHasArrived() und/oder die Methode onJobIsFinished(). Diese Methoden werden aufgerufen, wenn Ergebnisse zurückgekommen sind bzw. wenn die Dienstleistung beendet ist.

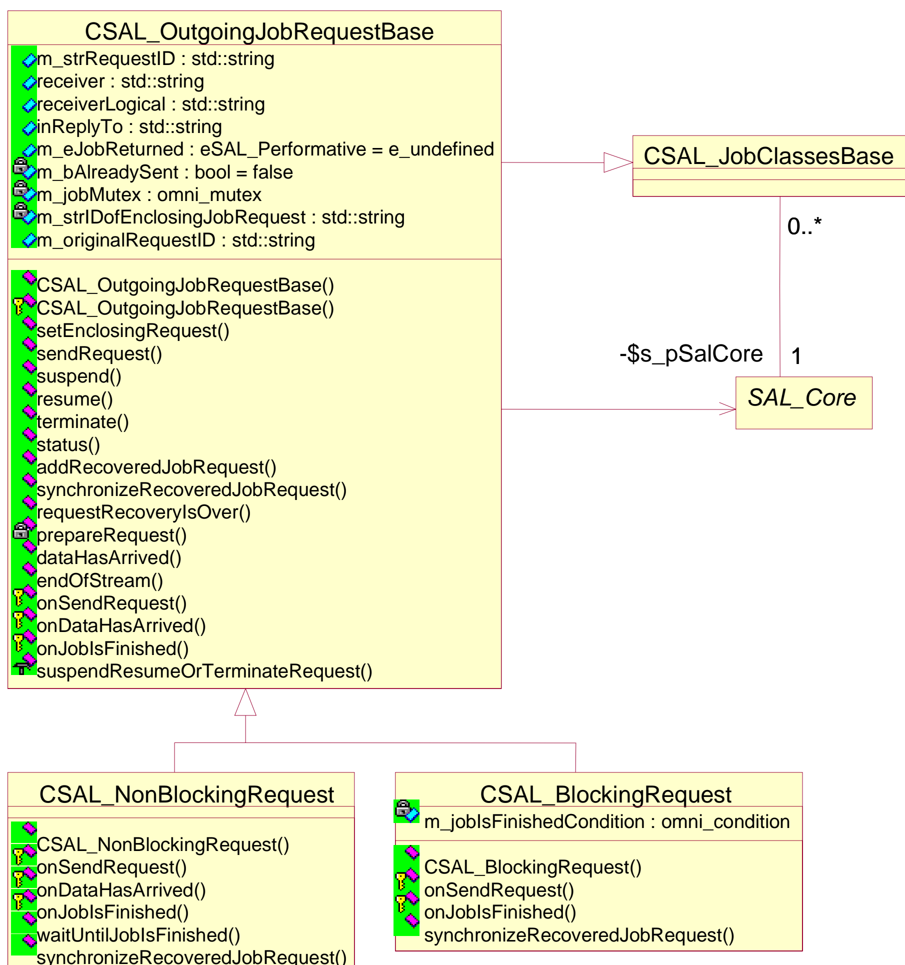


Abbildung 16: Blockierende und nicht blockierende Aufträge

Für den CSAL_Core ist es jedoch unerheblich, welche der beiden Request-Klassen verwendet wurde. Dies wird durch das sogenannte Schablonen-Muster [3] erreicht: Die konkreten Methoden von CSAL_OutgoingJobRequestBase sendRequest(), dataHasArrived() und endOfStream() rufen die virtuellen Methoden onSendRequest(), onDataHasArrived() und onJobIsFinished() auf. Die Implementierungen der virtuellen Methoden sorgen für die korrekte Synchronisation: onSendRequest() der Klasse CSAL_BlockingRequest wartet beispielsweise, bis in einem anderen Thread die Methode onJobIsFinished() aufgerufen wurde. Dadurch wird das blockierende Verhalten von CSAL_BlockingRequest erreicht.

Dieses Entwurfsmuster bedeutet sowohl für den Auftraggeber als auch für den CSAL_Core eine Erleichterung, weil sie nur mit den Methoden sendRequest(), dataHasArrived() und endOfStream()[15] arbeiten müssen und es dabei gleichgültig ist, ob es sich um einen blockierenden Request oder einen nicht-blockierenden Request handelt.

Die Methoden suspend(), resume() und terminate() werden in der Praxis nur von CSAL_NonBlockingRequests aufgerufen.[16] Über sie werden die entsprechenden Methoden des CSAL_Jobs auf der Auftragnehmerseite aufgerufen. Mit der Methode status() kann der Status der Dienstleistung abgefragt werden.

Im Falle des Wiederaufsetzens nach einem Ausfall ist das Wiederverbinden zu abgesetzten Requests nicht ganz trivial, da dabei ja auch gleichzeitig die Wiederherstellung der Synchronisation notwendig ist. Dies wird durch die Methoden addRecoveredJobRequest() und synchronizeRecoveredJobRequest() erreicht.

3.5.5 Erzeugung von Jobs

Bei der Erzeugung von Jobs tritt das Problem auf, dass diese von der SAL bei Eintreffen einer entsprechenden Nachricht veranlaßt wird, die SAL aber kein Wissen über die konkreten Jobs hat. Dieses Problem läßt sich durch das sogenannte Factory-Entwurfsmuster [3] lösen (vgl. Abbildung 17). Das Kernstück des Factory-Entwurfsmusters ist eine Klasse, die Objekte von anderen Klassen erzeugt. Aus der Analogie zum Produktionsvorgang ergibt sich auch der Name des Entwurfsmusters.

Die SAL definiert eine abstrakte Klasse CSAL_JobManager. Diese hat eine abstrakte Methode createJob(). Die Methode nimmt den Namen des zu erzeugenden Jobs entgegen und liefert ein Objekt einer von CSAL_Job abgeleiteten Klasse entgegen.

Beim Erzeugen muss dem CSAL_Core nun ein konkreter JobManager bekanntgegeben werden. In der Abbildung ist dies ein Objekt der Klasse UserDefinedJobManager. Deren Methode

---

[15] Die Methode sendRequest() wird vom Auftraggeber aufgerufen, während die anderen beiden Methoden ausschließlich vom CSAL_Core aufgerufen werden.

[16] Dies liegt daran, dass der grundsätzlich mögliche Aufruf von suspend(), resume() oder terminate() bei einem CSAL_BlockingRequest aus einem anderen Thread erfolgen müsste, da der Thread, der sendRequest() aufgerufen hat, blockiert ist. Dies ist umständlich und wird in der Regel vermieden.

createJob() kann auf Anfrage Jobs der Klasse UserDefinedJobA und UserDefinedJobB erzeugen. Wenn nun über eine Nachricht ein Auftrag zur Durchführung von Dienstleistung UserDefinedJobA hereinkommt, dann ruft die SAL die Methode UserDefinedJobManager::createJob() auf, erhält von dieser ein Objekt der Klasse UserDefinedJobA und ruft dessen Methode doJob() auf.

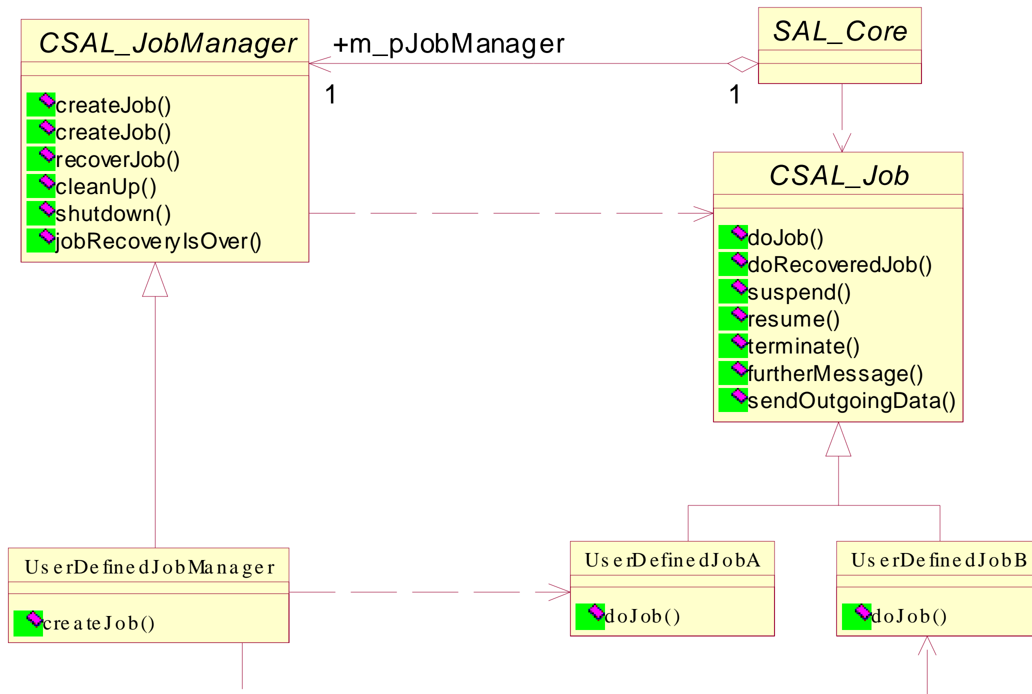


Abbildung 17: Erzeugung von Jobs nach dem Factory-Entwurfsmuster [3]

Auf diese Weise bleibt die SAL unabhängig von den spezifischen Dienstleistungen und Job-Klassen. Anzumerken bleibt noch, dass die Beziehung zwischen angebotenen Dienstleistungen und Jobklassen nicht eineindeutig sein muss. Es ist möglich, dass z.B. eine Jobklasse mehrere Dienstleistungen anbietet. Beim Erzeugen könnte der konkrete JobManager die Weichen des Objekts der Jobklasse so stellen, dass die jeweils gewünschte Dienstleistung erbracht wird. In der Praxis ist dies jedoch der Ausnahmefall.

3.5.6 Nachrichtenübertragung mit CORBA

Eine wichtige Anforderung an die SAL war die Unabhängigkeit von der verwendeten Middleware. Dies wird durch die Verwendung der abstrakten Klasse CSAL_Sender erreicht (vgl. Abbildung 18).

Wenn eine Nachricht zu senden ist, dann wird vom Core die abstrakte Methode sendMessage() aufgerufen. Das bei der Erzeugung bekanntzugebende Objekt einer von CSAL_Sender abgeleiteten Klasse übernimmt dann das Senden der Nachricht. Dazu stellt die SAL eine Klasse

CSAL_OmniOrbSender bereit, die dazu die von den Olivetti & Oracle Research Labs entwickelte und nun von AT & T übernommene CORBA-Implementierung OmniOrb [49] verwendet.

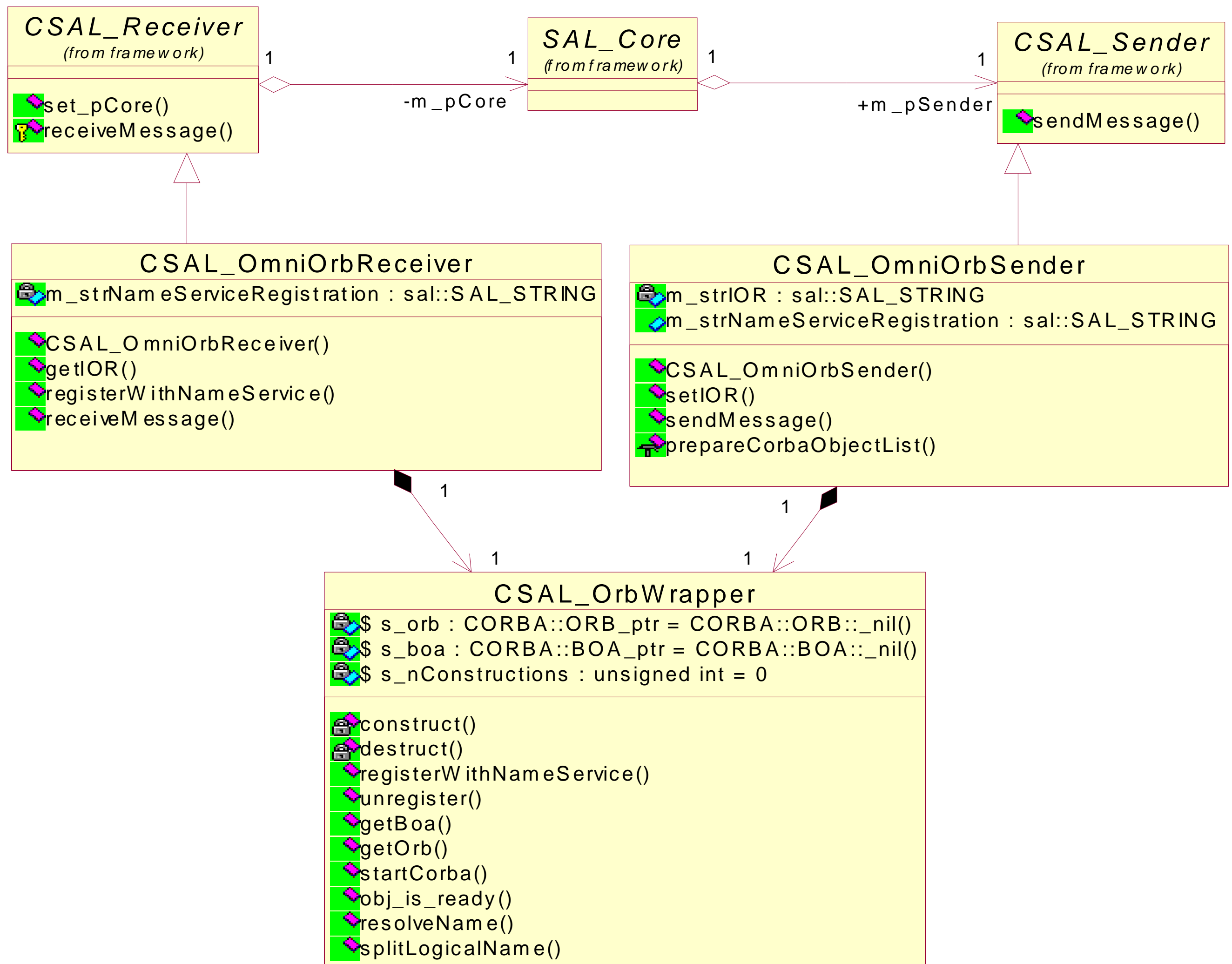


Abbildung 18: Senden und Empfangen von CORBA-Nachrichten

Trotz einiger Ähnlichkeiten ist die Klassenhierarchie CSAL_Receiver anders aufgebaut. Die SAL stellt eine konkrete Basisklasse bereit, die eine Methode receiveMessage() anbietet. Diese Methode nimmt den Kontakt zum CSAL_Core auf und leitet die Nachricht weiter. Die receiveMessage()-Methode der Klasse CSAL_OmniOrbReceiver ist jedoch keine Redefinition der gleichnamigen Methode von CSAL_Receiver. Vielmehr ist sie eine über CORBA ansprechbare Methode, mit der von außen eine Nachricht abgesetzt werden kann. Wenn also ein CSAL_OmniOrbSender eine Nachricht zu senden hat, ruft er über CORBA die

receiveMessage()-Methode der Klasse CSAL_OmniOrbReceiver auf. Diese bereitet die Nachricht auf und ruft die Methode receiveMessage() der Klasse CSAL_Receiver auf. Dass CSAL_OmniOrbReceiver von CSAL_Receiver erbt, ist hier nicht durch Polymorphie begründet, sondern durch ein bequemes Verwenden der receiveMessage()-Methode von CSAL_Receiver. Dies hätte aber auch z.B. durch Aggregation erreicht werden können.

Zur Vermeidung von Abhängigkeiten der konkreten CORBA-Implementierung, also von OmniOrb, wurde zusätzlich die Klasse OrbWrapper eingeführt, die Code enthält, der mit der CORBA-Funktionalität direkt zu tun hat. Dadurch wird sowohl eine Kapselung im Interesse einer leichten Austauschbarkeit erreicht, als auch eine Code-Duplizierung vermieden. Grundsätzlich ist dazu anzumerken, dass bei einer korrekten Implementierung des CORBA-Standards ein Wechsel der CORBA-Implementierung ohne weitere Anpassungen möglich sein müßte. Es kann jedoch in der Praxis vorkommen, dass es sinnvoll ist, auf eine CORBA-Implementierung zu wechseln, die nicht vollständig CORBA-konform ist.

Die geforderte Unabhängigkeit von der Middleware und auch von der konkreten CORBA-Implementierung konnte mit dieser Struktur zu nahezu 100 % erreicht werden. Bei einem Wechsel von CORBA zu einer anderen Middleware müßten die Klassen CSAL_OmniOrbReceiver und CSAL_OmniOrbSender durch andere Klassen ersetzt werden. Außerdem müßte möglicherweise das Adressformat der Empfänger von Nachrichten geändert werden, welches sich an CORBA orientiert. Je nach verwendeter Middleware könnte es auch sein, dass das Hochfahren des Dienstes nach leicht geänderten Gesichtspunkten erfolgen würde.

3.5.7 Entscheidungslogik und Verwaltung

Während in den vorangegangenen Kapiteln isolierte und periphere Aspekte der SAL betrachtet wurden, kommen wir nun zum eigentlichen Kernstück der SAL.[17] In Abbildung 19 wird der Kernbereich dargestellt. Die Klasse CSAL_Core wird durch die davon abgeleitete Klasse CSAL_StandardCore erweitert. Die Aufgabenverteilung zwischen diesen beiden Klassen erfolgte dabei nach folgenden Gesichtspunkten:

Die Klasse CSAL_Core übernimmt die Koordination der anderen Bereiche der SAL: sie kennt und verwaltet die Jobs und Requests, arbeitet mit dem JobManager zusammen, übernimmt Nachrichten vom CSAL_Receiver und schickt Nachrichten and den CSAL_Sender. Die Klasse CSAL_StandardCore hingegen übernimmt die Auswertung der Nachrichten, die Entscheidungslogik, was mit den Nachrichten passieren soll, und das Erstellen von ausgehenden Nachrichten.

Die zahlreichen Methoden der Klasse CSAL_StandardCore mögen auf den ersten Blick verwirrend erscheinen. Klarer wird das Bild, wenn man es in den jeweiligen Abläufen betrachtet.

---

[17] Diese isolierten und peripheren Aspekte sind für das Funktionieren der SAL ebenso wichtig wie der Kern. Sie übernehmen zahlreiche Aufgaben und entlasten ihn erheblich. Gerade in der weitgehenden Entflechtung verschiedener Bereiche, die sich auch in der Unabhängigkeit der vorangegangenen Unterkapitel wiederspiegelt, liegt eine Stärke des Aufbaus der SAL.

Die SAL bietet grundsätzlich die Möglichkeiten zum Hoch- und Herunterfahren durch die Methoden start() und stop().

Falls ein JobRequest geschickt wird, wird die Methode sendRequest() aufgerufen, die die entsprechenden Daten zur Entgegennahme der Ergebnisse speichert, den Request in eine Nachricht umwandelt und über den Core absetzt.

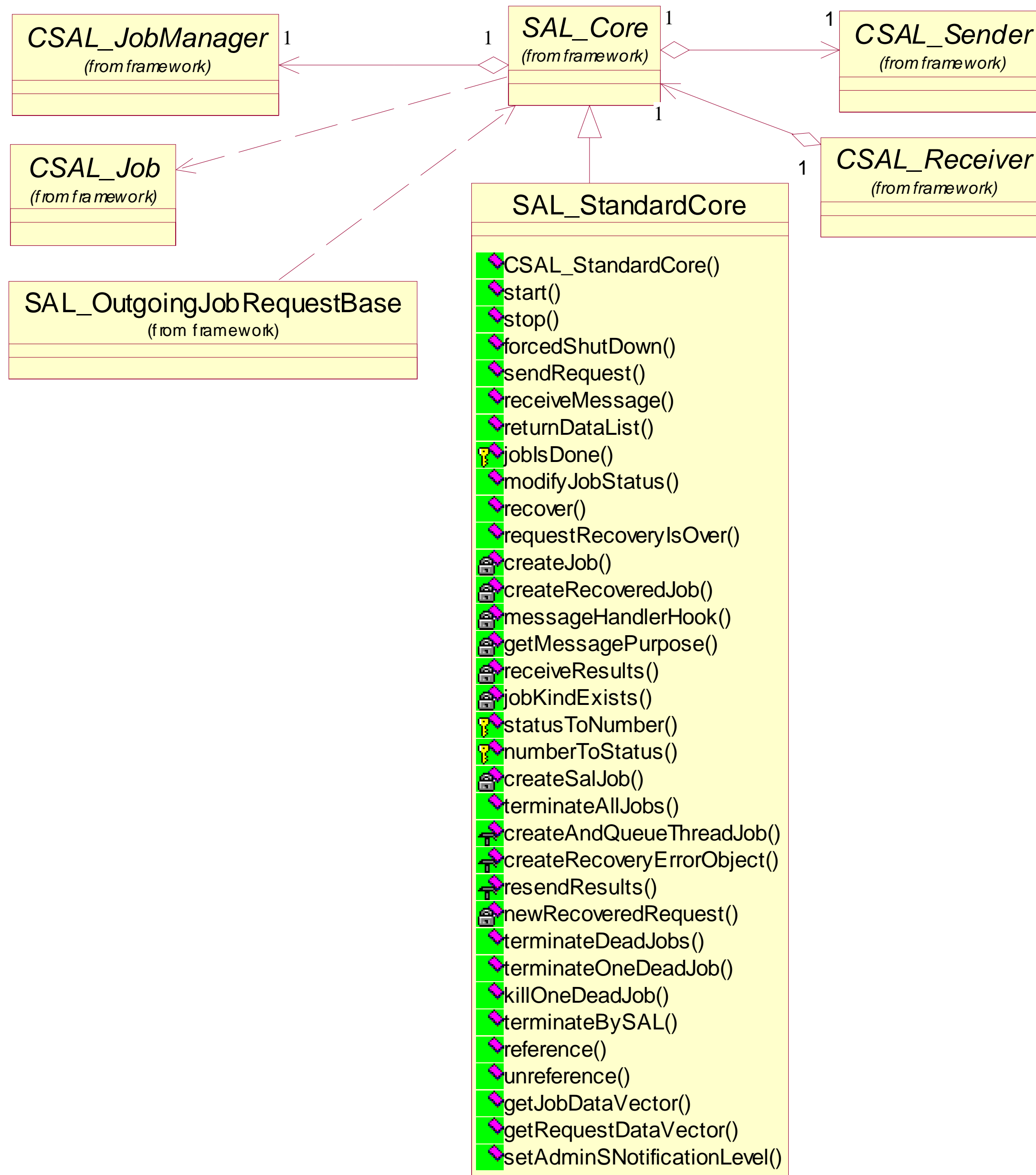


Abbildung 19: Aufgaben des CSAL_StandardCore

Wenn jedoch eine Nachricht hereinkommt, wird die Methode receiveMessage() aufgerufen, welche ihrerseits die Methode getMessagePurpose() aufruft, um herauszufinden, worauf sich die Nachricht bezieht und welche Aktionen damit verbunden sind. Hier gibt es nun mehrere Möglichkeiten:

- Die Nachricht enthält Ergebnisse einer nach außen delegierten Dienstleistung. Dann wird die Methode receiveResults() aufgerufen, die in weiterer Folge dafür sorgt, dass die Ergebnisse dem OutgoingJobRequest zur Verfügung gestellt werden und dort die entsprechenden Methoden dataHasArrived(), onDataHasArrived(), endOfStream(), onJobIsFinished() aufgerufen werden.
- Wenn es sich bei der Nachricht um einen Dienstleistungsauftrag handelt, dann wird im Rahmen der receiveMessage()-Methode zunächst geprüft, ob es sich um einen tatsächlich angebotenen Job handelt. Dies geschieht durch die Methode jobKindExists(). Anschließend wird über die Methode createJob() der CSAL_Core mit der Erzeugung eines konkreten Job-Objekts beauftragt. Da die Jobs in eigenen Threads aus einem Threadpool abgearbeitet werden [5], werden durch die Methode createAndQueueThreadJob() die entsprechenden Vorkehrungen getroffen. Wenn nun in weiterer Folge die Dienstleistung Ergebnisse liefert, wird die Methode returnDataList() aufgerufen, welche im Fall einer Stream-Anfrage eine Ergebnisnachricht erstellt und verschickt. Nach Abschluß der Dienstleistung wird die jobIsDone()-Methode aufgerufen, welche die Abschlußnachricht an den Auftraggeber erstellt und verschickt.
- Wenn die Nachricht eine Statusänderung (Anhalten, Wiederaufnehmen und Abbrechen) verlangt, wird die Methode modifyJobStatus() aufgerufen. Die Abläufe beim Abbrechen von Jobs verlangen darüber hinaus ein Aufräumen der SAL, welches in der Methode terminateIncomingJob() implementiert wird. Als weitere Hilfsfunktionen dienen die Umwandlungen von Job-Status in einen Code und umgekehrt durch statusToNumber() und numberToStatus().
- Wenn die Nachricht der Auftrag einer allgemeinen SAL-Dienstleistung ist (vgl. Kapitel 3.9), dann werden die Methoden createSalJob() und in weiterer Folge ggf. forcedShutDown() und terminateAllJobs() aufgerufen.
- Zur Erweiterung durch abgeleitete Klassen bietet CSAL_StandardCore zusätzlich die Methode messageHandlerHook() an, welche dann aufgerufen wird, wenn CSAL_StandardCore mit einer Nachricht nichts anfangen kann. So können weitere Spezialisierungen der SAL zusätzliche Nachrichtentypen verarbeiten.

Die weiteren Methoden betreffen das Wiederherstellen eines konsistenten Zustands nach einem Systemausfall. Die entsprechenden Abläufe werden in Kapitel 3.8 genauer beschrieben.

Diese Aufteilung zwischen CSAL_Core und CSAL_StandardCore ist nicht zuletzt dadurch motiviert, dass die SAL im Hinblick auf Erweiterbarkeit so konzipiert wurde, dass die Entscheidungslogik beispielsweise durch ein regelbasiertes System übernommen werden könnte. Daraus ergibt sich eine Trennung des Cores in allgemeine Funktionalitäten durch CSAL_Core und in die Entscheidungslogik im CSAL_StandardCore. Diese Trennung hat sich sehr bewährt und bei vielen Entwurfsfragen zu raschen Entscheidungen geführt.

3.5.8 Erzeugung der SAL

Um dem Anwender die Verwendung der SAL zu erleichtern, bietet die SAL eine Builder-Klasse an (siehe Abbildung 20). Diese erzeugt die richtigen Core-, Receiver-, Sender- und CORBA-Klassen und verknüpft sie so, dass das Framework sofort arbeitsbereit ist.

Abbildung 20: Erzeugung der SAL durch eine Builder-Klasse.

Damit sieht ein Programm, das die SAL verwendet, etwa so aus:

```
#include <sal/corbaRealisation/SAL_StandardBuilder.h>
#include "UserDefinedJobManager.h"
void main()
{
        CuserDefinedJobManager jobMgr;
        sal::CSAL_StandardBuilder salBuilder(
              &jobMgr,
              "UserSpecificConfigFile.cfg");
        bool bBlockingMode = true;
        salBuilder.startCorba(bBlockingMode);
        // blocks here. Incoming requests are handled
}
```

Ein solcher Dienst ist von sich aus nicht aktiv, aber er ist in der Lage, eingehende Job Requests zu bearbeiten. Dazu verwendet die SAL den dienstspezifischen Job Manager (im Programm jobMgr), der dem CSAL_StandardBuilder bekanntgegeben wird. Weitere Details bezüglich der vom Anwendungsentwickler bereitzustellenden Klassen werden in Kapitel 3.10 beschrieben.

3.6 Fehlerbehandlung

In einem komplexen System ist es sehr wichtig, dass die Fehlerbehandlung nach allgemeinen Gesichtspunkten erfolgt, andererseits jedoch flexibel genug ist, um spezielle Fehler und Sonderfälle zu unterstützen. Um das zu erreichen, erlaubt die SAL Fehlermeldungen auf zwei Kanälen:

- Durch eine spezielle Performative bei der Rücknachricht. Im Standardfall hat die Rücknachricht die Performative *tell*. Bei unvollständigen Daten wird jedoch *untell* verwendet; *error*, wenn ein grundsätzlicher Fehler auftritt; und *sorry*, wenn der Auftrag

verstanden und für grundsätzlich bearbeitbar gehalten wurde, aber trotzdem nicht bearbeitet werden konnte. Diese Performativen entsprechen formal und inhaltlich der KQML.

- In der System-Ontologie[18] wird eine Fehlerklasse definiert. Diese enthält einen Fehlertext, einen dienstleistungsspezifischen und in der Dienstleistungsschnittstelle festgelegten Fehlercode und eine Fehlerkategorie. Die möglichen Fehlerkategorien sind von der System-Ontologie vorgegeben. Fehlerobjekte werden mit den Ergebnissen einer Dienstleistung zurückgeschickt, und bei Bedarf können auch mehrere Fehlerobjekte gesendet werden.

Fehler-kategorie	**Bedeutung der Fehlerkategorie**	**Rückgabe-performative**
warning	Eine Dienstleistung konnte erbracht werden, jedoch möglicherweise mit Einschränkungen, z. B. qualitativer Art.	untell
temporary	Eine Dienstleistung konnte momentan nicht erbracht werden. Die Inanspruchnahme der Dienstleistung zu einem späteren Zeitpunkt ist jedoch möglich.	sorry
resource	Eine Dienstleistung konnte auf Grund fehlender Ressourcen nicht erbracht werden. Die Inanspruchnahme der Dienstleistung zu einem späteren Zeitpunkt ist jedoch möglich.	sorry
fatal	Eine Dienstleistung konnte nicht erbracht werden. Die Inanspruchnahme der Dienstleistung mit denselben Eingabedaten ist auch zu einem späteren Zeitpunkt nicht möglich.	error
general	Ein schwerwiegender, dienstleistungsübergreifender Fehler ist aufgetreten.	error

Tabelle 2: Zusammenhänge zwischen Fehlerkategorien und Rückgabeperformativen

Die Fehlerübertragung auf diesen beiden Kanälen gestattet eine sehr feine und vielschichtige Fehlerbehandlung:

1. Die Rückgabeperformative enthält eine grobe Angabe darüber, ob die Dienstleistung erfolgreich war oder nicht.
2. Die Existenz von Fehlerobjekten weist darauf hin, dass irgend etwas nicht planmäßig verlaufen ist.

---

[18] Die System-Ontologie unterscheidet sich von den anwendungsspezifischen Ontologien dadurch, dass sie sehr allgemeine Klassen enthält. Sie wird von System-Diensten und der SAL, aber auch von allen anderen Diensten verwendet.

3. Die Kategorie der Fehlerobjekte erlaubt eine allgemeine Aussage über die Art der aufgetretenen Fehler, die von allen Beteiligten im System verstanden wird, auch ohne die Dienstleistung zu kennen. Dies ist beispielsweise für den Workflow-Service wichtig, der im Fehlerfall vernünftig reagieren soll, ohne dass er die Dienstleistungen und die spezifische Bedeutung ihrer möglichen Fehler kennt.
4. Der Fehlercode erlaubt eine automatische, aber spezifische Fehlerbehandlung auf der Basis der Dienstleistungsbeschreibung.
5. Der Fehlertext erlaubt einem menschlichen Benutzer, den Fehler rasch zu bewerten. Er kann Kontextinformationen enthalten, die mit den Fehlercodes allein nicht erfasst werden können, wie beispielsweise die Kennung der Dienstleistung und das Datum.

Zwischen diesen beiden Kanälen bestehen aber auch Zusammenhänge. Rückgabeperformative und die Kategorie des schwerwiegendsten Fehlers sollen zusammenpassen, um ein konsistentes Bild des Fehlers zu geben. Diese Zusammenhänge sind in Tabelle 2 angeführt. Die Tabelle beschreibt die Fehlerkategorien in aufsteigender Reihenfolge ihres Schweregrades, wobei die Fehlerkategorien *temporary* und *resource* als gleich schwerwiegend gelten, ebenso *fatal* und *general*.

Diese Art der Fehlerbehandlung hat sich auch in der Praxis bewährt. Interessant ist außerdem, dass auch die SAL ihre Fehler nach diesem Schema meldet. So gibt es beispielsweise von der SAL definierte Fehlerkategorien für den Fall, dass die Implementierung einer Dienstleistung eine Ausnahme (exception) nicht abfängt oder dass bei einem Absturz und Wiederhochfahren des Agenten die laufenden Dienstleistungen nicht wiederhergestellt werden können (siehe dazu auch weiter unten das Kapitel "Persistenz").

3.7 Protokollierung

Die SAL protokolliert wichtige Ereignisse im Standardbetrieb und detaillierte Informationen im Fehlerfall. Dazu verwendet sie Protokollierungsfunktionalitäten, die von Dieter Wagner im Rahmen des ABRKFUe-Projekts erstellt wurden. Da diese auch von der Implementierung der einzelnen Agenten verwendet werden, ergibt sich eine umfassende Protokollierung der Agenten. In den Protokolldateien finden sich zeitlich geordnet die Protokolleinträge der Agenten-implementierung und der SAL in einem einheitlichen Format. Diese Protokolle erlauben einem Agentenentwickler oder dem Systemadministrator, Laufzeitfehler zu verstehen und zu beheben.

Die Protokolle können über spezielle, von der Protokollierungskomponente zur Verfügung gestellte und von den Diensten einzubindende Dienstleistungen eingesammelt werden. Auf diese Weise beschafft sich der Protokoll-Service im ABRKFUe zyklisch alle Protokolldateien von allen Diensten und ordnet die Protokolleinträge nach Sessions und Zeit.

Dies ist ein schönes Beispiel dafür, auf welch verschiedenen Ebenen das Dienstleistungskonzept nützlich ist. Hier werden Protokolle, die unter anderem Einträge der SAL enthalten, über die SAL an einer zentralen Stelle gesammelt und weiterverarbeitet. Es war nicht notwendig, dafür einen extra Übertragungsmechanismus zu implementieren.

Gleichzeitig ist die Protokollierung ein Beispiel für eine Funktionalität, die auch in einem verteilten System einheitlich erfolgen muss, wenn sie von einer zentralen Stelle ausgewertet werden soll.

3.8 Persistenz

Die SAL stellt Mechanismen für die Persistenz zur Verfügung. Das bedeutet, dass ein Dienst grundsätzlich laufende Dienstleistungen oder offene Dienstleistungsaufträge nicht verliert bzw. verlieren muss, wenn er – aus welchem Grund auch immer – beendet und neu gestartet wird. Diese Anforderung ergibt sich aus der zum Teil sehr langen Dauer einer Dienstleistung und aus den hohen Sicherheitsanforderungen an die Kernreaktor-Fernüberwachung, wo es nicht vorkommen darf, dass bei einer Rechnung im Alarmfall wertvolle Rechenzeit dadurch verlorengeht, dass beispielsweise auf Grund einer Systeminstabilität wegen Überlastung ein Dienst neu gestartet werden muss, der bereits mehrere Stunden Rechenzeit für eine numerisch aufwendige Rechnung investiert hat.

Persistenz ist kein einfaches Problem, und ihre Umsetzung verursacht Kosten, sowohl zur Laufzeit als auch zur Entwicklungszeit. Im folgenden sollen nicht alle Details und Sonderfälle beschrieben werden, die bei der Umsetzung der Persistenz relevant waren, sondern allgemeine Prinzipien dargestellt werden.

Konfigurierbarkeit. Es soll nicht jeder Dienst automatisch die höheren Laufzeitkosten zahlen müssen, auch wenn für ihn eine Persistenz nicht notwendig ist. Dies ist beispielsweise für Dienste der Fall, deren Dienstleistungen nur kurz dauern.

Speicherung in Dateien. Zum dauerhaften Abspeichern von Informationen verwendet die SAL Dateien und keine Datenbank. Dies ist auf Grund der klaren Strukturierung der Informationen und der geringen Anzahl an Querbeziehungen zwischen den Informationen leicht möglich. Gleichzeitig bleiben die Programme für die Dienste kleiner und einfacher.

Art der persistenten Informationen. Die SAL speichert zu jeder laufenden Dienstleistung ihre Kennung, welcher Art sie ist, von wem sie in Auftrag gegeben wurde, und ob sie im stream-modus oder im normalen Modus in Auftrag gegeben wurde (das hat Auswirkungen auf die Art der Rücknachrichten). Zu jedem offenen Dienstleistungsauftrag speichert die SAL die Kennung und den Auftragnehmer.

Mehrfache Ausfälle. Falls der Dienst während des Wiederhochfahrens und Wiederherstellens von Dienstleistungen oder Dienstleistungsaufträgen erneut abstürzt, gehen Zwischenzustände verloren. Die SAL versucht, den Zustand beim ersten Absturz wiederherzustellen.

Zusammenarbeit zwischen SAL und Implementierung. Die SAL kann nur wiederherstellen, was in ihrem Arbeitsbereich liegt. Dazu gehört in erster Linie die Verwaltung von Aufträgen und Ergebnisobjekten und die Koordination zwischen Auftraggebern und Auftragnehmern. Die Speicherung von für die Dienstleistung zum Wiederaufsetzen relevanten Informationen muss durch die Dienstleistung selbst erfolgen. Beim Wiederaufsetzen wendet sich die SAL an das konkrete JobManager-Objekt und teilt ihm gegebenenfalls mit, welche Jobs welcher Art und Kennung vor dem Absturz existiert hatten. Wenn der JobManager dafür ein funktionstüchtiges Jobobjekt herstellen kann, dann übernimmt die SAL es und stellt die Verbindungen zum

Auftraggeber wieder her. Etwas anders sind die Abläufe beim Wiederherstellen von Dienstleistungsaufträgen gelagert. Da Dienstleistungsaufträge (OutgoingJobRequest-Objekte) nicht von der SAL erzeugt werden, sondern vom dienstspezifischen Code, müssen auch nach einem Absturz Ersatzobjekte für die OutgoingJobRequests vom dienstspezifischen Code erstellt werden. Diese werden der SAL bekanntgegeben, und die SAL stellt die Verbindung zum Auftragnehmer wieder her. Wenn alle offenen Aufträge entweder wiederhergestellt oder verworfen worden sind, teilt der dienstspezifische Code der SAL mit, dass die Wiederherstellung von OutgoingJobRequests abgeschlossen ist. Nun kann die SAL in den Normalbetrieb übergehen.

Simultanes Wiederaufsetzen. Die SAL erlaubt das Wiederaufsetzen auf einen gültigen Zustand, wenn gleichzeitig der Auftragnehmer als auch der Auftraggeber abgestürzt oder beendet worden sind und anschließend Wiederhochfahren. Dabei ist es gleichgültig, welcher der beiden Beteiligten zuerst wieder hochfährt. Dies wird dadurch erreicht, dass die SAL beim Wiederhochfahren immer versucht, die relevanten Verbindungen wiederherzustellen. Misslingt dies, geht sie in Warteposition bis der andere Auftragspartner seinerseits versucht, die Verbindung wiederherzustellen. Natürlich ist der tatsächliche Ablauf in einem solchen Fall etwas komplizierter als hier dargestellt, und es gab einige kleine, aber wesentliche Details, die beachtet werden mussten, um ein simultanes Wiederaufsetzen in allen Fällen zu erlauben.

Umfang. Gerade die Wiederaufsetzbarkeit ist eine Anforderung, die sehr viel Aufwand nach sich zieht, gleichzeitig aber in der Praxis nicht zu 100% erreichbar ist. Die in der SAL umgesetzte Lösung ist eine pragmatische und funktionelle. Es wurde jedoch darauf verzichtet, für alle möglichen Komplikationen, insbesondere bei mehrfachen Abstürzen aller Beteiligten, Lösungen anzubieten. Das Resultat wäre ein sehr aufwendiges, fein strukturiertes und detailreiches Protokoll gewesen, das nicht nur von der SAL, sondern auch von den dienstspezifischen Implementierungen umgesetzt werden müßte. Dies war jedoch aus praktischen Überlegungen nicht geboten. Die SAL bietet jedoch Wiederaufsetz-Mechanismen an, die weit über das hinausgehen, was von den meisten Diensten im System überhaupt genutzt werden kann. Praktische Erfahrungen mit der Persistenz werden in Kapitel 4.1 berichtet.

3.9 Allgemeine Dienstleistungen

Im Rahmen der Protokollierung war schon von allgemeinen Dienstleistungen die Rede. Dabei handelt es sich um Dienstleistungen, die für einen Agenten sinnvoll sind, aber keinerlei agentenspezifischen Code benötigen. Zusätzlich bietet die SAL folgende allgemeine Dienstleistungen an, die in jedem Agenten automatisch verfügbar sind. Sie dienen sämtlich der Systemadministration:

Ping. Diese Dienstleistung liefert sofort einen Erfolg zurück. Mit ihrer Hilfe kann getestet werden, ob ein Agent prinzipiell ansprechbar ist.

Herunterfahren. Über diese Dienstleistung kann ein Agent heruntergefahren werden. Dabei gibt es drei Varianten: Warten, bis keine Dienstleistungen mehr bearbeitet werden; sofortiges Herunterfahren mit der Möglichkeit, beim Wiederhochfahren die Dienstleistungen wiederherzustellen; sofortiges Herunterfahren bei gleichzeitiger Verhinderung eines späteren Wiederherstellens der laufenden Dienstleistungen. Die letzte Variante ist wichtig, wenn eine

Dienstleistung hängengeblieben ist und ihre Wiederherstellung zu demselben Fehler führen würde. Dadurch kann – wenn auch unter Verlust von Daten und Rechenzeit – reiner Tisch gemacht werden.

Übersicht über laufende Dienstleistungen. Die SAL bietet eine Dienstleistung, die eine Übersicht über alle laufenden Dienstleistungen mit deren Eckdaten liefert.

Abbrechen einzelner Dienstleistungen. Normalerweise können Dienstleistungen nur vom Auftraggeber abgebrochen werden. Über diese Dienstleistung ist es jedoch möglich, von außerhalb eine Dienstleistung abzubrechen.

3.10 Verwendung der SAL

Am Ende der detaillierten Beschreibung der SAL soll zusammengefasst werden, was aus der Sicht eines Anwenders – also eines Menschen, der einen Dienst mit Hilfe der SAL programmiert – zu tun ist. Dies geschieht in der Absicht, einen anderen Blickwinkel auf die SAL zu eröffnen. Die bisherige Darstellung von internen Aspekten der SAL soll durch eine Betrachtung von außen abgerundet werden.

Wenn man einen Agenten programmieren will, der Dienstleistungen anbietet, muss man folgendes tun:

1. Entscheiden, welche Dienstleistungen man anbietet. Diese sollen genau spezifiziert werden, und zwar durch die Angabe von Ein-, Ausgabe- und Fehlerobjekten.[19] Für diese Dienstleistungen sind Namen notwendig, die in die Konfigurationsdatei der SAL eingetragen werden. Sie müssen natürlich auch implementiert werden.

2. Es muss eine JobManager-Klasse implementiert werden, die unter Angabe des Dienstleistungsnamen ein entsprechendes Job-Objekt erzeugt. Im Programm wird ein Objekt eines solchen Job-Managers erzeugt und dem ebenfalls zu erzeugenden CSAL_StandardBuilder bekanntgegeben.

Dies klingt relativ einfach, und es ist es auch in der Praxis. Trotzdem wollen wir nun diese beiden Schritte etwas genauer ansehen:

Die Implementierung der Dienstleistung erfolgt dadurch, dass man eine Klasse programmiert, welche von der SAL-Klasse CSAL_Job erbt. Diese definiert eine abstrakte Methode doJob(), innerhalb derer in der abgeleiteten Klasse die Implementierung der Dienstleistung erfolgt. Innerhalb der konkreten doJob-Methode werden die Eingabeobjekte der Dienstleistung deserialisiert und eventuelle Fehler abgefangen. Die Ergebnisse der Dienstleistung werden nach getaner Arbeit serialisiert. Anschließend wird eine Methode der SAL aufgerufen, mit der man der SAL mitteilt, dass Datenobjekte zum Zurückschicken anliegen. Der Rückgabewert der doJob-Methode entspricht der Rückgabeperformative der Dienstleistung.

---

[19] Dazu ist es natürlich notwendig, dass in der Ontologie die zu übertragenden Klassen definiert sind. Die Erstellung der Ontologie wird in der Regel unter Berücksichtigung der im System vorhergesehenen Dienstleistungen erfolgen. Dieser Schritt ist zwar frei von theoretischen Problemen, in der Praxis jedoch aufwendig und oft von langenDiskussionen begleitet.

Wenn die Dienstleistung angehalten können werden soll, muss die abgeleitete Job-Klasse die suspend-, resume- oder terminate-Methoden implementieren. Diese sollen schnell zurückkommen. Suspend und Terminate dürfen auch scheitern, aber nach einem erfolgreichen Suspend muss ein darauffolgendes Resume durchgeführt werden können.

Soll die Dienstleistung bei einem Wiederhochfahren nach einem Absturz wieder aufsetzen können, dann muss die abgeleitete Job-Klasse auch die Methode doRecoveredJob() implementieren. Diese wird in diesem Fall von der SAL aufgerufen. Sie kann Code enthalten, der von der doJob()-Methode abweicht.

Die Implementierung einer JobManager-Klasse ist notwendig, weil die SAL auf Anfrage ein Job-Objekt der entsprechenden Klasse erzeugen muss, aber nicht weiß, wie das geht. Wird ihr jedoch beim Hochfahren ein JobManager-Objekt mitgegeben, dann benutzt sie dieses, um die konkreten Jobs zu erzeugen. Die Implementierung von solchen JobManager-Klassen ist sehr einfach: Je nach Namen der Dienstleistung werden die entsprechenden Objekte erzeugt. Eine kleine Spur aufwendiger ist der Fall des Wiederaufsetzens: Hier ruft die SAL nicht die createJob()-Methode auf, sondern eine zu implementierende createRecoveredJob()-Methode. Diese muss etwaige Sonderfälle bei der Erzeugung von Ersatz-Jobs berücksichtigen. In einigen Diensten übernimmt der JobManager als zentraler Einstiegspunkt für Dienstleistungen auch zusätzliche, dienstspezifische Verwaltungstätigkeiten.

Bei der Erzeugung der SAL ist die Konfigurationsdatei anzupassen, in der beispielsweise der Name des Agenten angegeben ist, ferner die Adresse, mit der er sich beim Naming Service registriert und die von Auftraggebern benützt wird, und die Anzahl der gleichzeitig ausführbaren Dienstleistungen.

Bisher haben wir lediglich die Auftragnehmerseite beschrieben. Auf der Auftraggeberseite ist ein CSAL_OutgoingJobRequest-Objekt zu erzeugen. Ihm müssen die Adresse des Auftragnehmers und der Name der Dienstleistung bekanntgegeben werden. Wenn die Ergebnisse der Dienstleistung im stream-Modus geliefert werden sollen, ist stream-all als Anfrageperformative anzugeben.

Die Eingabeobjekte für die Dienstleistung müssen bereitgestellt und dem CSAL_OutgoingJobRequest mitgeteilt werden. Sie werden dabei auch serialisiert. Anschließend kann der Request gesendet werden. Je nach verwendeter spezialisierter Request-Klasse erhält der dienstspezifische Code sofort nach dem Absenden die Kontrolle zurück, oder er wird blockiert, bis die Dienstleistung erbracht ist. Im ersten Fall ist es sinnvoll, über virtuelle Methoden zu implementieren, was beim Empfang von Datenobjekten und beim Ende der Dienstleistung zu tun ist.

Ein solches CSAL_OutgoingJobRequest-Objekt bietet Methoden zum Anhalten, Wiederaufnehmen und Abbrechen der Dienstleistung. Werden sie aufgerufen, schickt die SAL diese Kommandos an den Auftragnehmer weiter, dessen Implementierung dann versucht, die Dienstleistung anzuhalten, wiederaufzunehmen oder abzubrechen. Erfolg und Mißerfolg werden dem CSAL_OutgoingJobRequest mitgeteilt.

Die Ergebnisse einer Dienstleistung können deserialisiert und weiterverarbeitet werden.

Es ist möglich, Requests von jedem Punkt im Programm abzuschicken, vorausgesetzt, die SAL wurde erzeugt. Dies kann beispielsweise im Hauptprogramm erfolgen, oder aber auch im Kontext der Abarbeitung einer Dienstleistung. Natürlich ist es auch möglich, in einem Klienten über eine grafische Benutzeroberfläche verschiedene Dienstleistungen in Auftrag zu geben.

Wie aus dieser Darstellung ersichtlich, ist vereinfacht die SAL die Implementierung von verteilten Systemen wesentlich. Es ist sehr einfach, Code mit der SAL zu verknüpfen und so einen Dienst oder Agenten in einem verteilten System zu realisieren.

4 Das Dienstleistungskonzept in der Praxis

In diesem Kapitel werden die wichtigsten Erfahrungen mit dem Dienstleistungskonzept wiedergegeben. Ihre Auswahl erfolgt in Hinblick auf die Beurteilung der SAL und die Brauchbarkeit der umgesetzten Konzepte und konzentriert sich auf

- den Einsatz im Rahmen der Entwicklung der Ausbreitungsrechnung für die Kernreaktor-Fernüberwachung,
- die Gestaltung und Berücksichtigung der semantischen Abhängigkeit zwischen Diensterbringern und Dienstnutzern,
- den Umgang mit Informationsobjekten und
- die Klasse von Problemen, die mit dem Dienstleistungskonzept gelöst werden können.

Erfahrungen in diesen Bereiche werden in den nachfolgenden Kapiteln dargestellt.

4.1 Erfahrungen in der KFÜ

Die Modernisierung der Kernreaktor-Fernüberwachung verfolgt einige Ziele; im Rahmen dieser Arbeit geht es um die Aufbereitung von Ausbreitungsrechnungen, die bisher nur durch Simulationsexperten durchgeführt werden konnten. Sie sollten jedoch automatisiert und so aufbereitet werden, dass sie auch von Strahlenschutzexperten, die mit dem Simulationssystem nicht vertraut sind, verwendet und zur Entscheidungsfindung genutzt werden können, und zwar auch unter den extremen Bedingungen von Reaktorstör- oder unfällen. Dies erfordert neue Qualitäten der Ausbreitungssimulation (Anwendungssicherheit, Verfügbarkeit, Verlässlichkeit) und vergrößert gleichzeitig ihren Einsatzbereich.

In diesem Zusammenhang konnten generell gute Erfahrungen mit dem Dienstleistungskonzept gemacht werden. Natürlich gab es auch manche Schwierigkeiten, die häufig auf eine unglückliche Anwendung des Dienstleistungskonzepts zurückzuführen waren: Die Neuartigkeit des Dienstleistungskonzepts und des Logischen Klienten führte aufgrund der zunächst mangelnden Erfahrung zu Entwurfsfehlern, die später korrigiert werden mussten. Die wichtigsten Erfahrungen lassen sich folgendermaßen zusammenfassen:

- Das Dienstleistungskonzept ist integraler Bestandteil des Systems. Es ist umfassend genug, um auch in einem komplexen System wie der KFÜ alle Aktivitäten im System zu organisieren und koordinieren.
- Die Standardisierung der Dienstleistungen brachte erhebliche Reduktionen des Entwurfs-, Erstellungs-, Test- und Integrationsaufwandes.
- Dienstleistungen stellen ein hervorragendes Strukturierungsmittel für komplexe Systeme dar. So war beispielsweise die Entwicklung einzelner Dienstleistungen durch externe Programmierer möglich. Wenn es Inkompatibilitäten gab, dann auf fachlicher bzw. die Dienstleistungsschnittstelle betreffender Ebene.

- Durch die von der SAL erzwungene Verwendung einer Ontologie gab es mehr Klarheit bezüglich der Schnittstellen und ihrer Semantik. Natürlich war damit auch ein Koordinationsaufwand verbunden, insbesondere auch zur Erstellungszeit der Ontologie, welche in Absprache mit den Dienstleistungsentwicklern erfolgt ist. Die Vorteile der Verwendung der Ontologie haben in der KFÜ den Aufwand bei weitem aufgewogen.

- In der KFÜ gibt es viele relativ wenig umfangreiche Dienstleistungen. Treibend war hier der Anspruch der Flexibilität: In einem Satz von vielen kleinen Bausteinen können Ersetzungen auch in kleinem Rahmen durchgeführt werden. Dieser Anspruch hat allerdings auch erhebliche Kosten nach sich gezogen: Die Erstellung von Workflows und deren Benutzung durch den Client Manager wird aufwendiger, und das Gesamtsystem wird unübersichtlicher. Deshalb empfiehlt es sich, Dienstleistungen für gut abgegrenzte Aufgaben so grobgranular wie möglich anzubieten und auch beispielsweise die Realisierung unterschiedlicher Varianten eher in einer Dienstleistung unterzubringen als diese auf verschiedene Dienstleistungen aufzuteilen. Diese Empfehlung wird auch dadurch gestützt, dass der tatsächliche Nutzen der durch feine Granularität ermöglichten Flexibilität keineswegs garantiert ist: Bei späteren Ersetzungen ist der Aufwand für die Änderung einer umfassenden Dienstleistung durch den Einbau einer Variante oft kleiner, als wenn zahlreiche Workflow-Beschreibungen erweitert oder angepasst und die Klienten der Workflows überprüft werden müssen.

- Die Umsetzung des Recovery war eingeschränkt erfolgreich. Grundsätzlich können Dienstleistungen in vollem Umfang wiederhergestellt werden. Dazu muss allerdings die Implementierung der Dienstleistung mitmachen. Dies war im Rahmen der Ausbreitungsrechnung durch die Verwendung bereits erprobter Simulationsprogramme nicht immer gegeben. Erschwerend kommt hinzu, dass das Recovery oft in komplexeren Situationen erfolgen soll, in denen es nicht nur um die Wiederherstellung einzelner Dienstleistungen, sondern auch um Workflows mit zahlreichen Unterdienstleistungen und Zwischenzuständen geht. In solchen Situationen genügt manchmal eine Dienstleistung, die nicht wiederhergestellt werden kann, um das Recovery eines gesamten Workflows zu verhindern. Das liegt daran, dass es gerade bei Rahmen- und Teildienstleistungen oft wichtig ist, ob eine Dienstleistung bereits angestoßen wurde oder nicht. Wird im Zuge des Recovery festgestellt, dass der momentane Zustand einer Teildienstleistung korrupt ist, dann ist es nicht möglich, die Teildienstleistung gänzlich ungeschehen zu machen. Hier wären Konzepte wie beispielsweise die Transaktionssicherheit bei Datenbanken denkbar, wo zu jedem Zustandsübergang auch ein sogenannter Roll-Back gemacht werden kann. Ob allerdings eine Verknüpfung von Dienstleistungen mit der Transaktionssicherheit Sinn macht, ist eine Frage, die noch genauer untersucht werden müsste.

- Die automatische Unterstützung von Beziehungen zwischen Informationsobjekten sollte verbessert werden. Dadurch würde der Umfang des händisch zu erstellenden Codes zur Handhabung der Beziehungen deutlich reduziert werden. Ein weiteres Vereinfachungspotenzial besteht auf der Ebene der Ontologie, wo einige Sachverhalte einfacher modelliert werden könnten.

- Ebenfalls lohnend wäre eine automatische Prüfung der Schnittstellen, insbesondere hinsichtlich des Vorhandenseins von Informationen, die in der Ontologie als optional gekennzeichnet sind, aber in der Schnittstellenbeschreibung erfordert oder garantiert werden.

Gerade in solchen Fällen entsteht oft ein erheblicher Integrationsaufwand, der durch eine automatische Schnittstellenprüfung früher erkannt werden könnte.

Die zuletzt angesprochenen Verbesserungsvorschläge geben eine grundsätzliche Erfahrung aus dem praktischen Umgang mit Dienstleistungen wieder: auch Experten können und müssen weiterlernen, insbesondere in einem so jungen und dynamischen Gebiet wie der Agentensysteme. Trotzdem soll hier aber noch einmal festgehalten werden, dass das Dienstleistungskonzept und seine Realisierung in der SAL sich in der KFÜ sehr gut bewährt haben.

4.2 Semantische Abhängigkeit von Diensterbringer und Dienstnutzer

Die Beziehung zwischen Diensterbringer und Dienstnutzer, die auf den ersten Blick so klar und einfach erscheint, hat zahlreiche Auswirkungen auf die Struktur des Gesamtsystems und verdient eine genauere Betrachtung.

Die Abhängigkeit von Diensterbringern zu (möglichen) Dienstnutzern beschränkt sich auf zwei Aspekte:

1. Bei der Festlegung der Schnittstellen wird in der Regel der Verwendungskontext berücksichtigt.

2. Während der Implementierung, der Tests und der Integration kann es passieren, dass von der festgelegten Schnittstelle abgewichen wird oder über sie hinaus weitere Konventionen entstehen. Beides betrifft das Zusammenspiel zwischen einem Diensterbringer und einem konkreten Dienstnutzer und erschwert die Verwendung einer Dienstleistung durch andere Dienstnutzer.

Die Abhängigkeitsbeziehung in der anderen Richtung, vom Dienstnutzer zum Diensterbringer, ist wesentlich interessanter und folgenreicher. Zunächst einmal macht die SAL durch die Vereinheitlichung der Dienstleistungen den Dienstnutzer vom Diensterbringer in syntaktischer und protokollarer Sicht unabhängig. Jeder kann jeden Dienst im System mit der Erbringung von Dienstleistungen beauftragen und die Durchführung anhalten, wiederaufnehmen und abbrechen. Ob ein Dienstleistungsauftrag tatsächlich über die SAL abgewickelt werden kann, entscheidet sich – wie in verteilten Systemen nicht anders zu erwarten – erst zur Laufzeit. Zur Implementierungszeit ist der Dienstnutzer "nur" semantisch vom Diensterbringer abhängig: Es muss entschieden werden, ob der Diensterbringer überhaupt für die vorgesehene Aufgabe eingesetzt werden kann, ob die erforderlichen Eingangsdaten zur Verfügung stehen und die zu erwartenden Ausgangsdaten sinnvoll weiterverarbeitet werden können.

Die semantische Abhängigkeit von Diensterbringern kann in intelligenten Systemen erst zur Laufzeit relevant werden: Wenn ein Dienstnutzer eine formale Dienstleistungsbeschreibung laden, auswerten und berücksichtigen kann, dann kann er erst zur Laufzeit entscheiden, welche Dienstleistungen geeignet, wie die dafür notwendigen Eingangsdaten bereitzustellen sind und die von der Dienstleistung produzierten Ausgangsdaten weiter verwendet werden können. Dieses Szenario wird von der SAL aufgrund der syntaktischen und protokollaren Unabhängigkeit voll unterstützt. Darin liegt – neben der Reduktion des Einarbeitungs- und Implementierungsaufwandes auf Grund der Einheitlichkeit der Inanspruchnahme von Dienstleistungen – eine der großen Stärken der SAL. Diese Stärke wird im Logischen Klienten

und in der KFÜ derzeit nicht genutzt, weil die Entwicklung solcher intelligenter Dienstnutzer schwierig und aufwändig ist. In einem Ausblick in Kapitel 5.1 wird dieser Gedanke jedoch weiter verfolgt.

Bei den meisten Dienstnutzern der Ausbreitungsrechnung in der KFÜ ist die semantische Abhängigkeit von Dienstleistungen im Programmcode berücksichtigt. Eine Sonderstellung hat der Workflow-Service inne, der Dienstleistungen anstößt, ohne dass in seiner Implementierung dienstleistungsspezifisches Wissen eingeflossen ist. Dies funktioniert, weil das dienstleistungsspezifische Wissen in der Workflow-Beschreibung enthalten ist und berücksichtigt werden kann. Damit wird die im Code festgehaltene Abhängigkeit von Dienstleistungen von der Implementierung des Workflow-Service zu den Workflow-Beschreibungen verlagert, was sich als äußerst mächtiges Modellierungskonstrukt erwiesen hat. Solange allerdings solche Workflow-Beschreibungen ebenso wie die Implementierungen der Dienstnutzer händisch erfolgen, bleibt die semantische Abhängigkeit des Dienstnutzers zum Diensterbringer zur Erstellungszeit in vollem Umfang bestehen.

Die Abhängigkeit des Dienstnutzers vom Diensterbringer ist von der Sache her keineswegs unerwartet. Durch die SAL kann zwar jeder Dienst jeden anderen Dienst mit der Erbringung von Dienstleistungen beauftragen (es handelt sich also um eine Peer-to-Peer-Architektur [50]), aber er muss gleichzeitig die Semantik der Dienstleistung berücksichtigen. Diese Berücksichtigung bedeutet in der Praxis mitunter erheblichen Wartungsaufwand: Wenn eine Dienstleistung geändert werden muss, dann müssen alle Nutzer dieser Dienstleistung – direkt oder über die Workflow-Beschreibungen – angepasst werden. Und wenn eine neue Dienstleistung angeboten wird, dann sollten die bestehenden Implementierungen geprüft werden, ob sie diese neue Dienstleistung nutzen können. Im schlimmsten Fall wächst der Wartungsaufwand quadratisch mit der Anzahl der in einem speziellen Kontext verwendbaren Dienstleistungen.

Um dieses Problem zu entschärfen macht der Logische Klient Vorgaben, welche die zunächst sehr flexible soziale Struktur eines Peer-to-Peer-Systems einschränkt. Es erfolgt eine Unterscheidung zwischen Systemdiensten und normalen Diensten: Normale Dienste kennen bloß die Dienstleistungen der Systemdienste, und Systemdienste sind nicht abhängig von den Dienstleistungen der normalen Dienste. Das logisches Bindeglied zwischen diesen beiden Welten ist der Strategie-Service, der die Dienstleistungsbeschreibungen der normalen Dienste verarbeiten kann und Dienstleistungen vermittelt. Das praktische Bindeglied zwischen diesen Welten ist der Workflow-Service, der im Laufe eines Workflows verschiedene Dienstleistungen anspricht und koordiniert.

Auf diese Weise wird das Wachstum des Wartungs- und Erweiterungsaufwandes reduziert. Wenn sich eine Dienstleistung ändert, müssen in der KFÜ alle Workflows angepasst werden, die diese Dienstleistung verwenden. Die Auftraggeber solcher Workflows können davon in der Regel unberührt bleiben. Hier bewährt sich einmal mehr die Abstraktionsstufe von Workflows als Aggregation von Dienstleistungen. Allerdings ist damit das Problem zwar graduell, aber nicht prinzipiell gelöst. Neue Dienstleistungen können zu neuen Workflows führen, und diese können nur dann konsumiert werden, wenn die Agenten angepasst werden, die Nutzen aus einem solchen neuen Workflow ziehen. In der KFÜ gibt es 144 Dienstleistungen und 47 Workflows.

Die Abhängigkeit des Dienstnutzers von den genutzten Dienstleistungen bzw. Workflows hat bedeutende Konsequenzen für die Systemarchitektur eines komplexen und verteilten Systems. Eine schlechte Aufteilung der Aufgaben und Kompetenzen kann zu zahlreichen, fachlich nicht begründeten Abhängigkeiten und unnötig hohem Wartungsaufwand führen. Dieses Grundproblem aller komplexen Systeme wird durch SAL und Logischen Klienten zwar nicht gelöst, wohl aber so transparent gemacht, dass es leichter erkannt und berücksichtigt wird.

4.3 Lebensdauer und Lebensraum von Informationsobjekten

Informationsobjekte dienen als Ein- und Ausgangsdaten von Dienstleistungen. Ein Informationsaustausch an der Dienstleistungsschnittstelle vorbei ist nicht vorgesehen und im Interesse der Klarheit der Schnittstellenbeschreibung auch nicht erwünscht. Eine Konsequenz daraus ist jedoch, dass es – im Gegensatz beispielsweise zu Blackboard-Architekturen – keinen "Ort" gibt, an dem Informationsobjekte dienstleistungsübergreifend aufbewahrt werden. Auch der Repository-Service ist dafür nicht gedacht: Es verwaltet zwar Dateien, aber die zugehörigen Meta-Informationsobjekte sind im System ebenso transient wie alle anderen Informationsobjekte: Sie werden irgendwann erzeugt, und können bei der Inanspruchnahme von Dienstleistungen oder Workflows als Eingangsinformationen mitgegeben oder als Ausgangsinformation einem Auftraggeber geliefert werden.

Wenn Informationsobjekte in verschiedenen Dienstleistungen existieren sollen, dann müssen sie vom Auftraggeber der Dienstleistungen zur Verfügung gestellt werden. Das bedeutet, dass die Informationsobjekte von einem übergeordneten Kontext - zum Beispiel einem gemeinsamen Auftraggeber - verwaltet werden müssen. Die Hierarchie der übergeordneten Kontexte ist jedoch begrenzt. Um zu vermeiden, dass übergeordnete Kontexte mit für deren Aufgaben eigentlichen irrelevanten Informationsobjekten belastet werden, ist eine umsichtige Wahl der Dienstleistungen und Workflows notwendig. Dieses Prinzip bzw. Problem ist auch in anderen Bereichen - etwa der Kompetenzenverteilung von Europäischer Union, Bund, Ländern und Gemeinden anzutreffen.

Ein Dienst, der unter anderem eigens dafür vorgesehen ist, übergeordnete Kontexte herzustellen und zur Laufzeit Informationsobjekte zu verwalten und weiterzugeben, ist der Workflow-Service. Er wird damit zu einem zur Laufzeit sehr wichtigen Bindeglied, das die isolierten Dienstleistungen verknüpft. Wenn es jedoch Aufgaben gibt, die nicht im Rahmen eines einzigen Workflows sinnvoll erledigt werden können, dann muss der Auftraggeber der Workflows einen übergeordneten Kontext herstellen. In der Praxis der Ausbreitungsrechnung ist hiervon der Client Manager betroffen, der den Kontext von Simulationsberechnungen herstellt, in deren Rahmen mehrere Workflows ablaufen. Dadurch fällt ihm aber auch die Aufgabe zu, die produzierten Informationsobjekte zu verwalten, wofür er eine eigene Datenbank verwendet. Um zu vermeiden, dass die Aufgabe der Verwaltung von Informationsobjekten an verschiedenen Stellen – eben zum Beispiel beim Workflow-Service und beim Client Manager – erfolgen muss, wäre es denkbar, für genau diese Aufgabe einen Datenservice zu realisieren. Das hätte jedoch die Konsequenz, dass die Schnittstellenbeschreibung einer Dienstleistung neben den Ein- und Ausgangsdaten auch noch Angaben zu den Informationsobjekten enthalten müsste, die beim Datenservice benötigt und abgelegt werden. Darauf wurde bei der Ausbreitungsrechnung im KFÜ jedoch verzichtet.

Als Konsequenz aus diesen Umständen muss beim Systementwurf nicht nur die Festlegung von Dienstleistungen, Diensten und Workflows erfolgen, sondern auch die Lebensdauer und der Lebensraum von Informationsobjekten berücksichtigt werden.

4.4 Dienstleistungen und das prozedurale Programmierparadigma

Dienstleistungen sind in ihrer Struktur stark an die klassischen Prozeduren angelehnt: Sie benötigen Eingabedaten, machen etwas und produzieren Ausgabedaten. Dieses Prinzip findet sich durchgängig im gesamten Logischen Klienten. Auch Workflows reihen sich nahtlos ein, ist doch ihre Abarbeitung als (aggregierte) Dienstleistung umgesetzt. Daraus ergibt sich eine natürliche Eignung des Logischen Klienten für Probleme, die prozedural gelöst werden können.

Im Gegensatz dazu gibt es Probleme, die sich besser durch einen regelbasierten Ansatz lösen lassen: Anstatt festzulegen, welche Schritte in welcher Reihenfolge durchgeführt werden müssen um ein Problem zu lösen, werden beim regelbasierten Programmieren Regeln angegeben. Eine Regel beschreibt einen Zustand und eine Aktion, die durchgeführt werden soll, wenn der Zustand erreicht ist. Dabei kann es passieren, dass die geforderten Zustände mehrerer Regeln erfüllt sind; in welcher Reihenfolge die Regeln dann abgearbeitet werden, ist zur Laufzeit jedoch von nachgeordneter Bedeutung. Das Erstellen von Regeln ist demnach mit der Schwierigkeit verbunden, dass es prinzipiell passieren kann, dass durch die Abarbeitung von Regeln Zustände erreicht werden, in denen keine Regel mehr greift, oder dass Zyklen auftreten, oder dass das Problem zwar in manchen Fällen, aber nicht immer gelöst werden kann. Während man also beim prozeduralen Programmieren einen Ablauf vorgibt bzw. vorgeben muss, der im Normalfall zum Ziel führt, muss man beim regelbasierten Programmieren nicht an Abläufe denken, aber es kann passieren, dass das Ziel nicht erreicht wird.

Nun sind die Grenzen zwischen prozeduralem und regelbasiertem Programmieren fließend. Es ist möglich, in einem regelbasierten System die Regeln so zu formulieren, dass damit ein strikter Ablauf programmiert wird. Umgekehrt kann man in einer prozeduralen Programmiersprache nach jedem Schritt Zustände prüfen und damit verknüpfte Aktionen durchführen, also genau das implementieren, was zur Laufzeit in einem regelbasierten System passieren würde. Dass man mit prozeduralen Sprachen regelbasiert arbeiten kann, wird auch dadurch sinnfällig, dass beispielsweise CLIPS, ein Akronym für "C Language Integrated Production Shell", mit dessen Hilfe regelbasierte Systeme integriert und umgesetzt werden können, selbst in der prozeduralen Sprache C geschrieben ist [51]. Prozedurales und regelbasiertes Programmieren schließen sich also keineswegs aus, aber es ist umständlich, mit Hilfe von Regeln Prozeduren zu programmieren, und es ist ebenfalls umständlich, prozedural zu programmieren, wenn dabei ständig gewisse Zustände überprüft werden müssen und es keine natürliche oder im Voraus bekannte Reihenfolge der Ausführung gibt[20]. Die Frage ist vielmehr, welche Probleme sich in welchem Programmierparadigma besser formulieren und lösen lassen bzw. praktisch überhaupt erst lösbar sind.

---

[20] Vgl. dazu auch [52], insbesondere die Kapitel "Procedural Paradigms", "Nonprocedural Paradigms" und "Misuse of Salience" (Salience ist ein Möglichkeit von CLIPS, die Reihenfolge der Ausführung von Regeln zu beeinflussen und damit letzlich auch strenge Abläufe und Prozeduren zu implementieren).

Für die regelbasierte Lösung von Teilproblemen stellt das Dienstleistungskonzept bzw. der Logische Klient keine Hilfsmittel zur Verfügung. Das bedeutet natürlich nicht, dass solche Teilprobleme unlösbar sind. Die Regeln werden innerhalb einer Dienstleistung entweder mit prozeduralen Sprachmitteln oder durch regelbasierte Sprachmittel implementiert. Wichtig ist jedoch, dass es im Logischen Klienten im Gegensatz zu Blackboard-Architekturen keine zentrale Stelle gibt, an der alle Informationen zusammenlaufen. Vielmehr muss das System so angelegt werden, dass die notwendigen Informationen der Dienstleistung zum richtigen Zeitpunkt zur Verfügung stehen.

Daraus ergibt sich, dass Teilprobleme, die sich regelbasiert besser lösen lassen, besondere Aufmerksamkeit beim Entwurf des Gesamtsystems erfordern. Im folgenden wird für solche Teilprobleme ein spezieller Agententyp skizziert. Prozedurale Aspekte hingegen, die in der Praxis ohnehin den überwiegenden Teil ausmachen, werden vom Dienstleistungskonzept und vom Logischen Klienten auf natürliche Weise unterstüzt.

5 Dienste in regelbasierten Systemen

Dieser Ausblick greift zunächst die Ansätze der vorangegangenen Seiten auf, um eine Kombination von Dienstleistungskonzept und regelbasierter Auftragsvergabe zu skizzieren. Daran anschließend wird ein Exkurs über die Unterschiede zwischen Daten, Informationen und Wissen gegeben und darauf aufbauend drei Bereiche dargestellt, in denen die KFÜ durch Übernahme der hier geschilderten Aspekte weiterentwickelt werden könnte.

5.1 Regelbasierte Auftragsvergabe

Wie im vergangenen Abschnitt dargestellt, ist das Dienstleistungskonzept und der Logische Klient stark dem prozeduralen Programmierparadigma verpflichtet. Um das Dienstleistungskonzept mit einer regelbasierten Ansteuerung von Aktivitäten zu verbinden, müsste man in den Logischen Klienten einen weiteren Agententypus einführen. Als Name soll hier "Master-Agent" dienen. Master-Agenten haben folgende Eigenschaften:

- Sie verwenden die SAL und bieten Dienstleistungen an.
- Sie können andere Agenten mit der Erbringung von Dienstleistungen beauftragen. Dies motiviert auch den Begriff Master-Agent, da sie wie im Master-Slave-Konzept an andere Agenten Arbeitsaufträge vergeben [50]. Hier kommt eine Stärke des Dienstleistungskonzepts, die syntaktische Einheitlichkeit der Dienstleistungen zum Tragen.
- Sie verfügen über ein regelbasiertes System, das wie beim Workflow-Service die externen Dienstleistungen anstößt. Die Regeln, welche die Dienstleistungen triggern, sind jedoch im Gegensatz zum Workflow-Service anwendungsspezifisch und können beispielsweise auf die allgemeine Situation (also Rahmenbedingungen des Auftrags), aber auch auf bisherige Ergebnisse Rücksicht nehmen. Daraus ergibt sich die Möglichkeit, Abläufe durchzuführen, die schwer zu planen und in eine prozedurale Form zu bringen sind. Bei Bedarf kann die Aufgabe eines Master-Agenten in Rahmen- und Teildienstleistungen aufgeteilt werden.
- Das Systemwissen ist im Regelwerk des Master-Agenten festgehalten. Im Idealfall kann es soweit gekapselt werden, dass der Auftraggeber des Master-Agenten kein Wissen über die im Master-Agent vorhandenen Regeln benötigt.
- Master-Agenten nehmen im Logischen Klienten eine bisher nicht vorgesehene Mittelstellung ein: Sie können andere Agenten direkt ansprechen (was derzeit nur der Workflow-Service darf), verfügen über Planungskompetenzen (was bisher allein dem Strategie-Service vorbehalten war), sind aber für jeweils einen Aufgabenbereich unmittelbar zuständig und kennen die Dienstleistungen, die zur Erfüllung der Aufgaben in diesem Bereich beitragen können.
- Planung und Anstoßen von Dienstleistungen erfolgt innerhalb des Regelwerks der Master-Agenten. Dadurch gibt es keine Trennung von Planung (Strategie-Service) und Durchführung (Workflow-Service) mehr, und die prozedurale Beschreibung der Abläufe (in den Workflow-Beschreibungen) und die Koordination zwischen Planung und Ausführung kann entfallen.

- Durch diese Zuständigkeitsbereiche von Master-Agenten kann das Gesamtsystem nach inhaltlichen Gesichtspunkten gegliedert werden. Dies kann einen weiteren Beitrag zur Komplexitätsreduktion darstellen. So könnte es in der KFÜ beispielsweise für Eingangsdatenbeschaffung und Rechnungsdurchführung eigenständige Master-Agenten geben.
- Bei der Umsetzung von Master-Agenten können wichtige Teile des Entwurfs und der Implementierung des Workflow-Service übernommen werden. Dies betrifft vor allem die Ansteuerung externer Dienstleistungen und die Verwaltung von Informationsobjekten.
- Master-Agenten sind durch ihr regelbasiertes Kernstück jedoch nur bedingt in der Lage, Prozeduren durchzuführen. Deshalb macht es Sinn, in einem Gesamtsystem für solche Aspekte auch weiterhin Workflow- und Strategie-Service einzusetzen.

Das Konzept der Master-Agenten lässt hoffen, dass die Anzahl der Workflows in einem System weiter reduziert wird. Dies kann um so mehr erfolgen, je mehr Abläufe sinnvoll durch Regeln beschrieben werden können. Zahlreiche Workflows in der KFÜ entstehen durch Variation einiger weniger Muster; diese Workflows könnten durch Regeln möglicherweise einfacher beschrieben werden. Damit hätte man in der Praxis nicht nur die Zahl der Workflows reduziert, man hätte auch das momentan in einer Vielzahl an Workflows implizit vorhandene Systemwissen in vermutlich wenigen Regeln explizit gemacht. Die dadurch gewonnene Klarheit würde den Wartungs- und Erweiterungsaufwand des Systems reduzieren.

Es ist die Überzeugung des Autors, dass mit Hilfe solcher relativ intelligenter Master-Agenten der Logische Klient strukturell verbessert und auch in Anwendungsgebieten eingesetzt werden kann, die sich einer prozeduralen Beschreibung der Abläufe verwehren. Ein praktischer Beweis für diese Überzeugung konnte im Rahmen dieser Promotionsarbeit aus zeitlichen Gründen nicht mehr gegeben werden. Allerdings lassen sich gute Gründe für diese Überzeugung angeben (Kap. 5.2) und einige interessante Konsequenzen daraus ableiten (Kap. 5.3).

5.2 Exkurs: Daten – Informationen – Wissen

Das Collaborative Agent Design Center der California Polytechnical University verfolgt bei der Entwicklung von Agentensystemen den regelbasierten Ansatz. Dabei legt es großen Wert auf die Unterscheidung zwischen Daten, Informationen und Wissen. Diese Unterscheidung ist auch für den Logischen Klienten und für die KFÜ sowie im Kontext des Umweltinformationssystems Baden-Württemberg [10] - [14] wichtig und soll deswegen an dieser Stelle vorgestellt werden.

In Anlehung an ein Beispiel von Jens Pohl [53] sei hier folgendes Szenario diskutiert: Eine elektronische Nachricht habe den Inhalt "Am Dienstag, 5. September 2000 fiel in Baden-Württemberg zwischen 9:00 und 9:30 Uhr 100 mm Niederschlag." Für den Computer sind dies Zeichen ohne irgend eine Bedeutung. Nach Abbildung 21 wäre dies dem Bereich der unorganisierten Daten zuzuordnen. Wird der Gehalt dieser Nachricht beispielsweise in eine Niederschlags-Datenbank eingetragen, so steigt der Organisationsgrad der Daten und damit deren Verwendbarkeit an. Um jedoch die Stufe der Information zu erreichen, ist eine weitere Verknüpfung der Daten wichtig: So könnten damit weitere geografische Daten/Informationen verknüpft sein, zum Beispiel über betroffene Städte, Berge, Straßen und Flüsse, oder aber auch die weitere Wetterprognose.

Wissen entsteht durch zusätzliche Interpretation und die Anwendung von Regeln[21]. So könnte ein Mensch bei entsprechender Sachkenntnis aus obiger Meldung eine Warnung abgeben: "Ein derartiges Ausmaß an Regen in so kurzer Zeit führt mit hoher Wahrscheinlichkeit zu Überschwemmungen und kann bei starken Hangneigungen zu Erdrutschen führen. Exponierte Straßen im Schwarzwald sind gefährdet. Angesichts der Prognose für die nächsten Tage ist in weiterer Folge mit einer Überschwemmung des Rheins zu rechnen." Aus dieser fiktiven Interpretation wird deutlich, dass ein reiches Beziehungsgeflecht zwischen den Daten enorm hilfreich ist: Eine Vernetzung geographischer Information wie hier des Regengebietes, der darin vorkommenden Flüsse und Hangneigungen, Städten und Straßen und der Wetterprognose erlaubt es Menschen, daraus wichtige Schlüsse schnell zu ziehen und beispielsweise Straßenabschnitte gezielt zu sperren.

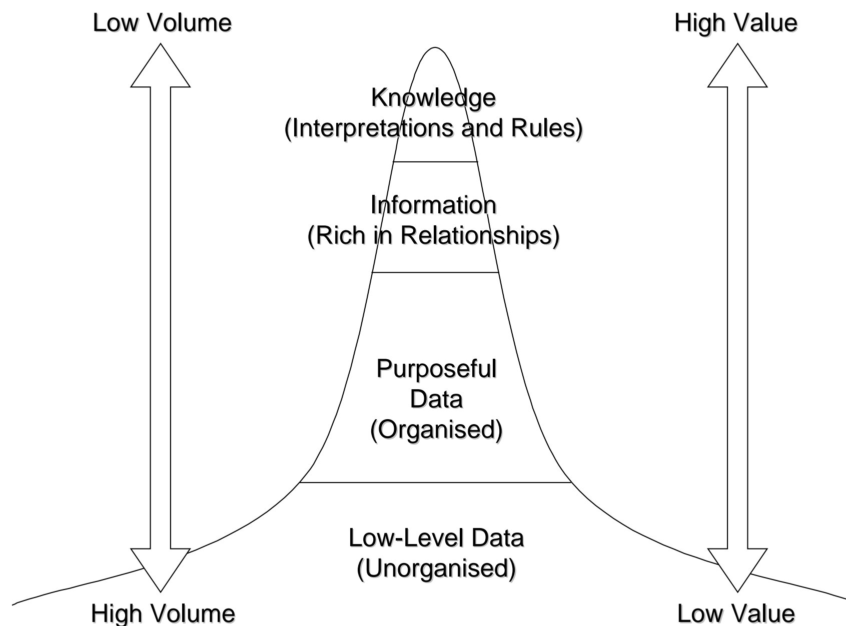


Abbildung 21: Volumen und Wert von Daten und Informationen nach [53].

Diese Unterscheidung Daten – Informationen – Wissen passt sehr gut zu den Anwendungsgebieten der Agentensysteme des Collaborative Agent Design Center, bei denen ein Informationssystem den Menschen bei seiner manchmal auch intuitiven Entscheidungsfindung unterstützt. Eine Übertragung auf andere Gebiete sollte nur mit Bedacht erfolgen.

Tatsache scheint jedoch zu sein, dass das Volumen von Daten allgemein höher ist als das von Informationen und dass der Wert und die unmittelbare Verwendbarkeit von Informationen wesentlich höher ist als die von Daten. Gleichzeitig können Computersysteme nur in sehr seltenen und sehr gut abgegrenzten Anwendungsgebieten bis zur Ebene des Wissens vordringen. Computersysteme bleiben oft auf einer Stufe. Gelingt es jedoch, automationsgestützt eine höhere Stufe zu erreichen, dann wächst die Nützlichkeit das Systems rasch. Ein Beispiel hierfür ist

[21] Mit Regeln sind hier zunächst nicht formale Regeln bei regelbasierten Systemen gemeint, sondern menschliches Wissen, das Schlußfolgerungen erlaubt.

Bilderkennung, wo aus Bitmaps (unorganisierte Daten) zunächst geometrische Objekte extrahiert werden (organisierte Daten), aus denen in weiterer Folge auch Informationen generiert werden können (das Auto befindet sich auf der Straße und fährt auf eine Linkskurve zu); das Werkstück befindet sich in Position X) und Entscheidungen getroffen werden können (Lenkvorgang einleiten; Maschine einschalten).

In der KFÜ bewegen wir uns im Allgemeinen eher auf den Daten-Ebenen. Das muss nicht so bleiben; im folgenden Kapitel werden diesbezüglich einige Anregungen gegeben.

5.3 Ausbaumöglichkeiten der KFÜ

Im vorigen Kapitel wurde die Taxonomie Daten – Informationen – Wissen eingeführt. In der derzeitigen Ausbaustufe der Kernreaktor-Fernüberwachung Baden-Württemberg operiert das System vorwiegend auf der Ebene der Daten. In eine Datenbank werden Messwerte eingetragen, wenn Radioaktivitäts-Messwerte eine bestimmte Schwelle überschreiten, wird automatisch eine Alarmrechnung gestartet. Innerhalb dieser wird eine enorme Menge an Daten produziert, welche für das System nicht weiter auswertbar sind. Diese Daten werden an der Benutzeroberfläche aufbereitet und mit geografischen Informationen verknüpft. Der Benutzer sieht letztendlich farbige Strahlenbelastungskarten und kann auf dieser Messstationen anklicken und deren Werte abfragen.

Die Verknüpfung der Daten mit geografischen Informationen lässt sich bereits der Ebene der Informationen zuordnen.[22] Eine Interpretation, also ein Operieren auf der Wissensebene, erfolgt erst im Kopf der Benutzer.

Wenn es nun gelingt, in mehreren Bereichen das System von der Ebene der Daten auf die Ebene der Informationen bzw. von der Ebene der Informationen auf die Ebene des Wissens zu heben, dann kann die Leistungsfähigkeit und Akzeptanz des Systems enorm steigen. Dazu bieten sich vor allem folgende Bereiche an:

- Agenten könnten bei der Alarmauslösung differenzierter vorgehen. Der überragende Anteil an Alarmauslösungen tritt beispielsweise dann auf, wenn nach einer längeren Trockenperiode die "natürliche" Radioaktivität in der Atmosphäre angestiegen ist. Der erste ergiebige Regenfall führt dann zur nassen Deposition der Nuklide, und die Messwerte schnellen hoch. Durch eine Verknüpfung der Radioaktivitäts-Messwerte mit Niederschlagsmesswerten können solche falschen Alarmmeldungen deutlich eingeschränkt werden.
- Agenten könnten unter Berücksichtigung weiterer Informationen die Unfallkategorie ermitteln. Dies ist ein Schwachpunkt im gegenwärtigen System, wo nach dem Motto "Sicher ist sicher" bei einer Alarmrechnung automatisch die schwerste Unfallkategorie angenommen wird, was zu überhöhten Ergebnissen führt.

---

[22] Interessant ist in diesem Zusammenhang, dass es am meisten Spaß macht bzw. am beeindruckendsten ist, wenn man an der Benutzeroberfläche kreuz und quer durch die Karte klickt und Messwerte in verschiedenen Formen anzeigen lässt. Dies ist wahrscheinlich kein Zufall, ist dies doch der Bereich, der in der Taxonomie Daten – Informationen – Wissen am höchsten angesiedelt ist.

- Agenten könnten die Simulationsergebnisse inhaltlich auswerten, indem sie feststellen, in welchen Gebieten welche Belastungen mit welchen Folgewirkungen auftreten. Dementsprechend könnten sie gezielt Alarm schlagen und dafür sorgen, dass ein Entscheidungsträger in einer Alarmsituation, die ja immer auch eine enorme Stresssituation ist, nichts übersieht.

Um dies zu erreichen ist folgendes notwendig:

- Die Ontologie muss verfeinert werden. So verfügen zur Zeit beispielsweise die Dosis-Ontologieobjekte über folgende Attribute: Organ, auf das sich die Angabe bezieht, Personengruppe (Erwachsene, Kinder), Expositionspfad (Gammasubmersion, Inhalation, Ingestion), Ablagerungszeit (ob die Angabe für die momentane Situation, oder kumuliert über eine Woche, ein Jahr, 50 Jahre oder 70 Jahre gilt). Entsprechend der Trennung von Daten und Metadaten gibt es eine Datei, die die Daten enthält, und ein beschreibendes Objekt, in dem die angeführten Attribute vermerkt sind. Auf dieser Basis kann ein Agent jedoch noch keinerlei Rückschlüsse ziehen auf den Schweregrad und den Ort der Belastung. Doch genau diese Verknüpfung wäre nötig, um die Intelligenz des Systems zu erhöhen.
- Die Übergänge von der Ebene der Daten zur Ebene der Informationen müssen implementiert werden. So muss beispielsweise aus der vom Simulationsmodul generierten Datei zur Dosisleistung der Schweregrad und der Ort der Belastung extrahiert werden. Aus Niederschlagsmesswerten muss ermittelt werden, ob es sich um eine Trockenperiode handelt oder nicht. Aus Messwerten aus dem Kraftwerk müssen Rückschlüsse auf eine eventuelle Unfallkategorie gezogen werden.
- Das Operieren auf der Ebene der Informationen sollte sinnvollerweise durch ein regelbasiertes System erfolgen. Hier sind die in Kapitel 5.1 Szenario I skizzierten Schritte zu gehen.

Mit diesen Schritten könnte die Leistungsfähigkeit und Akzeptanz der KFÜ verbessert werden. Die dazu notwendige Infrastruktur ist mit dem Logischen Klienten und dem Dienstleistungskonzept bereits gegeben.

6 Zusammenfassung

Diese Zusammenfassung gliedert sich in drei Teile: In eine Zusammenfassung wesentlicher Aspekte und Ergebnisse der SAL als Dienstleistungsframework, in Erfahrungen mit der SAL und in Anmerkungen zur Entwicklung von Agentensystemen.

6.1 Die SAL als Dienstleistungsframework

Die folgenden Ergebnisse betreffen die SAL und belegen, dass die gestellten Anforderungen vollständig umgesetzt werden konnten.

1. Eine standardisierte Nachrichtenübertragung zwischen Agenten oder Diensten ist ein wesentlicher Schritt für eine flexible und leistungsfähige Systemarchitektur.

2. In Simulationssystemen wird auf Grund der Dauer der einzelnen Aktivitäten und der Komplexität von Ein- und Ausgabe ein Dienstleistungskonzept notwendig.

3. Die SAL ist ein von allen Agenten oder Diensten verwendetes Dienstleistungsframework. Im Rahmen der SAL wurde ein Dienstleistungskonzept entwickelt und auf der Basis der Übertragung von Nachrichten umgesetzt. Dienste und Agenten implementieren spezielle Dienstleistungen und können über die SAL leicht Dienstleistungs-Aufträge vergeben.

4. Die erhoffte Einsparung an Implementierungsaufwand bei den einzelnen Agenten durch die Verwendung eines einheitlichen Frameworks ist eingetreten.

5. Durch die SAL kann grundsätzlich jeder Agent jeden anderen Agenten mit Dienstleistungen beauftragen. Dadurch bleibt die Systemarchitektur flexibel. Das Dienstleistungskonzept ist eine echte Erweiterung bestehender Agentensysteme.

6. Die Übertragung von Informationen durch die SAL geschieht auf generische Art. Die SAL ist nicht von den übertragenen Informationen und dem Anwendungsbereich abhängig. Zur Übertragung wird auf eine Ontologie zurückgegriffen, auf deren Basis durch Skripte C++ - Code zum Serialisieren und Deserialisieren erzeugt wird.

7. Die Kommunikation zwischen Agenten basiert auf CORBA und erlaubt die einfache Verteilung der Agenten auch über Rechnergrenzen hinweg. Gleichzeitig ist ein Austausch der Middleware leicht möglich, wodurch keine Abhängigkeit von CORBA entsteht.

8. Die Zahl der zur Umsetzung des Dienstleistungskonzepts notwendigen Nachrichtentypen ist erstaunlich gering: (1) Auftragsvergabe im normalen Modus, (2) Auftragsvergabe im stream-modus, (3) Senden von Ergebnissen, (4) Benachrichtigung über den Abschluss eines Auftrags, (5) Melden von Warnungen und Fehlern, (6) Anhalten einer Dienstleistung, (7) Wiederaufnahme einer angehaltenen Dienstleistung, (8) Abbrechen einer Dienstleistung, (9) Statusabfrage einer Dienstleistung, (10) Aufforderung zum erneuten Senden von Ergebnissen nach Kommunikationsproblemen.

9. Intern verwendet die SAL ein genaues Transaktionsprotokoll mit klar definierten Zustandsübergängen.

10. Die SAL kann ihre Zustände persistent machen und in Zusammenarbeit mit der Implementierung von Dienstleistungen und der agentenspezifischen Dienstleistungs-Auftragsvergabe ein vollständiges Recovery nach einem Ausfall durchführen.
11. Die SAL unterstützt ein mehrstufiges Fehlerkonzept.
12. Die SAL protokolliert ihre Aktionen. Die Protokollierung kann gemeinsam mit der Implementierung der Dienstleistungen erfolgen.
13. Die SAL bietet allgemeine Dienstleistungen an, die dem Herunterfahren, dem Testen der Verbindung, der Abfrage nach laufenden Dienstleistungen und Dienstleistungsaufträgen und dem Anhalten, Wiederaufnehmen und Abbrechen von Dienstleistungen dienen.
14. Die SAL ist plattformunabhängig, greift aber vereinzelt auf plattformabhängige, allgemeine Bibliotheken zu. Bei einer Portierung wären lediglich diese Bibliotheken anzupassen.
15. Die von C++ und anderen Softwarebibliotheken zur Verfügung gestellten fortgeschrittenen Programmiertechnologien konnten gewinnbringend eingesetzt werden. Hierzu wären Namespaces, Vererbung, Polymorphie, Multithreading, Verteilung mit CORBA, Entwurfsmuster, allgemeine Softwarebibliotheken und die Standard Template Library zu zählen.
16. Der Frameworkansatz und die Klassenstruktur der SAL gewähren eine leichte Anpass- und Erweiterbarkeit.
17. Die SAL hat sich in der KFÜ mit seinen sehr hohen Anforderungen bewährt. Die Einbindung vorhandenen Codes ist leicht möglich, und trotz der parallelen Entwicklung und Integration sind die Schnittstellen klar und einfach geblieben.

6.2 Erfahrungen mit der SAL

Während die bisherigen Punkte Eigenschaften der SAL zusammenfassen, werden im Folgenden einige Erfahrungen bei der Verwendung der SAL und auf Systemebene wiedergegeben.

18. Der Auftraggeber einer Dienstleistung bzw. eines Workflows muss über entsprechendes Wissen um die Dienstleistung bzw. Workflow verfügen. Zur Vermeidung unnötigen Implementierungs- und Wartungsaufwandes muss die Wahl der Dienstleistungen und Workflows und deren Inanspruchnahme bedacht erfolgen.
19. Auch die Lebensdauer und der Kontext, in dem Informationsobjekte verfügbar sein müssen, sollen bei der Systemarchitektur berücksichtigt werden, um zu vermeiden, dass durch eine ungeschickte Wahl von Schnittstellen der Verwaltungsaufwand für Informationsobjekte erhöht wird.
20. Der Logische Klient ist sehr gut für prozedural beschreibbare Abläufe geeignet, hat aber Schwächen bei Problemen, die besser regelbasiert erfasst und gelöst werden können. Er lässt sich aber um regelbasierte und soziale Agenten erweitern und würde dadurch an Leistungsfähigkeit gewinnen.
21. Es war im Rahmen der zur Verfügung stehenden Mittel nicht möglich, Workflows automatisch zu erstellen. Dies würde eine genauere semantische Modellierung der Dienstleistungen erfordern, auf deren Basis der Strategie-Service gültige Workflows finden

kann. Ferner müsste die Ontologie erweitert werden, um eine Kommunikation über Dienstleistungen und Workflows zu ermöglichen. In der KFÜ erfolgte die Erstellung und Wartung der Workflows von Hand, was mit Aufwand verbunden ist.

22. Die Trennung von Daten und Metadaten hat sich bewährt.

23. In der Taxonomie Daten – Informationen – Wissen deckt die KFÜ eher die untere Ebene der Daten ab. Eine Weiterentwicklung auf höheren Ebenen hätte wahrscheinlich gute Chancen, in der Praxis gebraucht, nachgefragt, finanziert und verwendet zu werden.

24. Die Entwicklung einer Ontologie ist zwar mühsam, aber auch sehr hilfreich. In komplexeren Systemen lohnen sich wissensbasierte Ansätze.

25. Das komplexe System der Kernreaktor-Fernüberwachung Baden-Württemberg (KFÜ) konnte mit Hilfe des Dienstleistungskonzepts zufriedenstellend umgesetzt werden.

6.3 Anmerkungen zur Entwicklung von Agentensystemen

Agenten und Multi-Agenten-Systeme sind ein starker Modetrend der Software-Entwicklung. Das hat nicht nur positive Seiten, gelegentlich wird auch das Wort "Hype" mit seinen Assoziationen zu Übertreibung, Hektik und Unzuverlässigkeit gewählt. Dies liegt möglicherweise daran, dass das Konzept der Agenten stark an menschliches Handeln und menschliche Vernunft angelehnt ist. Bei der Umsetzung oder der Verwendung von Agentensystemen ergeben sich zwangsläufig Diskrepanzen zwischen dem Machbaren und dem Wünschenswerten bzw. auf Grund der bei Agentensystemen verwendeten Analogie zum menschlichen Handeln als gelöst Angenommenen. Diese schmerzliche Erfahrung mussten wir bei der Entwicklung des Logischen Klienten auch machen. Mehr als einmal musste festgestellt werden, dass zur Lösung eines als einfach eingeschätzten oder gar nicht erkannten Problems viel grundlegende Arbeit erforderlich war.

Die Entwicklung eines Agentensystems ist zwar eine spannende und lehrreiche, aber auch riskante Aufgabe, insbesondere wenn mit der Anwendung so hohe Anforderungen wie in der KFÜ verbunden sind. Man muss sich bewusst sein, dass dabei erhöhte Ansprüche an Ontologien und Werkzeugen gelten, von denen viele erst entwickelt werden müssen. Es lohnt sich, vor der Implementierung eines eigenen Agentensystems kritisch zu prüfen, welche Voraussetzungen tatsächlich erfüllt sind, was Wunschdenken ist, und welcher Nutzen vom Einsatz von Agentensystemen realistischerweise zu erwarten ist. Klar ist, dass der Einsatz von Agentensystemen ein großes, wenn auch nicht einfach zu realisierendes Potential hat. Das Dienstleistungskonzept ist in diesem Zusammenhang eher als Grundlage zu sehen. Es kann seine Stärken schon bei wesentlich einfacheren Systemen voll ausspielen.

* * *

Eine Absicht dieser Promotionsarbeit ist die Weitergabe von Wissen und Erfahrungen, eine andere die Bereitstellung von Lösungen. Der Leser, der vor ähnliche Probleme gestellt ist und im Dienstleistungskonzept eine interessante Lösungsmöglichkeit sieht, sei ausdrücklich darauf hingewiesen, dass die SAL, aber auch Teile des Logischen Klienten als Quellcode verfügbar sind. Man wende sich entweder unter Axel.Grohmann@gmx.net an den Autor oder unter Fritz.Schmidt@ike.uni-stuttgart.de an die Abteilung Wissensverarbeitung und Numerik des Instituts für Kernenergetik und Energiesysteme der Universität Stuttgart.

7 Literaturverzeichnis

[1] Kopetzky, Roland (2000): Neue Methoden zur Modellierung komplexer technischer Systeme und ihre Anwendung im Rahmen der Kernreaktor-Fernüberwachung Baden-Württemberg. Eingereichte Dissertation an der Universität Stuttgart.

[2] Coplien, James O. (1994): Advanced C++ Programming Styles and Idioms. Addison-Wesley, Reading (Massachusetts) et al.

[3] Gamma, Erich et al. (1996): Entwurfsmuster. Elemente wiederverwendbarer objektorientierter Software. Deutsch Übersetzung von Dirk Riehle. Addison-Wesley Bonn et al.

[4] GregoryF. Rogers (1997): Framework-Based Software Development in C++. Prentice Hall, London et al.

[5] Nichols, Bradford et al. (1996): Pthreads Programming. O'Reilly & Associates, Cambridge et al.

[6] Weigele, Michael (1997): Berechnung der nassen Deposition von Spurenstoffen im Rahmen des Notfallschutzes. Dissertation an der Universität Stuttgart, IKE 4-146.

[7] Schweizer, Stefan (1995): Modellbasierte Analyse und Bewertung von meteorologischen Meßdaten im Hinblick auf ihre Eignung für Ausbreitungsrechnungen im Rahmen des Notfallschutzes. Dissertation an der Universität Stuttgart, IKE 4-144.

[8] Atomgesetz mit Verordnungen und einer Einführung von Eberhard Ziegler (1997). 20. Aufl. Nomos Verlagsgesellschaft, Baden-Baden.

[9] Bundesministerium für Umwelt, Naturschutz und Reaktorsicherheit (Hg.) (1995): Leitfaden für den Fachberater Strahlenschutz der Katastrophenschutzleitung bei kerntechnischen Notfällen. 2. Aufl. (=Veröffentlichungen der Strahlenschutzkommission, Bd. 13). Gustav Fischer Verlag, Stuttgart, Jena, New York.

[10] R. Mayer-Föll, A. Jaeschke (Hrsg., 1995): Projekt GLOBUS - Konzeption und prototypische Realisierung einer aktiven Auskunftskomponente für globale Umweltsachdaten im Umweltinformations-system Baden-Württemberg Phase II. Wissenschaftliche Berichte FZKA 5700.

[11] R. Mayer-Föll, A. Jaeschke (Hrsg., 1996): Projekt GLOBUS - Konsolidierung der neuen Systemarchitektur und Entwicklung erster Produktionssysteme für globale Umweltsachdaten im Umweltinformationssystem Baden-Württemberg Phase III. Wissenschaftliche Berichte FZKA 5900.

[12] R. Mayer-Föll, A. Jaeschke (Hrsg., 1997): Projekt GLOBUS - Umsetzung der neuen Systemarchitektur und Entwicklung weiterer Produktionssysteme für globale Umweltsachdaten im Umweltinformationssystem Baden-Württemberg Phase IV. Wissenschaftliche Berichte FZKA 6000.

[13] R. Mayer-Föll, A. Jaeschke (Hrsg., 1998): Projekt GLOBUS - Multimediales Recherchieren und Verar-beiten von globalen Umweltsachdaten im Umweltinformationssystem Baden-Württemberg Phase V. Wissenschaftliche Berichte FZKA 6250.

[14] R. Mayer-Föll, A. Jaeschke (Hrsg., 1999); Projekt GLOBUS – Von Komponenten zu vernetzten Systemen für die Nutzung globaler Umweltsachdaten im Umweltinformationssystem Baden-Württemberg und anderen fachübergreifenden Anwendungen Phase VI. Wissenschaftliche Berichte FZKA 6410.

[15] Schöckle, Martin (1999): Modellierung und Simulation komplexer technischer Systeme bei Verwendung des objektorientierten Paradigmas. Dissertation zur Erlangung der Doktorwürde. Institut für Kernenergetik und Energiesysteme (IKE), Universität Stuttgart.

[16] Wooldridge, Michael (1999): Intelligent Agents. – In: In: Gerhard Weis (ed.): Multiagent Systems. A Modern Approach to Distributed Artificial Intelligence. MIT Press, Cambridge (Massachusetts), London, pp. 27 - 78.

[17] Lange, Danny B. (1998): Mobile Agents: Environments, Technologies and Applications. – In: PAAM98. Proceedings of the Third International Conference on the Practical Application of Intelligent Agents and Multi-Agent Technology. The Practical Application Company, London, S. 10-14.

[18] Huhns, Michael N. (1998): Agent Foundations for Cooperative Information Systems. – In: PAAM98. Proceedings of the Third International Conference on the Practical Application of Intelligent Agents and Multi-Agent Technology. The Practical Application Company, London, S. 1-9.

[19] Satapathy, Goutam and Soundar R.T. Kumara (1999): Object Oriented Design based Agent Modeling. – In: PAAM99. Proceedings of the Fourth International Conference on the Practical Application of Intelligent Agents and Multi-Agent Technology. The Practical Application Company, London, S. 143-162.

[20] Grohmann, Axel and Roland Kopetzky (1998): An Agent-Enhanced Architecture for the Logical Client in an Environment Information System. - In: PAAM98. Proceedings of the Third International Conference on the Practical Application of Intelligent Agents and Multi-Agent Technology, pp. 317 – 330.

[21] Grohmann, Axel and Roland Kopetzky (1998): Dynamic Process Modelling and Communication in Environment Information Systems of the Third Generation. - In: Angel Pasqual del Pobil, José Mira and Moonis Ali (Eds.): Tasks and Methods in Applied Artificial Intelligence. 11th International Conference on Industrial and Engineering Applications of Artificial Intelligence and Expert Systems IEA-98-AIE. Proceedings, vol. II, pp. 838 – 848.

[22] Grohmann, Axel and Roland Kopetzky (2000): An Agent-Oriented Information System for Simulating the Dispersion of Radioactive Particles. – In: Jens Pohl and Thomas Fowler IV (eds.): Advances in Computer-Based and Web-Based Collaborative Systems.

Preconference Proceedings of Intersymp 2000, 12th International Conference on Systems Research, Informatics and Cybernetics.

[23] De Marco, Kurt (1998): Konzeption und prototypische Umsetzung der Entscheidungskomponente des Strategie-Service auf Basis eines regelbasierten Systems. IKE 4-D-215, Universität Stuttgart.

[24] D. Hollingsworth (1995): The Workflow Reference Model. In: Workflow Management Coalition (Hrsg.): Document TC00-1003, Draft 1.1.

[25] Rumbaugh, James, Ivar Jacobson und Grady Booch (1999): The Unified Modeling Language Reference Manual. Addison-Wesley, Reading (Massachusetts) et al.

[26] Fowler, Martin und Kendall Scott (1999): UML Distilled. Addison-Wesley, Reading (Massachusetts) et al.

[27] Object Management Group (2000): Unified Modelling Language Resource Page. http://www.omg.org/technology/uml/, gesehen am 28.9.2000.

[28] Weis, Gerhard (Hg.) (1999): Multiagent Systems. A Modern Approach to Distributed Artificial Intelligence. MIT Press, Cambridge (Massachusetts), London.

[29] Object Management Group (1998): The Common Object Request Broker: Architecture and Specification. http://www.omg.org/publications.htm, gesehen am 27. 9. 2000.

[30] Redlich, Jens-Peter (1996): CORBA 2.0. Praktische Einführung für C++ und Java. Addison-Wesley Bonn et al.

[31] Schulze, Wolfgang (2000): Workflow-Management für CORBA-basierte Anwendungen. Systematischer Architekturentwurf eines OMG-konformen Workflow-Management-Dienstes. Springer Berlin Heidelberg New York.

[32] Sun Microsystems (1999): Java 2 Plattform Enterprise Edition Specification, Version 1.2. http://java.sun.com/j2ee/download.html, gesehen am 27. 9. 2000.

[33] Haas, Stephan de (2000): E-business-Anwendungen mit J2EE. – In: OBJEKTspektrum 3/2000, S. 14 – 24.

[34] Box, Don (1999): COM. Microsofts Technologie für komponentenbasierte Softwareentwicklung. Addison-Wesley, Reading (Massachusetts) et al.

[35] Eddon, Guy und Henry Eddon (1998): Inside Distributed COM. Microsoft Press, München.

[36] Göbl, Wolfgang (2000): Der Komponenten-Hype. – In: OBJEKTspekturm 3/2000, S. 34-38.

[37] Szyperski, Clemens (1998): Component Software. Beyond Object-Oriented Programming. Addison Wesley, Harlow et. al.

[38] Stal, Michael (2000): Reich der Mitte. Die Komponententechnologien COM+, EJB und "CORBA Components". – In: OBJEKTspektrum 3/2000, S. 26-33.

[39] Huhns, Michael N. and Larry M. Stephens (1999): Multiagent Systems and Societies of Agents. – In: Gerhard Weis (ed.): Multiagent Systems. A Modern Approach to Distributed Artificial Intelligence. MIT Press, Cambridge (Massachusetts), London, pp. 79-120.

[40] Pohl, Kym Jason (2000): A Inference Engine-Based Subscription Service. – In: Jens Pohl und Thomas Fowler IV: Advances in Computer-Based and Web-Based Collaborative Systems. Preconference Proceedings of InterSymp 2000, 12th International Conference on Systems Research, Informatics and Cybernetics, pp. 85-90.

[41] Foundation for Intelligent Physical Agents (FIPA). http://www.fipa.org, gesehen am 21. 9. 2000.

[42] Foundation for Intelligent Physical Agents (1998): FIPA 97 Specification, Version 2.0 (Part 2) - Agent Communication Language, Geneva, Switzerland.

[43] Labrou, Y. and T. Finin (1997): A Proposal for a new KQML Specification, Computer Sci-ence and Electrical Engineering Department, University of Maryland Baltimore County. http://www.cs.umbc.edu/~finin/papers/kqml97.pdf, gesehen am 28. 9. 2000.

[44] Gensereth, M. and R. Fikes (1998): Knowledge Interchange Format. Stanford Logic Report Logic-92-1, Stanford Univ.; http://logic.stanford.edu/kif/kif.html; Gesehen am 28.9.2000

[45] Collaborative Agent Design Center (CADRC): http://www.cadrc.calpoly.edu, gesehen am 28.9.2000.

[46] Meyers, Scott (1996): More Effective C++. 35 New Ways to Improve Your Programs and Designs. Addison-Wesley, Reading (Massachusetts) et al.

[47] Dumpleton, Graham (1993-2000): The OSE Environment. http://www.dscpl.com.au/, Gesehen am 20. 9. 2000.

[48] Breymann, Ulrich (1998): Komponenten entwerfen mit der C++ STL. Addison-Wesley, München et al.

[49] AT & T Laboratories (2000): omniORB. Free High Performance CORBA 2 ORB. http://www.uk.research.att.com/omniORB, gesehen am 28. 9. 2000.

[50] Simon, Errol (1996): Distributed Information Systems. From Client/Server to Distributed Multimedia. Mc Graw-Hill, Cambridge.

[51] Riley, Garry (1999): CLIPS. A Tool for Building Expert Systems. http://www.ghgcorp.com/clips/CLIPS.html, gesehen am 28. 9. 2000.

[52] Giarratano, Joseph and Gary Riley (1993): Expert Systems. Principles and Programming. Second Edition. PWS, Boston, MA.

[53] Pohl, Jens (2000): Adapting to the Information Age. – In: Jens Pohl und Thomas Fowler IV: Advances in Computer-Based and Web-Based Collaborative Systems. Preconference Proceedings of InterSymp 2000, 12th International Conference on Systems Research, Informatics and Cybernetics, pp. 9-25.