Universität Trier

Der Smart Data Server

Neue Konzepte und praktische Anwendung einer Middleware-Architektur

Dissertation
zur Erlangung des akademischen Grades
des Doktors der Naturwissenschaften
am Fachbereich IV der Universität Trier

vorgelegt von

Diplom-Informatiker
Uwe Roth

im Oktober 2003

Disputation am 30.Juni 2004

Gutachter: Prof. Dr. sc. Christoph Meinel (Universität Trier)

Prof. Dr. Peter Sturm (Universität Trier)

Datum der Disputation: 30. Juni 2004

Vorwort

Wenn Sie Überlegungen mit Taten kombinieren,
werden Sie gute Ergebnisse erzielen.

SPRUCH AUS EINEM GLÜCKSKEKS VOM 30.08.2003

Die vorliegende Dissertation ist im Rahmen meiner Tätigkeit als wissenschaftlicher Mitarbeiter am Institut für Telematik in Trier entstanden. Dieses Institut beschäftigte sich mit Zukunftstechnologien im Internet-Umfeld. Ein Teilbereich dieser Zukunftstechnologien ist Forschungsschwerpunkt dieser Arbeit: der Bereich der Middle-Tier-Architekturen insbesondere der Middleware-Architekturen. Middleware-Architekturen sind Mittler zwischen Informationsanbietern und Informationskonsumenten also beispielsweise Datenbanken und spezialisierte Anwendungen.

Für das Zustandekommen dieser Arbeit möchte ich mich bei Prof. Dr. Christoph Meinel bedanken, der mir die Möglichkeit zur Promotion während meiner Tätigkeit am Institut für Telematik und später an der Universität Trier gegeben hat und mich inhaltlich während der vergangenen Jahre begleitete.

Schließlich möchte ich noch einigen Personen danken, die mir in der Zeit meiner Promotion mit Rat und Tat zur Seite standen. Zunächst möchte ich den ehemaligen Mitarbeitern des Instituts für Telematik für die wunderbare Zusammenarbeit danken, insbesondere Andreas Heuer und Frank Losemann. Besonders möchte ich Ernst-Georg Haffner danken, ohne den die Entwicklung des Smart Data Servers nicht möglich gewesen wäre. Die fruchtbaren Diskussionen haben den Weg des SDS maßgeblich beeinflusst. Mein besonderer Dank geht auch an Michael Schmitt, der mit seiner konstruktiven und mich teilweise an den Rand des Wahnsinn treibenden Kritik, sehr zur Verbesserung des vorliegenden Dokumentes beigetragen hat.

Zum Schluss noch einige persönliche Bemerkungen:

Ich möchte meinen Eltern und meinem Bruder Jörg für Ihr Vertrauen in mich und ihre Liebe danken. Ich liebe euch!

Bea, du bist eine wunderbare Freundin. Ich bin froh, dass es dich gibt!

Iris, meine Liebe! Die Zeit mit dir ist einfach wunderbar. Du machst mich zu einem glücklichen Menschen. Ich liebe dich!

Der Text dieser Abhandlung entspricht den Regeln der neuen deutschen Rechtschreibung.

Folgende Programme wurden bei der Erstellung dieses Dokumentes verwendet:

- Der Text wurde mit *LaTeX2e* (*MiKTeX 2.4*[1]) unter Verwendung des Tools *TeXnicCenter 1Beta6.21*[2] erstellt.
- Grafiken wurden mit *PowerPoint 2000* erstellt, als WMF-Dateien exportiert und mittels *WMF2EPS 1.32*[3] in EPS-Dateien konvertiert.
- Als Versionierungssystem wurde auf der Server-Seite *CVSNT 2.04*[4] sowie auf der Client-Seite *WinCvs 1.3Beta13* [5] eingesetzt.

Ich möchte allen an den erwähnten Programmen Beteiligten herzlich für ihre Arbeit danken.

[1] `http://www.miktex.org`
[2] `http://www.toolscenter.org`
[3] `http://www.wmf2eps.de.vu`
[4] `http://www.cvsnt.org/`
[5] `http://www.wincvs.org`

Inhaltsverzeichnis

Abbildungsverzeichnis

Tabellenverzeichnis

Code-Verzeichnis

1 Einführung

Jedes Wort ist ein wahres Juwel...
was mich beunruhigt, ist nur die Reihenfolge, in der sie hier aneinander gefügt wurden.

TERRY JONES

Die Geschichte verteilter Systeme und Anwendungen hat die unterschiedlichsten Technologien hervorgebracht: Client-Server-Architekturen, Peer-To-Peer-Netzwerke und Komponentensysteme sind nur einige Vertreter. Die vorliegende Arbeit ist im Bereich der Middleware-Architekturen angesiedelt, einem zur Zeit sehr stark beachteten Zweig der verteilten Anwendungen. Als Mittler zwischen den Anwendungen auf der einen Seite sowie Datenbanken, eMail-Systemen oder weiterer Servern auf der anderen Seite, schlägt sie die Brücke in heterogenen IT-Landschaften. Sie verbirgt Details und Änderungen dieser IT-Umgebungen und schafft gleichzeitig einen transparenten Zugriff auf sie.

Bengel (2002) definiert den Begriff *Middleware* folgendermaßen:

> *Die Middleware ist eine Softwareschicht, welche auf Basis standardisierter Schnittstellen und Protokolle Dienste für eine transparente Kommunikation verteilter Anwendungen bereitstellt.*
> *Middlewaredienste stellen eine Infrastruktur für die Integration von Anwendungen und Daten in einem heterogenen und verteilten Umfeld zur Verfügung.*
> *Middleware wird auch als Verteilungsplattform oder Verteilungsinfrastruktur bezeichnet.*

Nach dieser Definition lassen sich zwei Aspekte für Middlewaren herausarbeiten: Die Middleware agiert einerseits als *Diensteanbieter* gegenüber einem Client zur Entwicklung verteilter Anwendungen. Ein Zugriff auf den entfernten Dienst vollzieht sich meist in Form von entfernten Funktionsaufrufen, Objekt- oder Komponentenzugriffen. Andererseits agiert die Middleware auch als *Infrastruktur-Anbieter* gegenüber den Diensten, mit der auf Server-Interna sowie auf das Umfeld der Middleware zugegriffen werden kann. Der Zugriff auf diese Infrastruktur kann auch als Zugriff auf einen Server-internen Dienst verstanden werden. Der Begriff „Dienst“ kommt somit in zwei verschiedenen Kontexten vor, je nachdem, ob man von einem externen oder von einem internen Zugriff auf die Middleware-Infrastruktur spricht.

1.1 Fokus der Arbeit

Die Forschung hat viele Technologien im Middleware-Umfeld hervorgebracht und setzt bei den Zielen unterschiedliche Schwerpunkte. Die zuvor beschriebene Sicht auf Middleware hat den Entwickler im Fokus und soll ihn bei der Erstellung von Anwendungen und Lösungen unterstützen (*entwicklerzentrierte Sichtweise*). Aus diesem Grund stehen hier die Schnittstellen zum Server und zur Server-Infrastruktur sowie die Einbettung der Dienste in die Middleware im Vordergrund. Der interne Aufbau sowie die inneren Datenflüsse der Middleware dienen nur der Erfüllung dieser Ziele.

Eine andere Sichtweise stellt den inneren Aufbau in den Fokus der Forschung (*innerer-Aufbau zentrierte Sichtweise*). Hier ist das Ziel die Schaffung von flexiblen und erweiterbaren Server-Strukturen sowie die Effizienz von internen Abläufen und Datenflüssen. Damit ist eine einfache Anpassung an verschiedene Einsatzumgebungen sowie die Fähigkeit, auf die Leistungsmerkmale von Clients individuell eingehen zu können, möglich.

Die im Rahmen dieser Arbeit entstandene Middleware-Architektur „Smart Data Server (SDS)“ setzt Konzepte um, die beide Sichtweisen miteinander kombiniert.

Ein Schwerpunkt neben der Entwicklung neuer Konzepte, liegt in der **praktischen Anwendbarkeit** der entwickelten Middleware. Viele der an Universitäten und Forschungseinrichtungen entwickelten Lösungen betrachten einzelne Aspekte nur losgelöst von vorgegebenen Randbedingungen und scheitern bei dem Versuch, unter realen Bedingungen zu bestehen. Andere Forschungsvorhaben erweitern bestehende Strukturen um Fähigkeiten, die in kommerziellen Produkten zum Einsatz kommen oder implementieren alle Details bestehender Standards, ohne dabei zusätzliche Erkenntnisse zu gewinnen oder die Struktur in wesentlichen Punkten zu erweitern.

Im Gegensatz dazu kann der Smart Data Server die Praxistauglichkeit seiner Konzepte unter realen Bedingungen unter Beweis stellen und zeigt gleichzeitig, welche Konstrukte für den praktischen Einsatz tatsächlich notwendig sind.

Üblicherweise folgt auf die Übermittlung von Anfragen an einen Server die Berechnung einer Antwort und deren Rücktransport. Alle drei Schritte erfolgen nacheinander. Ein zweiter Schwerpunkt liegt in der Entwicklung von Mechanismen zur **Abarbeitung von streamingfähigen Anfragen**.

Bei Streaming-Anfragen ist neben der Übermittlung von Anfrageparametern ein kontinuierlicher und in seiner Dimension nicht beschränkter Datenstrom Bestandteil der Anfrage. Der Server startet seine Berechnungen wenn möglich schon vor dem Ende der Datenübermittlung und liefert bei Vorliegen von Teilergebnissen diese an den Client. Die Anfrageübermittlung, die Berechnung der Ergebnisse sowie die Antwortübermittlung kann (in Abhängigkeit der Anfrage) parallel erfolgen. Neben diesem Vorteil kann der für eine Anfrage benötigte Bedarf an Speicherplatz im Server optimiert werden, da nur so viele Daten der

Anfrage im Server vorgehalten werden müssen wie zur Berechnung tatsächlich notwendig sind und nicht etwa die gesamte Anfrage.

Die Entwicklung eines Streaming-Mechanismuses muss entsprechende Möglichkeiten in der Anfragestruktur sowie in der Architektur der Middleware vorsehen. Der Smart Data Server schafft mit der Einführung einer streamingfähigen Anfragestruktur die Voraussetzung für die Verarbeitung solcher Anfragen.

Zusätzlich muss eine Server-Architektur geschaffen werden, die das Abarbeiten von Streaming-basierten Anfragen unterstützt. Hier überschneiden sich die zuvor beschriebene *entwicklerzentrierte Sichtweise* mit der *innerer-Aufbau zentrierten Sichtweise*.

Hier liegt der dritte Schwerpunkt der Arbeit: Die Schaffung einer **flexiblen und erweiterbaren Server-Struktur** sowie die **Effizienz von internen Abläufen und Datenflüssen** unter Berücksichtigung der Anforderung einer streamingfähigen Anfragebearbeitung. Diese Anforderungen werden mit der neu entwickelten Technologie der *inneren Workflow-Programme* erfüllt. Innere Workflow-Programme repräsentieren Netzwerke von hochgradig unabhängigen Serverkomponenten. Durch die Umgestaltung dieser Netze sind neue Serverstrukturen möglich, um neuen Einsatzbedingungen angepasst werden zu können. Neben diesem statischen Netzwerk von Server-Komponenten kann zur Laufzeit eine Pipeline zur Abarbeitung von Datenströmen aufgebaut werden, mit der dann letztendlich die Streamingfähigkeit des Servers hergestellt wird.

All diese zunächst sehr abstrakt definierten Schwerpunkte der Arbeit können erst im Lichte von verwandten Technologien, deren Lösungen und Schwächen, weiter präzisiert werden. Die genauen Zielvorgaben werden aus diesem Grund in Kapitel 4 direkt im Anschluss an den aktuellen Forschungsstand (Teil I) definiert.

1.2 Aufbau der Arbeit

Die vorliegende Arbeit gliedert sich in drei Teile. In Teil I wird der Stand der Technik erläutert. Zunächst werden die Technologien betrachtet, die Middlewaren als Diensteanbieter verwenden, um die gewünschten Dienste zugreifbar zu machen. Dies sind die entfernten Funktionsaufrufe, Objektzugriffe oder Komponententechnologien. Beispiele für solche Technologien werden in Kapitel 2 aufgeführt, unter anderem als Gesamtlösungen, die mehrere Technologien in einer Middleware-Lösung vereinen, aber auch als Lösung für einen einzelnen Teilaspekt (z.B. den entfernten Funktionsaufruf).

Die in Kapitel 2 beschriebenen Technologien behandeln den Teil von Middlewaren, der gegenüber dem Anwender oder Entwickler sichtbar wird. Hierbei handelt es sich um Protokolle oder Nachrichtenformate, Schnittstellen, die Dienste implementieren müssen, um in der Middleware lauffähig zu sein, oder um Verfahren die zeigen wie Dienste in die Middleware eingebunden werden.

Mit diesem Komplex des inneren Aufbaus, der inneren Abläufe und Datenflüsse beschäftigt sich Kapitel 3. Dabei werden nicht nur verwandte Technologien, die direkt im Kontext von Middleware stehen, erläutert, sondern auch Grenzgebiete, die auf Middlewaren übertragbar sind. So können Erkenntnisse aus Protokoll-Frameworks durchaus auf Middleware-Architekturen übertragen werden, geht es doch bei beiden um die Abarbeitung von Datenflüssen.

Mit dem Abschluss von Teil I ist das Umfeld beschrieben, in dem sich diese Arbeit befindet. Teil II geht auf die Konzepte und Entwicklung der Smart Data Server Middleware-Plattform ein. Ausgehend von den Erkenntnissen der aktuellen Forschung werden die Gründe für die Eigenentwicklung von neuen Konzepten einer Middleware-Architektur durch das Aufzeigen von Problemen oder Lücken herausgearbeitet. Die sich daraus ergebenen Ziele werden in Kapitel 4 motiviert und weiter präzisiert. In den folgenden Kapiteln werden diese Ziele dann aufgegriffen und umgesetzt.

Die Kommunikation des Smart Data Servers zwischen Client und Server, also der entfernte Aufruf von Funktionen, spielt eine wichtige Rolle und wird in Kapitel 5 erläutert. Sie vollzieht sich auf zwei Ebenen: Die Anfrage und Antwort werden in einem speziellen streamingfähigen Nachrichtenformat kodiert und unter Zuhilfenahme eines neu entwickelten Protokolls zwischen Client und Server ausgetauscht. Erläuterungen zur Anwendung von entfernten Aufrufen, insbesondere dem Spezialfall des Datenstreamings, schließen dieses Kapitel ab.

Kapitel 6 beschreibt die weiteren Aktivitäten des Servers beim Empfang einer Anfrage durch den Client. Dies umfasst den Aufbau einer Sitzung, die Autorisation/Authentifikation des Clients und den wichtigen Bereich, der sich mit dem Auffinden von Modulen und Funktionen beschäftigt. Das Kapitel endet mit serverinternen, zeitgesteuerten Anfragen.

Als Middleware stellt die Plattform gegenüber seine Komponenten eine Infrastruktur zur Verfügung, die in Kapitel 7 erläutert wird. Drei Dienste werden erläutert: Ein Basisdienst stellt das Logging von Informationen dar, mit dem alle aktiven Teile des Servers Mitteilungen oder Fehler protokollieren können. Als Datenserver nimmt der transparente Zugriff auf die heterogene Datenbanklandschaft eine weitere wichtige Rolle ein. Zusätzlich besteht die Fähigkeit des Server, eMails zu senden und zu empfangen.

Mit der Optimierung der internen Abläufe und Datenflüsse in Form von Workflow-Programmen in Kapitel 8 schließt Teil II ab. Zunächst werden auf einer abstrakten Ebene die Grundstruktur und die Elemente dieser Technologie erläutert, die anschließend um die Fähigkeit zum Pipelining von Daten erweitert wird. Aufbauend auf dieser neuen Technologie folgt eine Beschreibung der Umsetzung im Smart Data Server.

Der Smart Data Server ist nicht nur ein theoretisches Konzept, sondern konnte als Implementierung seine Praxistauglichkeit beweisen. Teil III führt in Kapitel 9 zunächst einige Beispiele aus der Praxis auf und zeigt die Flexibilität des Smart Data Servers an konkreten

Problemfällen auf. Das Kapitel 10 folgt mit Laufzeittests, die den Smart Data Server mit zwei weiteren Technologien vergleichen.

Die Arbeit endet in Kapitel 11 mit einer Zusammenfassung und einem Ausblick über mögliche zukünftige Erweiterungen.

Teil I

Stand der Technik

2 Entfernte Funktionen, Objekte und Komponenten

Jeder Zuwachs an Technik bedingt,
wenn damit ein Zuwachs und nicht eine Schmälerung des menschlichen Glücks verbunden sein soll,
einen entsprechenden Zuwachs an Weisheit.

BERTRAND RUSSELL

In Computer-Netzwerken können Rechnersysteme bezüglich ihrer primären Aufgaben in zwei Gruppen unterteilt werden. Auf der einen Seite stehen die Server als Dienste-Anbieter, auf der anderen Seite die Clients als Dienste-Konsumenten.

Ein möglicher Dienst eines Servers ist die Übernahme der Funktionsausführung, die ein Client in seiner Anwendung verwendet. Um diesen Vorgang transparent zu halten, also so zu gestalten, dass sich die Anwendung der Funktion nicht von einer Ausführung innerhalb der lokalen[1] Applikation unterscheidet, wurden verschiedene Basis-Technologien entwickelt und mit der Zeit immer weiter verfeinert.

Dieses Kapitel beschäftigt sich mit diesen Technologien: die einfachen entfernten Funktionsaufrufe, die Zugriffe auf entfernte Objekte und Komponenten sowie darauf aufbauende Architekturen.

2.1 SUNs Remote Procedure Call (RPC)

RPC (Remote Procedure Call) ist eine Technologie zum Aufruf entfernter Prozeduren.

RPC wurde ursprünglich von XEROX in den 70er Jahren beschrieben und 1981 als Produkt *Courier* eingeführt[2]. SUNs Version *ONC (Open Network Computing)* wurde 1985 der Öffentlichkeit vorgestellt[3]. Durch die Offenlegung des Programmcodes entwickelte sich diese Version schnell zum Standard.

RPC hat eine weite Verbreitung auf allen Unix-Varianten; der Mechanismus wurde jedoch auch auf andere Betriebssysteme portiert (z.B. System/370 VM, MVS, VMS).

[1] Adressraum der Applikation auf einem Rechnersystem.
[2] Xerox Corporation (1981); Birrel und Nelson (1984)
[3] Sun Microsystems (1988)

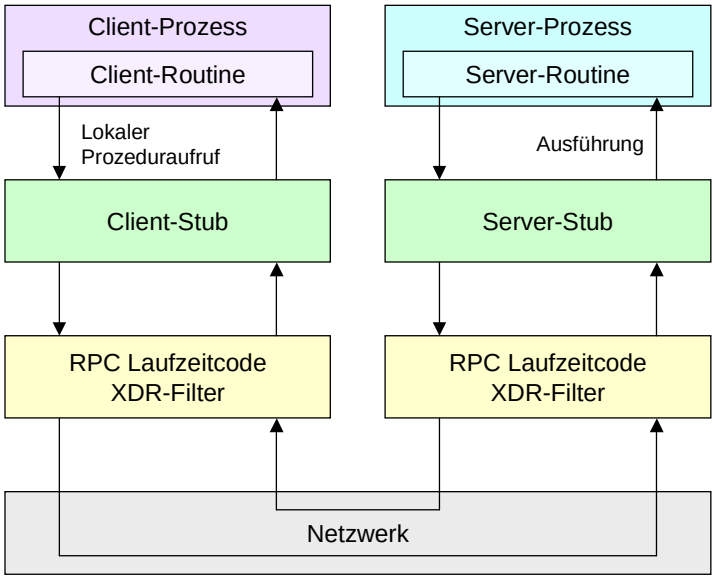

Abbildung 2.1: Entfernter Prozeduraufruf mit RPC

Bei Aufrufen von entfernten Prozeduren über RPC sind zwei Prozesse beteiligt (siehe Abb. 2.1 nach Roth, 2000a), ein Client-Prozess, der den Aufruf initiiert sowie ein Server-Prozess, der die Ausführung der Prozedur übernimmt. Da der Aufruf transparent durchgeführt werden soll, ruft die Anwendung lokal eine Stellvertreter-Prozedur auf, den *Client-Stub*. Dieser übernimmt die Aufgabe, die Anfrage über das Netzwerk an den Server zu übertragen. Die dabei verwendeten Funktionen werden als *RPC-Laufzeitcode* bezeichnet.

Die Übergabeparameter werden vor der Übertragung in ein rechnerunabhängiges Datenformat *XDR (eXternal Data Representation)*[4] überführt. Auf der Server-Seite nimmt der *Server-Stub* die Parameter entgegen, führt den lokalen Prozedur-Aufruf aus und verpackt der Ergebnisdaten, die dann auf dem umgekehrten Weg zum Client übertragen werden.

Beim Aufruf der entfernten Prozedur wird ein 3-Tupel als eindeutiger Identifikator der Prozedur verwendet: *Programm-Nummer*, *Prozedur-Nummer* sowie *Versionsnummer*. Dies erlaubt es, auf der Server-Seite gleiche Funktionen in verschiedenen Versionen bereitzuhalten und damit Kompatibilitätsprobleme zu vermeiden.

Damit der RPC-Mechanismus funktionieren kann, wird auf der Server-Seite ein RPC-Dämon benötigt, der *RPC-Port-Mapper*. Er wird bei einem entfernten Prozedur-Aufruf zunächst angesprochen um den Port zu ermitteln, auf dem der eigentliche Server-Prozess erreichbar ist. Der Client-Prozess kann dann ohne Beteiligung des Port-Mappers die Kom-

[4] Sun Microsystems (1987)

munikation zu dem Server-Prozess aufbauen oder aber die Anfrage indirekt über den Port-Mapper weiterleiten lassen.

Um RPC-Anwendungen entwickeln zu können, werden zunächst die entfernten Prozedur-Aufrufe in einer C-ähnlichen Programmiersprache beschrieben, der *RPCL (RPC Language)*. Mittels dieser Sprache werden die Prozedur-Köpfe sowie selbst definierte Datentypen beschrieben. Das Programm `rpcgen` erzeugt dann den Client-Stub, den Server-Stub sowie Header-Dateien mit den notwendigen Deklarationen, auf deren Basis die Anwendungen geschrieben werden können.

RPC-Aufrufe können synchron oder asynchron durchgeführt werden. Im ersten Fall ist der Client-Prozess solange blockiert, bis eine Antwort des Servers vorliegt. Im zweiten Fall erhält der Client die Kontrolle unmittelbar nach dem Absetzen der Anfrage. Hier muss der Client-Prozess durch gelegentliches Nachfragen ermitteln, ob das Ergebnis der Anfrage vorliegt.

2.2 Distributed Computing Environment (DCE)

Anfang der 90er Jahre begann die *Open Software Foundation (OSF)* mit der Entwicklung des *Distributed Computing Environment (DCE)*[5]. Der DCE-Standards kann von verschiedenen Software-Herstellern implementiert werden. Durch Dienste und Werkzeuge, die DCE zur Verfügung stellt, können verteilte Anwendungen entwickelt werden.

DCE ist eine Middleware, die zwischen Anwendung und Betriebssystem liegt (siehe Abb. 2.2 aus Laifer, 1998). Es besteht aus einer Reihe von Komponenten, die von unterschiedlichen Software-Herstellern entwickelt wurden:

DCE Threads: Sie ermöglichen die parallele Ausführung von Befehlsabläufen innerhalb eines Prozesses. Diese Funktionalität ist wichtig, da ein auf Antwort wartender RPC-Aufruf normalerweise den aktuellen Prozess blockiert. Um dies zu verhindern, wird zur Initialisierungszeit des Systems ein Pool von Threads generiert, an die dann die RPC-Aufrufe bei Bedarf delegiert werden. Die Implementierung des DCE-Thread-Service entspricht dem POSIX 1003.4a-Standard.

DCE Remote Procedure Call: Diese Komponente ermöglicht entfernte Funktionsaufrufe. Hierzu wird zunächst ein Interface, das die entfernte Funktion mit ihrem Ein- und Ausgabe-Verhalten beschreibt, in der Schnittstellen-Beschreibungssprache *DCE-IDL* [6] definiert. Anschließend wird mittels eines IDL-Compilers ein Client-Stub und ein Server-Skeleton erzeugt. Der DCE-Remote Procedure Call (DCE-RPC) übernimmt mehrere Aufgaben, darunter auch die Authentifizierung von Client und Server

[5] Rosenberry (1992), `http://www.opengroup.org/tech/dce/`
[6] IDL = Interface Definition Language

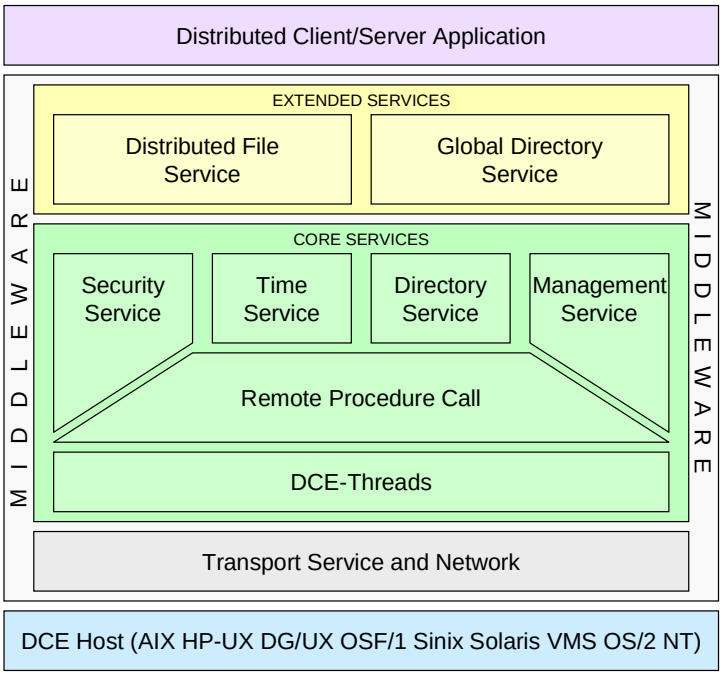

Abbildung 2.2: DCE Architektur

(*authenticated RPC*), die Sicherung der Daten-Integrität und die Verschlüsselung der Daten.

DCE Security Service: Der DCE Security Service hält in einer Datenbank Informationen wie Namen und Passwörter aller beteiligten *Principals* (Benutzer, Rechner, Dienste) bereit. Er ist verantwortlich für die Authentifizierung der Kommunikationsstellen, die Zugriffskontrolle auf Ressourcen mittels *ACLs (Access Control Lists)*, die Integrität der übertragenen Daten sowie die Vertraulichkeit der übertragenen Daten gegenüber Dritten.

DCE Directory Service: Dieser Dienst ermöglicht das Auffinden von Diensten ohne direkte Kenntnis des Servers, der diesen Dienst anbietet. Es wird unterschieden zwischen dem *DCE Cell Directory Service* und dem *DCE Global Directory Service*. Der DCE Cell Directory Service dient dem Auffinden von Diensten innerhalb einer *DCE-Zelle*, einer administrativen Einheit von Principals, die über einen eindeutigen Namen angesprochen werden kann und einen gemeinsamen Security- und Directory-Server besitzt. Mit dem DCE Global Directory Service können Dienste außerhalb der DCE-Zelle aufgefunden werden.

DCE Distributed File Service: Der DCE Distributed File Service ist ein globales Dateisystem über Rechnergrenzen hinweg. Für jeden Rechner ist die Sicht auf das Dateisystem identisch, da der Namensraum für jeden identisch ist. Der Service nutzt den DCE Securtity Service bei der Zugriffskontrolle.

DCE Time Service: Dieser Service dient der Synchronisierung der Rechnerzeiten innerhalb einer DCE-Zelle. Dies ist insbesondere bei der Authentifizierung wichtig, da einige Informationen nach Ablauf einer definierten Zeit ihre Gültigkeit verlieren.

2.3 Entfernte Objekte und Komponenten unter Java

Java[7] wurde ursprünglich unter dem Namen *Oak* innerhalb des *Green Project* bei Sun Microsystems zur Steuerung von unterschiedlichsten Geräten (z.B. TV, Waschmaschine) entwickelt[8]. Da auch bei internetfähigen Anwendungen[9] eine wohldefinierte Menge von Anforderungen an das ausführende Betriebssystem gestellt werden dürfen, bot sich der Einsatz dieses Programmiersprachen-Konzepts auch dort an.

Java ist eine Sprache, die interpretiert wird. Dazu wird das Programm in einen rechner- und betriebssystemunabhängigen *Bytecode* übersetzt. Zur Ausführung dieses Codes wird eine *Java Virtual Machine (JVM)*[10] benötigt, die den Bytecode interpretiert und die Laufzeitumgebung für das Programm zur Verfügung stellt.

Zwei Aspekte von Java erleichtern das Programmieren erheblich. Zum einen besitzt Java keine expliziten Pointer [11], zum anderen wird das Speichermanagement durch die *Garbage Collection* unterstützt, die nicht mehr benötigten Speicher automatisch freigibt.

In Java besteht die Möglichkeit, Objekte in einen Datenstrom zu überführen (*serialisieren*). Nach der Deserialisierung besitzt das Java-Objekt wieder alle Daten und Objektunterstrukturen. Eine Anwendung, die Zugriff auf ein deserialisiertes Objekt besitzt, aber dessen Klasseneigenschaften unbekannt sind, kann mittels der Java-*Reflection*-API diese erfragen und Methoden des Objektes ausführen.

Die Internetfähigkeit stellte Java durch seine *Applets* unter Beweis, kleine Java-Programme, die von einem Web-Server zum Web-Browser übertragen werden und innerhalb der dargestellten HTML-Seite ablaufen. Sie stellen Interaktionsfähigkeiten komplexer Anwendungen über grafische Benutzerschnittstellen zur Verfügung, wie sie mittels HTML oder JavaScript nicht möglich sind.

[7] Sun Microsystems (2001b, 1997); Harold (1999); Roth (2000a); Denninger und Peters (2000); Flanagan (2003); Campione und Walrath (1998)

[8] Lerner u. a. (2000); Byous (1998)

[9] Anwendungen, die über das Internet zu einem Rechner übertragen werden.

[10] Bestandteil einer *Java Runtime Environment (JRE)*, das zusätzlich alle Standard-Java-Klassen, die zur Ausführung von Java-Programmen benötigt werden, bereithält.

[11] Zeiger auf Speicher.

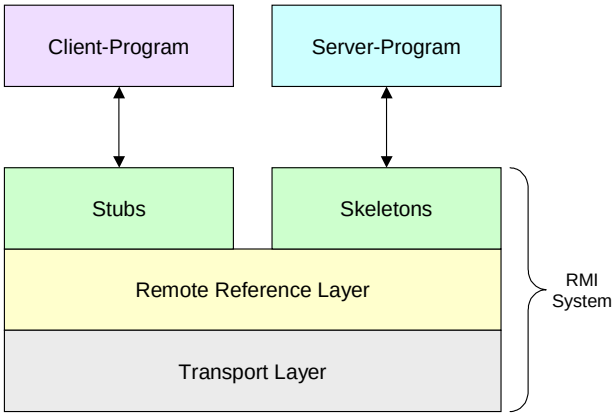

Abbildung 2.3: RMI System

Der Einsatz von Java verlagert sich jedoch immer mehr auf die Server-Seite, unter anderem als *Servlets* innerhalb von Web-Servern oder auch in Form von *Enterprise Java Beans* (siehe Abs. 2.3.2).

2.3.1 Remote Method Invocation (RMI)

Java-Programme können zunächst nur auf Objekte innerhalb derselben JVM zugreifen. Java stellt mit der *Remote Method Invocation (RMI)*[12] eine Technologie zur Verfügung, mit der über die Grenzen der JVM hinweg auf entfernte Objekte zugegriffen werden kann. Dabei ist es nicht von Bedeutung, ob das Objekt in einer anderen JVM auf demselben Rechner liegt oder in einer JVM auf einem entfernten Rechner. Für den Anwendungsentwickler ist der Zugriff transparent, d.h. es handelt sich für ihn um einen lokalen Zugriff auf das Objekt.

Die Ebenen der RMI-Architektur sind in Abbildung 2.3 (aus Roth, 2000a) dargestellt. Die Client-Anwendung greift nicht direkt auf das entfernte Objekt zu, sondern auf ein Stellvertreter-Objekt, das Stub-Objekt. Dieses greift unter Verwendung des *Remote Reference Layers* auf das zugehörige Skeleton-Objekt der entfernten JVM zu, das die Verbindung zur entfernten Objekt-Instanz herstellt. Der Transport Layer basiert auf TCP/IP-Verbindungen und wird in den RMI-Prozess auch dann einbezogen, wenn das entfernte Objekt auf demselben Rechner in einer zweiten JVM liegt.

Die Ebenen-Architektur ermöglicht den Austausch einzelner Ebenen ohne das Gesamtsystem zu beeinflussen. So setzt RMI oberhalb von TCP/IP ursprünglich auf das proprietäre

[12] Sun Microsystems (2002a); Steflik und Sridharan (2000); Harold (1999); Campione und Walrath (1998); Farley (1998); Roth (2000a)

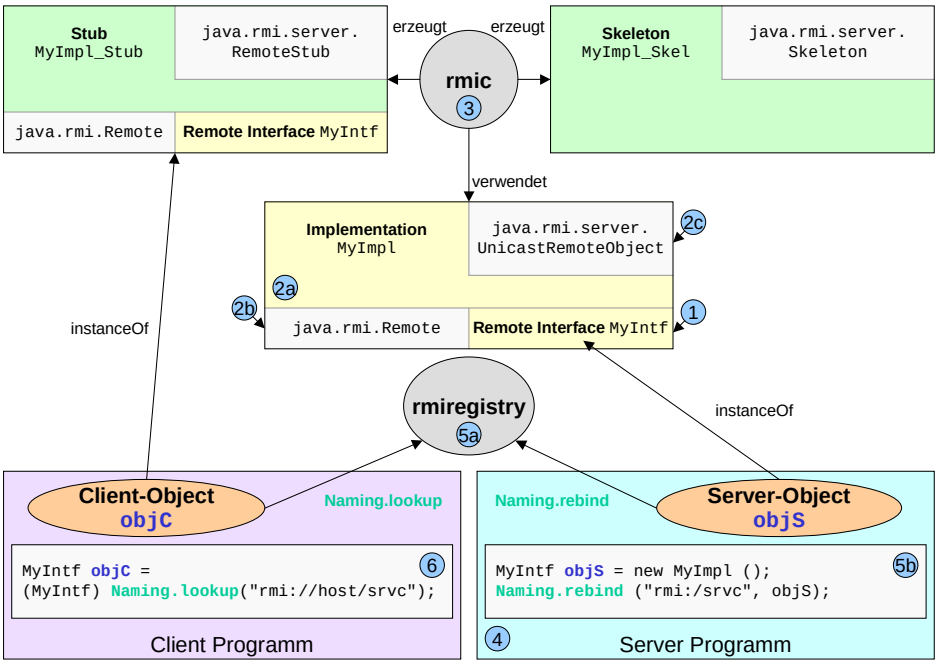

Abbildung 2.4: Entwicklung und Anwendung von RMI

Protokoll *JRMP (Java Remote Method Protocol)* auf; inzwischen ist jedoch auch der Einsatz des Protokolls *RMI-IIOP*[13] möglich. *IIOP (Internet Inter-ORB Protocol)* ist ein standardisiertes Protokoll auf das im Abschnitt *CORBA* (siehe Abs. 2.4) näher eingegangen wird.

Die Entwicklung einer RMI-Anwendung vollzieht sich in mehreren Schritten (siehe Abb. 2.4)[14].

1. Spezifikation des *Remote-Interface*, das alle notwendigen Methodenrümpfe des entfernten Objektes definiert (hier: `MyIntf`).

2. Implementierung der Klasse zu dem Remote-Interface (hier: `MyImpl`).

   Damit ein Objekt der Klasse als mögliches entferntes Objekt erkannt wird, muss es zusätzlich das leere Interface[15] `java.rmi.Remote` implementieren.

   Um den Zugriff letztendlich zu ermöglichen, muss es zusätzlich von der Klasse `java.rmi.server.UnicastRemoteObject` abgeleitet werden.

[13] RMI über IIOP

[14] In der Grafik werden nur die Entwicklungsschritte und Abhängigkeiten von Objekten und Klassen dargestellt, nicht jedoch der Kommunikationsfluss zwischen der Anwendung und dem entfernten Objekt.

[15] Leer bedeutet hier, dass das Interface keine Methoden oder Felder definiert.

3. Nach der Übersetzung des Remote-Interfaces und dessen Implementation werden Stub und Skeleton unter Verwendung des Programms `rmic` erzeugt[16]. Dieser Befehl verwendet als Aufruf-Parameter die Implementation und nicht das Remote-Interface, da eine Klasse mehrere Remote-Interfaces gleichzeitig implementieren kann und dies bei den Stub- und Skeleton-Klassen dann berücksichtigt wird.

4. Das entfernte Objekt benötigt einen Server, der es instanziiert und verwaltet. Diese Aufgabe bleibt dem Programmierer überlassen, der unter Verwendung der Implementation `MyImpl` die entsprechende Anwendung entwickelt, übersetzt und startet.

5. Die Verknüpfung zu einem entfernten Objekt muss über den Umweg eines Namensdienstes (*Naming Service*), der `rmiregistry` (*RMI Registry*), hergestellt werden. Hier werden die Objekte unter eindeutigen Dienstbezeichnungen registriert. Die Server-Anwendung führt nach der Instanziierung des Objekts hierzu eine Registrierung, das *Binding*, durch. Der Aufruf

   ```
   Naming.rebind ("rmi:/srvc", objS);
   ```

   bindet das Objekt `objS` unter dem Namen `srvc` in der Registry. Aus Sicherheitsgründen darf der Host nur der lokale Rechner sein. Es ist also nicht möglich, ein Objekt in der Registry eines entfernten Rechners zu binden.

6. Bei der Entwicklung der Client-Anwendung muss nur das zuvor definierte Remote-Interface `MyIntf` bekannt sein. Über den Vorgang des *Lookup* wird bei der Registry nach dem entfernten Objekt gefragt und als Ergebnis ein Stub-Objekt geliefert. Dieses implementiert das gewünschte Interface und stellt die Verbindung zu dem entfernten Objekt her:

   ```
   MyIntf objC =
   (MyIntf) Naming.lookup ("rmi://host/srvc");
   ```

Neben dem Zugang über Rebind und Lookup können entfernte Objekte auch als Rückgabewerte von Methoden mittels des Objekt-Serialisierungsmechanismus zum Client übertragen werden. So wird üblicherweise nur ein entferntes Objekt in der Registry registriert, das Methoden für den Zugriff auf weitere entfernte Objekte bietet. Auf dem gleichen Weg ist auch die Übertragung von Objekten vom Client zum Server möglich und damit die Entwicklung von Call-Back-Anwendungen.

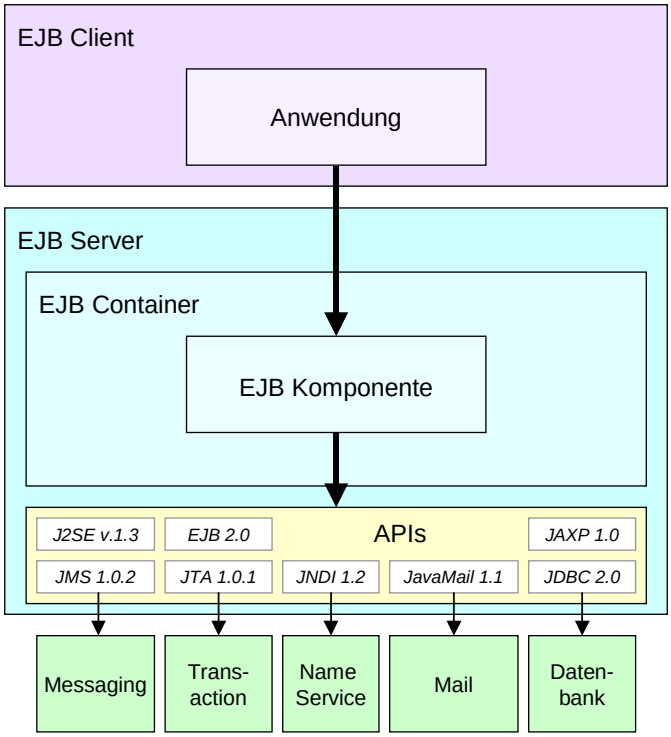

Abbildung 2.5: Enterprise Java Beans

2.3.2 Enterprise Java Beans EJB

Enterprise Java Beans (EJB)[17] ist eine Technologie zur Entwicklung von Business-Anwendungen. Grundlage dieser Technologie sind wiederverwendbare Komponenten, die so genannten *Beans*, die jeweils ein Business-Objekt repräsentieren und damit den Kern der Business-Logik darstellen.

Enterprise Java Beans dürfen nicht mit den *Java Beans* verwechselt werden. Auch dort geht es um wiederverwendbare Komponenten, allerdings liegt die Java Beans-Technologie mehr im Bereich der Entwicklung von Benutzungsschnittstellen. Enterprise Java Beans hingegen sind Server-seitige Komponenten.

EJBs benötigen einen Enterprise Java Beans Server, der für die gesamte Laufzeit eines Beans verantwortlich ist. Durch die Schaffung von *EJB Containern*, in denen ein Bean in-

[16] Ab der Version 1.2 von Java wird das Skeleton nicht mehr benötigt und durch ein allgemein gültiges (anwendungsunabhängiges) Skeleton ersetzt. Dieses stellt die Verbindung zur Objekt-Instanz durch Reflection her.

[17] Sun Microsystems (2001a); Denninger und Peters (2000); Monson-Haefel (2000)

nerhalb eines EJB-Servers eingebettet ist, ergibt sich ein transparenter Zugriff des Servers auf ein Bean sowie umgekehrt für das Bean auf die Server-Infrastruktur (siehe Abb. 2.5). Über diesen Mechanismus können Aspekte der Sicherheit, der persistenten[18] Speicherung von Beans in Datenbanken sowie die Behandlung von Transaktionen berücksichtigt werden, ohne dass diese im Bean kodiert werden müssen. Diese Informationen werden in einer XML-Datei festgelegt, dem *Deployment Descriptor* und zum Zeitpunkt der Installation des Beans in den Bean-Server berücksichtigt.

Die EJB-Spezifikation 2.0 definiert drei verschiedene Arten von Beans:

Entity-Bean: Dieses Bean ist vergleichbar mit einem Datensatz in einer Datenbank. Die Eindeutigkeit eines Datensatzes wird über einen Primärschlüssel garantiert, für den eine Primärschlüsselklasse (PrimaryKey-Klasse) definiert werden muss. Über den Primärschlüssel ist es möglich, dass mehrere Anwendungen gleichzeitig auf das gleiche Bean (denselben Datensatz) zugreifen können. Entsprechend wirkt sich eine Änderung dieses Datensatzes global, also auch unmittelbar auf alle anderen Anwendungen aus.

Session-Bean: Bei der Session-Bean handelt es sich um eine Ressource, die einem bestimmten Client in seiner aktuellen Sitzung exklusiv zur Verfügung steht. Session-Beans stehen nicht für Datensätze, sondern für die eigentliche Geschäftslogik oder die Business-Prozesse. Man kann desweiteren zwei Arten von Session-Beans unterscheiden:

Stateless-Session-Beans speichern keine Daten innerhalb des Beans. Die Grundlage der Berechnungen werden ausschließlich aus den übergebenen Parametern ermittelt.

Stateful-Session-Beans speichern Daten innerhalb des Bean. Wird die Sitzung beendet, z.B. weil der EJB-Server herunter gefahren wird oder weil er abstürzt, so gehen diese Daten verloren.

Message Driven Bean: Diese Art der Beans unterscheidet sich grundlegend von Entity- oder Session-Beans. Sie werden durch einen Client über den Eingang von Nachrichten, die über einen Java Messaning Server (JMS) versendet werden , aktiviert. Es besteht also keine direkte Verbindung zwischen Client und Server. Ansonsten entspricht diese Art von Bean den zustandslosen Session-Beans.

Der Zugriff auf das Business-Objekt im Server erfolgt bei den Entity- und Session-Beans über die RMI-Technologie (siehe Abs. 2.3.1). Zwei Interfaces spielen hier eine wichtige Rolle. Zum einen muss ein Home-Interface definiert werden, das die Logik für das Erzeugen, Löschen und im Falle der Entity-Beans für das Auffinden des Beans beinhaltet, zum anderen das Remote-Interface, das die eigentliche Business-Funktionalität des Bean

[18] persistent = dauerhaft

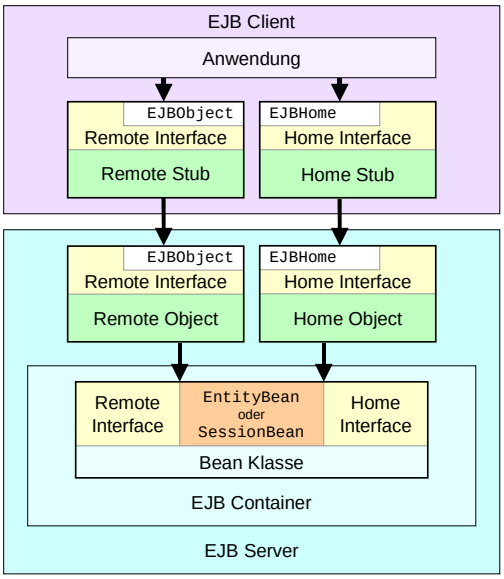

Abbildung 2.6: Entity- und Session-Beans

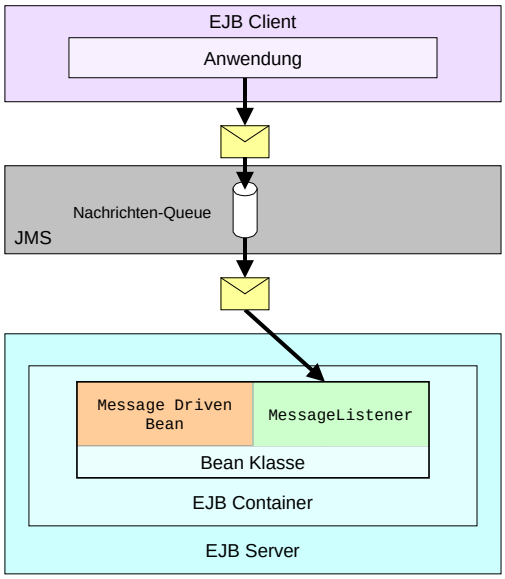

Abbildung 2.7: Message Driven Beans

repräsentiert (siehe Abb. 2.6). Auf der Server-Seite implementieren diese Interfaces zwei Stellvertreter-Objekte, das Home-Objekt und das Remote-Objekt. Beide Objekte stellen die RMI-Brücke zwischen der Client-Anwendung und dem Bean-Objekt im EJB-Container her, das alle Methoden der beiden Interfaces implementieren muss.

Anders sieht dies bei den Message-Driven Beans aus. Hier wird weder ein Home- noch ein Remote-Interface implementiert. Damit fehlen auch die Stellvertreter-Objekte. Stattdessen empfängt das Bean eintreffende und durch den Container weitergeleitete Nachrichten über das `javax.jms.MessageListener`-Interface (siehe Abb. 2.7) .

Um Entity- und Session-Beans in einen EJB-Server einbinden zu können und gleichzeitig über einen Client zugreifbar zu machen, setzt die RMI-Technologie die Existenz von Stub- und Skeleton-Klassen voraus. Im EJB-Fall werden diese Klassen von einem *Container-Tool* erzeugt, das der Hersteller eines EJB-Server zur Verfügung stellen muss (siehe Abb. 2.8). Dieses Tool wird auf der einen Seite mit den Home- und Remote-Interfaces der Bean-Klasse gestartet und im Falle von Entity-Beans zusätzlich mit der Primary-Key-Klasse (die für die Eindeutigkeit von Datensätzen durch die Bestimmung von Primary-Keys sorgt). Im Allgemeinen geschieht dies über eine grafische Benutzungsschnittstelle des Server-Herstellers. Neben den Interfaces und Klassen wird dem Tool der Deployment-Descriptor übergeben, eine XML-Datei, die Struktur-Informationen sowie externe Abhängigkeiten über das Bean liefert.

Auf Grundlage dieser Informationen werden für den Client die Home- und Remote-Stubs, und für den Server die Home- und Remote-Objekte erzeugt. Nach der Erstellung der

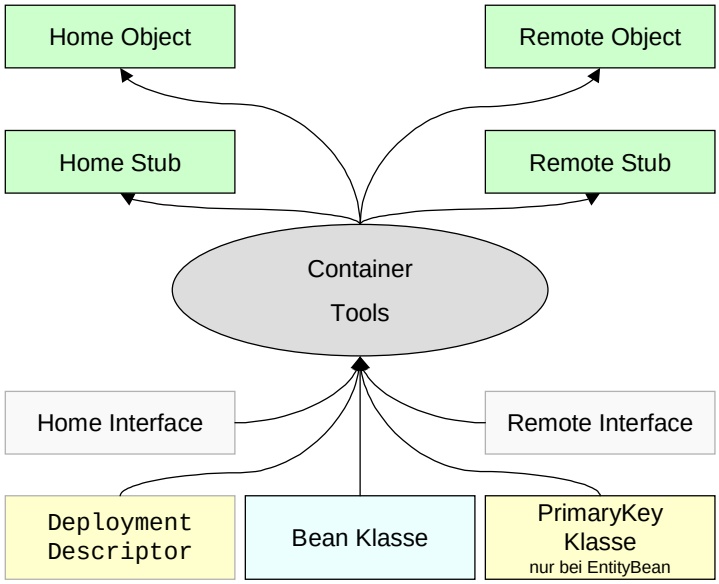

Abbildung 2.8: Erzeugen der benötigten Klassen

notwendigen Objekte kann das Bean in den EJB-Server eingebunden und damit zugreifbar gemacht werden.

Der Zugriff auf Beans muss nicht notwendigerweise von entfernten Rechnern aus durchgeführt werden. Ebenso ist dieser auf demselben Rechner innerhalb der gleichen Java Virtual Machine denkbar. In diesem Falle ist der Zugriff über RMI und die Remote- und Home-Interfaces gegenüber einem lokalen Aufruf wesentlich langsamer. Aus diesem Grunde existieren die *Local Interfaces*. Die Übergabe von Parametern erfolgt in diesem Falle per Referenz und ist somit wesentlich schneller als über den Weg der Objekt-Serialisation in RMI.

Die EJB-Spezifikation schreibt das Vorhandensein einiger APIs vor (siehe Abb. 2.5 auf Seite 17), auf die ein Bean Zugriff haben muss. Weitere APIs können vom Hersteller eines EJB-Servers zur Verfügung gestellt werden, reduzieren jedoch die Chance, Beans zwischen EJB-Servern verschiedener Hersteller austauschen zu können.

Eine positive Eigenschaft von Enterprise Java Beans ist die Persistenz von Entity Beans. Mit der *Container Managed Persistence (CMP)* sorgt der Container für die Speicherung der Bean-Daten. Die Informationen, welche Daten des Beans berücksichtigt werden müssen, werden dabei im Deployment-Descriptor festgelegt.

Mit der J2EE Referenz-Implementierung (*Java 2 Enterprise Edition SDK*) existiert ein Software-Paket zur Ausführung von Enterprise Java Beans. Sie ist Bestandteil der J2EE

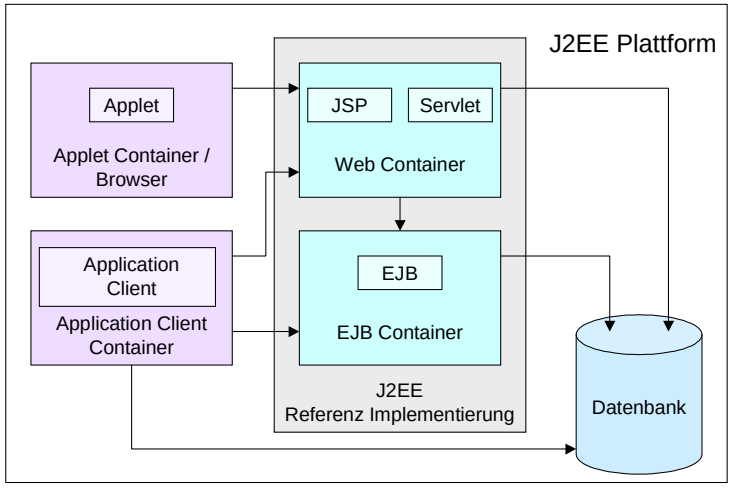

Abbildung 2.9: Die J2EE Architektur-Diagramm

Plattform[19], einer Spezifikation, die mehrere Konzepte und Technologien miteinander vereint. Abbildung 2.9 (nach Sun Microsystems, 2002b) gibt einen Überblick über die Bestandteile der Plattform. Sie setzt sich aus vier Container-Typen zusammen: Auf der Client-Seite mit den *Applet*- und *Application Client-Containern*, die das Front-End der Business-Anwendung repräsentieren, und auf der Server-Seite mit den *Web*- und *EJB-Containern* hinter denen sich die Business-Logik der Anwendung verbirgt. Alle Container stellen jeweils die Java-Laufzeit-Umgebung[20] für ihre Komponenten (Applikationen, Applets, Servlets, JSPs, EJBs) zur Verfügung. Die J2EE Referenz-Implementierung umfasst in der Spezifikation die Web- und EJB-Container.

Dem EJB-Container muss (mindestens) eine Datenbank zur Seite stehen, um die Persistenz der Entity-Beans zu ermöglichen oder den EJBs mittels der JDBC Schnittstelle Datenbankzugriffe zu ermöglichen. Ein Web-Container (ein Web-Server) arbeitet Java Server Pages (JSP) und Servlets ab und liefert statische HTML-Seiten. Die Möglichkeit des Zugriffs von JSPs und Servlets auf die EJBs sowie über JDBC auf Datenbanken ist gegeben.

Auf der Client-Seite entsprechen Application-Clients beliebigen Java-Programmen. Ihnen steht die Möglichkeit des Zugriffs auf den Web-Container, den EJB Container oder direkt auf die Datenbanken zur Verfügung. Im Gegensatz dazu steht den Applets gemäß (der Spezifikation) nur der Zugriff auf den Web-Container zu.

[19] Sun Microsystems (2002b); Denninger und Peters (2000); Monson-Haefel (2000)
[20] in der Regel durch eigene JVMs

2.4 Common Object Request Broker Architecture (CORBA)

Die *Common Object Request Broker Architecture (CORBA)* ist eine Spezifikation, die von der *Objekt Management Group (OMG)*[21] definiert wurde, um den Zugriff auf verteilte Objekte zu definieren[22].

Bei der OMG handelt es sich um eine Non-Profit-Organisation, die 1989 von Unternehmen unterschiedlicher Zielrichtung (Hardware-Hersteller, Software-Hersteller, Netzkomponenten-Hersteller, Fluggesellschaften etc.) gegründet wurde. Inzwischen umfasst die OMG mehrere hundert Firmen und Forschungseinrichtungen. Ziel der OMG ist die Schaffung von Standards, auf deren Grundlage Implementationen und Produkte entwickelt werden können. Die Ziele des CORBA-Standards lassen sich folgendermaßen zusammenfassen:

- Der Zugriff auf entfernte Objekte soll über eine gemeinsame Schnittstelle möglich sein.
- Der Zugriff auf ein entferntes Objekte muss über verschiedene Programmiersprachen hinweg möglich sein.
- Der Zugriff auf ein entferntes Objekt muss über verschiedene Betriebssysteme hinweg möglich sein.
- Der Zugriff auf ein entferntes Objekt soll für den Anwendungsentwickler transparent sein.

Im Vordergrund der Spezifikation stand die Schaffung von Interoperabilität zwischen heterogenen Systemen, an zweiter Stelle erst der entfernte Objekt-Zugriff.

Ein wichtiger Punkt der CORBA-Spezifikation stellt der *Object Request Broker (ORB)* dar (siehe Abb. 2.10 nach OMG, 2002 und Roth, 2000a). Er bietet die Möglichkeit der Weiterleitung von Anfragen eines Client zu dem entsprechenden Objekt. Die Interoperabilität zwischen ORBs wird dabei mit dem *General Inter-ORB Protocol (GIOP)* definiert. Die Ausprägung dieses Protokolls für TCP/IP ist das *Internet Inter-ORB-Protocol (IIOP)*.

Zwei Möglichkeiten stehen dem Client für Methodenaufrufe entfernter Objekte zur Verfügung: statische und dynamische Aufrufe.

Statische Aufrufe: Unter Verwendung von Schnittstellen-Spezifikationen, die in der programmiersprachenunabhängigen Schnittstellen-Beschreibungssprache *IDL (Interface Definition Language)* entwickelt werden, erzeugt ein *IDL-Compiler* den *IDL-Stub* und das *IDL-Skeleton*. IDL-Compiler existieren für verschiedene Ziel-Programmiersprachen wie Java, C++ und SmallTalk. Erst durch die Existenz der IDL kann die Programmiersprache der Objekt-Implementierung transparent gehalten werden.

[21] `http:\\www.omg.org`
[22] OMG (2002); Hofmann u. a. (2001); Roth (2000a); Orfali und Harkey (1997); Adam u. a. (1997)

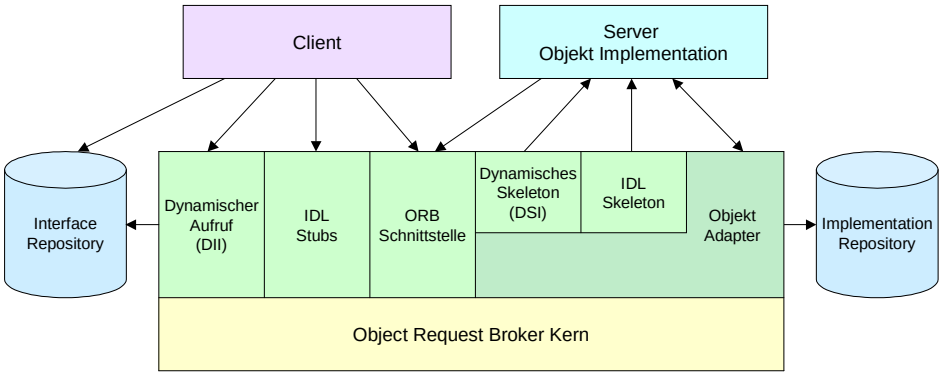

Abbildung 2.10: CORBA Struktur

Dynamische Aufrufe: Diese werden auf der Client-Seite durch das *Dynamic Invocation Interface (DII)* und auf der Server-Seite durch das *Dynamic Skeleton Interface (DSI)* abgearbeitet. Über das DII kann der Client Dienste des Servers zur Laufzeit erfragen und nutzen, ohne dass dafür die Interface-Definition zur Übersetzungszeit bekannt war. Das DSI nimmt diese Anfragen entgegen und ermittelt das entsprechende Objekt, um den Methodenaufruf durchführen zu können.

Der Objekt-Adapter stellt die Laufzeitumgebung der Server-Objekte zur Verfügung. Er ist zuständig für die Aktivierung von Objekten, die Entgegennahme von Anfragen, die Zuordnung von Objekt-Referenzen zu Objekt-Instanzen und für die Registrierung der Objekte und Klassen im *Implementation Repository*, das Informationen über die vom Server unterstützten Klassen und Objekte liefert. Das *Interface Repository* dient zur Speicherung vorhandener IDL-Schnittstellen (inkl. Meta-Daten).

Ein weiterer Teil der CORBA-Spezifikationen ist die *Objekt Management Architektur (OMA)*[23], die Referenz-Architektur, auf die sich alle anderen Spezifikationen beziehen. Die OMA führt die Begriffe *Komponente* und *Dienst* ein. Bei einer Komponente handelt es sich um die Zusammenfassung von Objekten, die eine gemeinsame Aufgabe erfüllen, während die Dienste die wahrgenommen Aufgaben darstellen. Abhängig von den Diensten werden die Komponenten den vier folgenden Gruppen in Abbildung 2.11 (nach Hofmann u. a., 2001) zugeordnet:

Application Objects: Hiermit sind die Objekte gemeint, die der Anwendungsentwickler im Rahmen seiner Problemlösung entwickelt und die als *die Anwendung* verstanden werden.

[23] OMG (2002); Hofmann u. a. (2001); Roth (2000a); Orfali und Harkey (1997); Adam u. a. (1997)

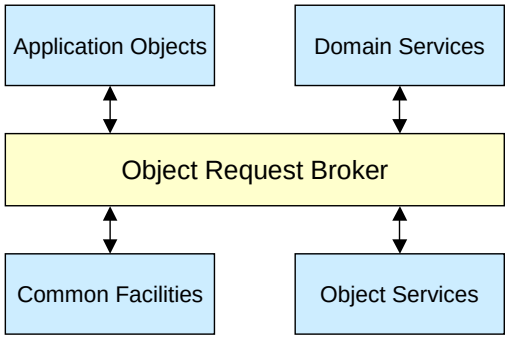

Abbildung 2.11: Object Management Architektur (OMA)

Domain Services: Diese Dienste bieten Lösungen für wiederkehrende Probleme, z.B. aus den Bereichen Gesundheitswesen, Transport, Telekommunikation oder Finanz-Dienstleistung.

Common Facilities: Dienste aus diesem Bereich sind nicht grundlegender Natur, jedoch können sie in vielen Anwendungen nützlich sein. Beispiele hierfür sind Drucken, Internationalisierung oder Datenaustausch.

Object Services: Anwendungen werden über die *Object Services* Basisdienste zur Verfügung gestellt, auf die CORBA-basierte Anwendungen zurückgreifen können. Jede OMA-konforme Implementation muss diese Dienste implementieren. Eine Auswahl dieser Dienste sind u.a. die Folgenden:

Naming Service: Dieser Dienst liefert anhand eines Namens eine Objekt-Referenz zu einem Objekt, das zuvor unter diesem Namen angemeldet wurde.

Life Cycle Service: Dieser Dienst stellt Operationen zum Erzeugen, Löschen, Kopieren und Bewegen eines Objekts zur Verfügung.

Persistence Service: Zur persistenten Speicherung von Objekten in Datenbanken oder Dateien stellt dieser Dienst eine Schnittstelle bereit.

Event Service: Objekte können sich für bestimmte Ereignisse registrieren lassen. Bei Eintreffen des Ereignisses wird das Objekt darüber informiert.

Time Service: Hiermit lassen sich verteilte Anwendungen zeitlich synchronisieren.

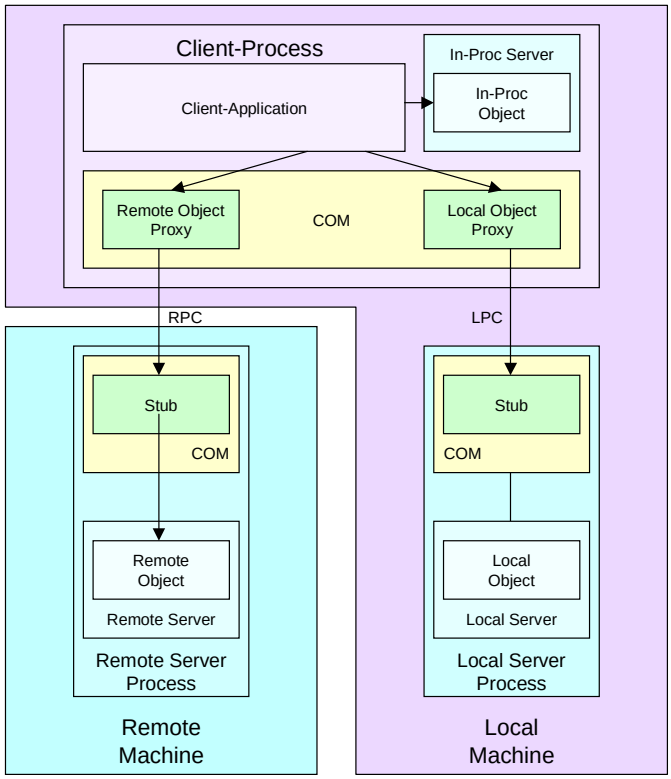

Abbildung 2.12: DCOM-Kommunikation zwischen Client und Server

2.5 Komponenten-Technologien in Windows

Dieses Kapitel beschäftigt sich mit Komponenten-Technologien für verteilte Anwendungen unter dem Windows-Betriebssystem.

2.5.1 Distributed Component Object Modell (DCOM)

Das *Distributed Component Object Model (DCOM)*[24] ist die Erweiterung des *Component Object Models (COM)* um die Fähigkeit Komponenten verteilt in einem Netz bereitzuhalten. Der Zugriff auf die entfernte Komponente über das Netz geht dabei auf die RPC-Spezifikation der *Open Software Foundation (OSF)* zurück, die Teil des *DCE (Distributed Computing Environment)* ist (siehe Abs. 2.2).

[24] Eddon und Eddon (1998); Sessions (1997)

COM-Objekte sind sprachunabhängig und folgen einem objektbasierten Binär-Standard. Dies bedeutet auf der einen Seite, dass alle Programmiersprachen zur Entwicklung von COM-Objekten in Frage kommen, die diesen Binär-Standard unterstützen. Auf der anderen Seite ergibt sich daraus jedoch eine große Abhängigkeit vom Windows-Betriebssystem.

Der Zugriff auf ein COM-Objekt erfolgt über Interfaces (Schnittstellen), die eine Abstraktionsebene zwischen Anwendung und Objekt schaffen. Jedes Objekt kann mehrere Interfaces implementieren, die durch eindeutige *IIDs (Interface IDs)*[25] bestimmt sind. Per Konvention darf ein Interface nicht verändert werden, wenn sich anschließend nicht auch seine GUID ändert. Neben den eindeutigen Interface IDs existiert für jede DCOM-Klasse eine eindeutige Klassen-ID, der *CLSID (Class Identifier).*

Die Schnittstellen-Definitionen werden mittels der sprachunabhängigen Beschreibungssprache *IDL (Interface Description Language)* definiert. Es handelt sich dabei um eine Erweiterung der IDL-Spezifikation des DCE-Standards.

Der Zugriff auf ein DCOM-Objekt unterscheidet sich in Abhängigkeit des Ortes eines Objektes (siehe Abb. 2.12 nach Eddon und Eddon, 1998). Beim *In-Proc-Server* befindet sich das Objekt im selben Prozess wie der aufrufende Client. Beim *Local-Server* findet der *Local-Procedure-Call (LPC)* Anwendung. Hier befindet sich das Objekt auf derselben Maschine, jedoch in einem anderen Prozess. Beim Remote-Server findet der *Remote-Procedure-Call (RPC)* Anwendung, wobei der Server-Prozess auf einem (physikalisch) anderem Rechnersystem läuft.

Wie auch bei RMI und CORBA, existiert in DCOM ein Verfahren zum dynamischen Aufruf von Methoden, die erst zur Laufzeit bestimmt werden. DCOM nennt dieses Verfahren *Automation* und benötigt einen *Automation Server,* auf den der Client als *Automation Controller* zugreift. Hierfür ist weiterer Implementationsaufwand erforderlich, der bei RMI und CORBA entfällt, da dort automatisierte Verfahren existieren.

2.5.2 .NET

.NET[26] ist Microsofts Technologie für die zukünftige Entwicklung von Anwendungen für die Windows-Plattformen[27]. Vom Anspruch her geht .NET in eine ähnliche Richtung wie Java und J2EE, im Detail unterscheiden sich beide Ansätze jedoch.

Kern des *.NET-Framework* ist die *Common Language Runtime (CLR),* das die Laufzeitumgebung für .NET-Anwendungen darstellt (siehe Abb. 2.13). Diese liegen nicht in nativem Code vor, sondern in der plattformunabhängigen Zwischensprache *Common Intermediate Language (CIL)*[28]. Im Gegensatz zu Java, bei dem der Java-Bytecode entweder interpretiert

[25] IIDs sind spezielle *Globally Unique Identifers (GUIDs).*

[26] Beer u. a. (2003); Richter (2001); Hoffman u. a. (2001); Stal (2001); Microsoft Corporation (2002)

[27] Microsoft verwendet die Bezeichnung „.NET“ im Allgemeinen in einem größeren Kontext, hier wird jedoch das .NET-Framework beschrieben.

[28] Auch als *MSIL (Microsoft Intermediate Language)* oder einfach nur *IL (Intermediate Language)* bezeichnet.

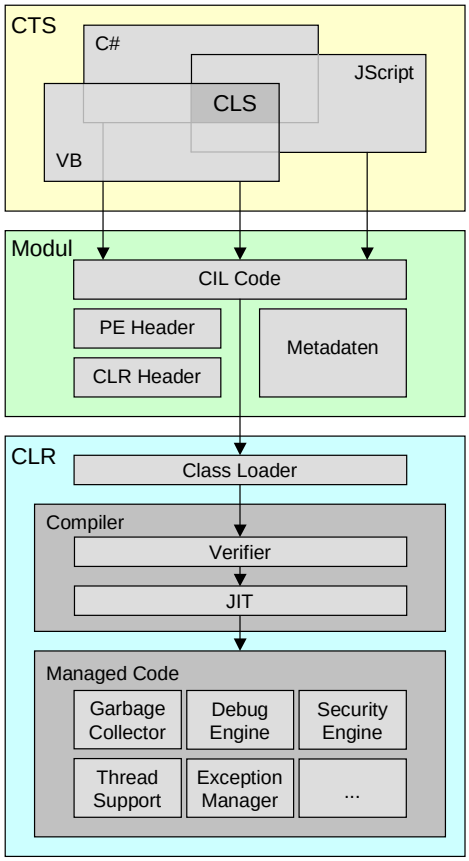

Abbildung 2.13: Technologien des .NET-Frameworks

oder im Falle des Java Just-In-Time-Compilers (JIT) bei Bedarf compiliert wird, findet eine Ausführung der CIL ausschließlich im übersetzten Zustand statt. Der so übersetzte und von der CLR ausgeführte Code wird als *Managed Code* bezeichnet.

Die CLR überprüft vor der Ausführung die Zulässigkeit der Anwendung und stellt zur Laufzeit die Typ-Sicherheit beim Zugriff auf Objekte sicher. Das Speicher- und Ressourcenmanagement wird durch eine Garbage-Collector-Einheit übernommen. Eine Sicherheitseinheit sorgt für den kontrollierten Zugriff auf Ressourcen.

Im Gegensatz zu Java, bei dem der Java-Bytecode ausschließlich durch das Übersetzen von Java-Programmen erzeugt wird, kann der Ursprung der Common Intermediate Language verschiedene Programmiersprachen zur Grundlage haben, z.B. C# oder Visual Basic (VB). Damit dies aber möglich ist, sind einige Anforderungen zu erfüllen: Zunächst müssen alle Programmiersprachen die *Common Language Specification (CLS)* umsetzen. Dabei

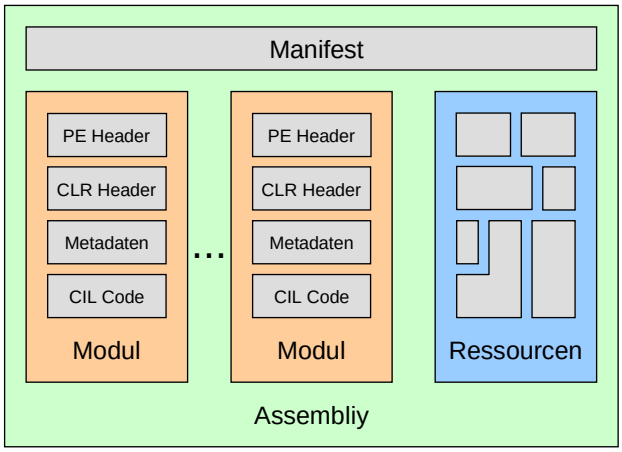

Abbildung 2.14: Assembly

handelt es sich um einen Satz von Vorschriften, die garantieren, dass ein Compiler einer Programmiersprache CIL-Code erzeugt, der innerhalb einer CLR lauffähig ist. Damit auch ein sprachenübergreifender Zugriff auf Datentypen möglich ist, muss das *Common Type System (CTS)* in jeder Programmiersprache zur Verfügung stehen.

Die Regeln der CLS stellen eine Teilmenge des CTS dar, was bedeutet, dass alle Regeln der CTS auch für die CLS gelten. Durch die Umsetzung von CLS und CTS ist nicht nur sichergestellt, dass zwei entsprechende Datentypen zweier Programmiersprachen interoperabel sind. Es ist nun auch möglich, die Grenzen zwischen zwei Programmiersprachen zu verwischen: Eine Klasse, die in einer bestimmten Programmiersprache entwickelt wurde, kann durch eine Klasse einer anderen Programmiersprache abgeleitet werden.

Der Compiler für eine CLS-konforme Programmiersprache (z.B. für C# oder VBasic-.NET) erzeugt neben dem CIL-Code weitere Informationen, die in einem Modul zusammengefasst sind (siehe Abb. 2.13). Es wird der *PE*[29]-Header erzeugt, der neben dem Datei-Typ auch Zeitstempel-Informationen beinhaltet. Zusätzlich wird der CLR-Header erzeugt, der unter anderem Angaben über die CLR-Version, weitere Flags und auch die Einstiegsmethode des Moduls definiert. Im Metadaten-Bereich werden Tabellen abgelegt, die die vollständige Schnittstellenbeschreibung der Klassen, Felder und Methoden beinhalten. Diese Informationen können mittels einer Reflection-Schnittstelle erfragt werden.

Das .NET-Framework ist nicht alleine zur Entwicklung neuer Anwendungen für die Windows-Plattform geeignet, sondern auch zum Aufbau verteilter Systeme. Der Zugang ist über den Web-Server (z.B der IIS) und die .NET-Bibliothek *ASP.NET* zur .NET-Komponente

[29] PE = Portable Executable

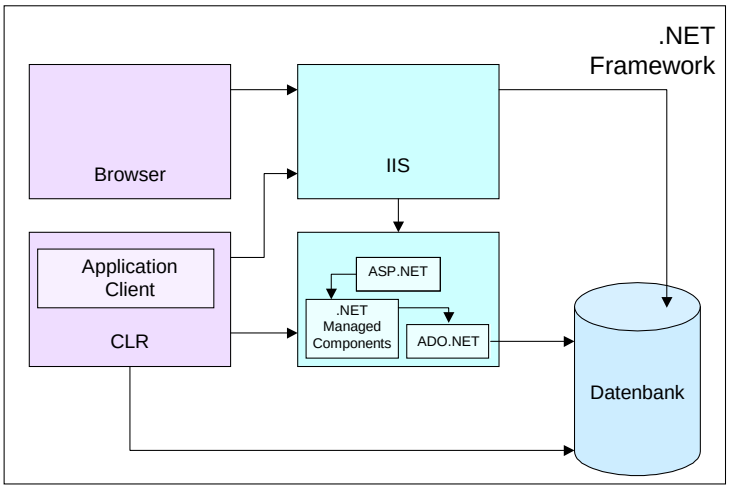

Abbildung 2.15: .NET Framework

möglich. Der zweite wesentliche Weg ist der Zugriff auf die .NET-Komponente als entfernter Methodenaufruf. Hier findet das SOAP-Protokoll (auf das in Abschnitt 2.7 näher eingegangen wird) seine Anwendung, bei dem die Komponente als Web-Service bereitgestellt wird.

Die beschriebenen Wege sind in Abbildung 2.15 aufgeführt, wobei eine Ähnlichkeit zu der J2EE-Technologie deutlich wird.

2.6 XML-RPC

XML-RPC[30] ist ein einfacher Ansatz zur Ausführung entfernter Funktionen. Im Gegensatz zu RMI, CORBA oder DCOM definiert XML-RPC jedoch nur das Protokoll, das zwischen Client und Server ausgetauscht wird. Ein Versuch, einen Methodenaufruf unter Einsatz von Stub und Skeleton transparent zu gestalten, existiert nicht. Ebenso beschränkt sich XML-RPC auf die Ausführung entfernter Prozeduren. Ein Zugriff auf ein entferntes Objekt findet nicht statt.

Eine XML-RPC-Anfrage ist ein HTTP-POST-Request mit einem XML-Dokument als Anhang (siehe Code A.4 im Anhang A auf Seite 141). XML (eXtensible Markup Language)[31] ist eine Meta-Sprache. Mit ihr lassen sich Sprachen definieren, die Dokumente zum Datenaustausch zwischen verschiedenen Systemen spezifizieren. Zur Definition der

[30] Winer (1999); Laurent (2001); McLaughlin (2001)
[31] Bray u. a. (2000, 1999)

XML-Tag	Bedeutung	Beispiel
`<i4>` oder `<int>`	32-Bit Ganzzahl mit Vorzeichen	-1234
`<boolean>`	Wahr oder falsch	0
`<string>`	ASCII String	dies ist ein String
`<double>`	Doppeltgenaue Floating-Point-Zahl mit Vorzeichen	1.234
`<dateTime.iso8601>`	Datum/Uhrzeit	20020929T21:58:34
`<base64>`	base64-kodierte Binärdaten	VXdlIGxpZWJ0IElyaXM=

Tabelle 2.1: Skalare Datentypen bei XML-RPC

XML-Sprache kann entweder eine *DTD (Document Type Definition)*[32] oder ein *XML Schema*[33] herangezogen werden. XML-Schemas haben gegenüber DTDs den Vorteil, dass sie selbst XML-Dokumente sind und deren Verarbeitung somit erleichtert wird. Zusätzlich ist die Entwicklung vollständiger Datenmodelle mittels XML-Schemas einfacher, da primitive Datentypen schon vordefiniert sind.

Normalerweise wird in einem XML-Dokument angegeben, wo sich dessen Sprachdefinition befindet, um die Gültigkeit des XML-Dokumentes feststellen zu können. Die Entwickler von XML-RPC haben auf eine solche Definition verzichtet, es existieren jedoch DTDs im Internet, die der Spezifikation entsprechen.

Bei einem HTTP-Request wird eine URL übergeben, die normalerweise die angeforderte Ressource spezifiziert. XML-RPC überlässt es der Implementation des Servers, in welcher Weise die angeforderte URL berücksichtigt wird.

In dem übermittelten XML-Dokument werden die aufgerufene Methode sowie die Aufruf-Parameter kodiert. Im Gegenzug wird ein XML-Dokument geliefert, das entweder die Rückgabewerte (siehe Code A.5) oder eine Fehlerbeschreibung beinhaltet (siehe Code A.6).

Als Parameter sind drei Klassen von Datentypen möglich: *Skalare* (siehe Code A.1), *Strukturen* (siehe Code A.2) und *Listen* (siehe Code A.3). Die möglichen skalaren (einfachen) Datentypen sind in der Tabelle 2.1 definiert; der Code zeigt, wie die Datentypen in XML kodiert werden.

Strukturen beinhalten eine Liste von Paaren mit Schlüsselworten und den damit assoziierten Werten, die wieder entweder vom Typ Skalar, Struktur oder Liste sein können. Die Reihenfolge der Schlüsselworte-Wert-Paare spielt im kodierten XML-Dokument keine Rolle.

Führt eine Anfrage zu einem Fehler, so ist der Inhalt der Antwort eine Struktur mit dem Fehlercode `faultCode` und der Fehler-Beschreibung `faultString` (siehe Code A.6). Zusätzliche Angaben sind möglich, werden aber von der Spezifikation nicht empfohlen.

[32] Bray u. a. (2000)
[33] Fallside (2001); Thompson und Mendelsohn (2001); Biron und Malhotra (2001)

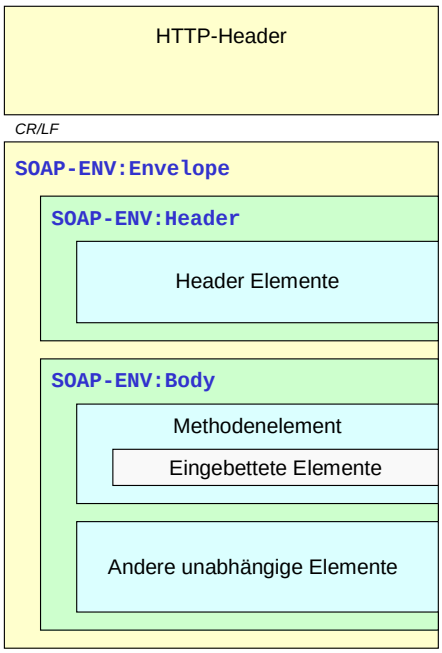

Abbildung 2.16: Die SOAP Anfragestruktur

Die Spezifikation von XML-RPC umfasst ca. 5 DIN-A4-Seiten. Dies macht eine Implementation des Protokolls einfach. Aus diesem Grunde sind zahlreiche Implementationen auf Grundlage der unterschiedlichsten Programmiersprachen verfügbar. Als Beispiel hierzu sei die Java-Implementation des Apache Web-Service Projektes genannt[34].

2.7 Simple Object Access Protocol (SOAP)

SOAP[35] verfolgt einen ähnlichen Ansatz wie XML-RPC: Unter Verwendung von Standard-Internet-Technologien sollen entfernte Methodenaufrufe (RPC's) ermöglicht werden[36].

Das Format zum Austausch der Nachrichten wurde dabei als XML definiert. Als Nachrichten-Transportprotokoll ist jedes Internet-Protokoll denkbar, das in der Lage ist, XML-Dokumente zu transportierten (FTP, SMTP, etc.). Allerdings kommt der Request-Response-Ansatz von HTTP dem Ansatz von SOAP besonders entgegen. Die folgenden Erläuterungen

[34] Apache Software Foundation (2003)
[35] Bis zur Spezifikation 1.1 als Akronym für *Simple Object Access Protocol* definiert, heute für sich selbst stehend.
[36] Mitra (2002); Gudgin u. a. (2002a,b); Scribner und Stiver (2001)

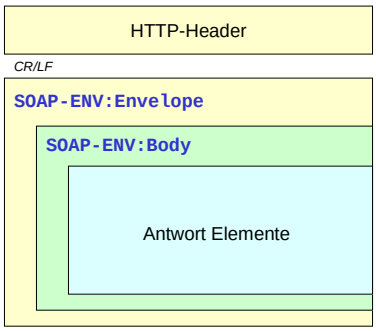

Abbildung 2.17: Die SOAP-Antwort

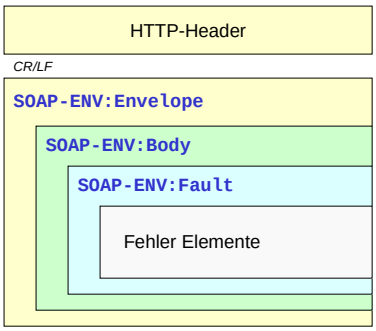

Abbildung 2.18: Die SOAP-Fehlermitteilung

beziehen sich aus diesem Grunde auf HTTP als Transportprotokoll.

Eine SOAP-Anfrage über HTTP ist schematisch in Abb. 2.16 dargestellt. Die SOAP-Nutzlast ist in einem SOAP-Envelope (`SOAP-ENV:Envelope`) kodiert, der seinerseits einen (optionalen) SOAP-Header (`SOAP-ENV:Header`) sowie einen SOAP-Body (`SOAP-ENV:Body`) umschließt. Der SOAP-Body enthält die eigentliche Methoden-Anfrage, der SOAP-Header gibt weitergehende Informationen zu der Anfrage (z.B. den Weg, den die Nachricht gehen soll).

Eine SOAP-Antwort ist ähnlich einer SOAP-Anfrage aufgebaut (siehe Abb. 2.17). Der SOAP-Body wird dazu verwendet, die Methoden-Antwort zu beschreiben. Im Falle eines Fehlers kann entweder ein HTTP-Error-Code verwendet werden, um auf diesen Umstand hinzuweisen oder der Fehler wird als SOAP-Fehler (`SOAP-ENV:Fault`) im Body der Anfrage übertragen (siehe Abb. 2.18).

Die eigentlichen Nutzdaten werden im SOAP-Body übertragen. Hierzu müssen die Daten unter Verwendung von Datentypen kodiert werden. Dabei ist SOAP wesentlich flexibler als XML-RPC, das ja nur eine kleine Anzahl von Datentypen bereitstellt.

In SOAP sind zunächst alle primitiven Datentypen von XML-Schema erlaubt, aber auch komplexe zusammengesetzte Datentypen (Strukturen, Listen) für die ein XML-Schema definiert werden kann. Interessant ist die Möglichkeit, Referenzen auf andere Werte anzugegeben, also Pointer innerhalb des SOAP-Body.

Üblicherweise werden durch Firewalls alle Ports blockiert, die im Zusammenhang mit RPC stehen. Damit ist die Kommunikation zwischen Client und Server und die Ausführung von RPCs nicht immer möglich. Jedoch wird der HTTP-Port im WWW-Umfeld immer freigegeben. Da SOAP die RPC-Anfragen über HTTP stellt, kann die Firewall an dieser Stelle überwunden werden.

Auf der anderen Seite umgeht man damit sinnvolle Sicherheitsbarrieren, die man ursprünglich durch Firewalls errichten wollte. Es müssen also zusätzliche Maßnahmen ergriffen werden, die einen Missbrauch durch Dritte verhindern.

Ein anderer Nachteil von SOAP ist seine Schwergewichtigkeit. Alle Daten müssen als Text kodiert und in XML strukturiert werden. Damit ist zwar eine äußerst flexible Repräsentation von Informationen möglich, das Verhältnis der Informationen zu den übertragenen Daten ist jedoch sehr ungünstig. In einem Umfeld, in dem man die Kontrolle über die Client- und Server-Implementation hat oder wo Geschwindigkeit eine Rolle spielt, sollte man auf jeden Fall native Kommunikationsprotolle, wie bei RMI, CORBA oder DCOM einsetzen.

Trotzdem entwickelt sich SOAP immer mehr zum Standard im Umfeld der entfernten Objektzugriffe, insbesondere bei *Web Services* (siehe Abs. 2.8), da es vorhandene Internet-Standards verwendet.

2.8 Web Services

Web Services[37] sind eine relativ neue Technologie. Viele Aspekte dieser Technologie sind noch im Entstehungs- oder Entscheidungsprozess, doch es zeichnet sich langsam eine gemeinsame Sicht ab. Das W3C versteht Folgendes unter Web Service[38]:

> *Ein Web Service ist eine Software, die durch eine URL identifiziert ist (RFC 2396) und deren öffentliche Schnittstellen und Bindungen durch XML definiert werden. Seine Definitionen können durch andere Software-Systeme ermittelt werden. Diese Systeme können mit dem Web Service in der vorgeschriebenen Art interagieren, wobei über Internet-Protokolle versandte XML-basierte Nachrichten verwendet werden.*

Für die Art der Kommunikation zwischen Service-Anbieter und Service-Anfragendem hat sich SOAP als Lösung etabliert. (siehe Abb. 2.19 auf der nächsten Seite nach Gottschalk u. a., 2002). Zur Erfüllung der Definition werden jedoch zusätzliche Technologien benötigt: Die *Web Service Description Language (WSDL)*[39] und die *UDDI (Universal Description, Discovery and Integration)*[40].

In einem ersten Schritt veröffentlicht der Web-Service-Anbieter die Beschreibung seiner Services mittels des auf XML aufbauenden WSDL-Formates bei einem Service-Register. Dieses Register bietet hierzu selbst einen über SOAP ansprechbaren Dienst an, der durch UDDI definiert ist.

Durch den gleichen Dienst ist der Service-Anfragende in der Lage, einen Service bei dem Service-Register zu erfragen. Das übertragene WSDL-Dokument ermöglicht es nun, den Dienst zu nutzen.

Im Folgenden sollen nun WSDL und UDDI etwas genauer beschrieben werden.

[37] Champion u. a. (2002); Haas und Orchard (2002); Deitel u. a. (2003); Gottschalk u. a. (2002)
[38] Übersetzung aus Brown und Haas, 2002
[39] Cinnini u. a. (2003); Moreau und Schlimmer (2003); Sadiq und Kumar (2002)
[40] UDDI-Tech (2000); Bellwood u. a. (2002)

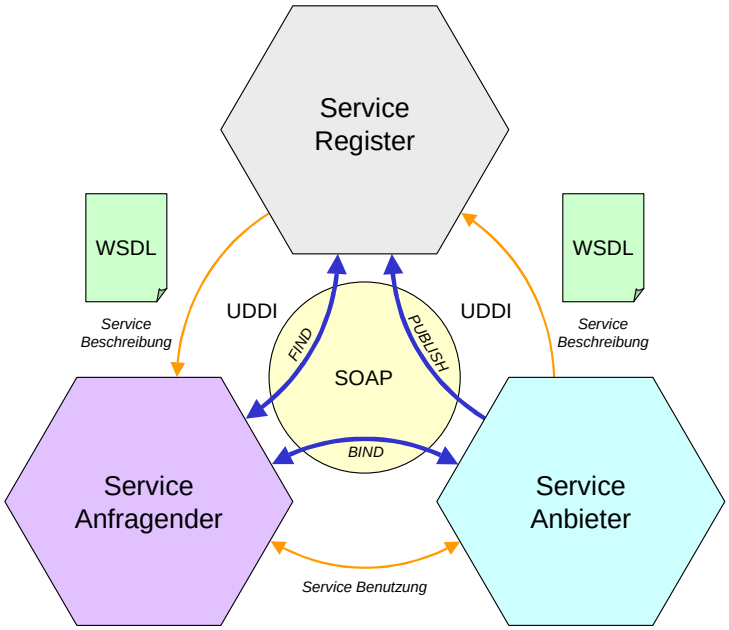

Abbildung 2.19: Web Service: Akteure und Protokolle

2.8.1 Web Service Description Language (WSDL)

Ein WSDL-Dokument ist ein XML-Dokument und unterteilt sich in mehrere Bereiche, die folgende Informationen beinhalten:

Der `message`-Bereich definiert die übertragenen Daten. Eine `message`-Nachricht wird durch einen Namen identifiziert und fasst mehrere Paare aus Bezeichner und Datentyp (`part`) zusammen.

Im `interface`-Bereich wird das Ein- und Ausgabeverhalten einer Operation beschrieben. Eine Operation ist eine Menge von Ein- und Ausgabenachrichten, ein Interface ist eine Menge von Operationen.

Die Bindung eines Interfaces an das verwendete Protokoll und die Festlegung des Nachrichtenformats findet in der `binding`-Sektion statt.

Die Anbindung eines bestimmten Service erfolgt an einen `end-point` (End-Punkt), womit die URL des Service sowie die Bindung zu einem Übertragungsprotokoll festgelegt wird.

Schließlich legt die `service`-Sektion genau das Interface fest, das einen Service anbietet sowie den End-Punkt über den er verfügbar ist.

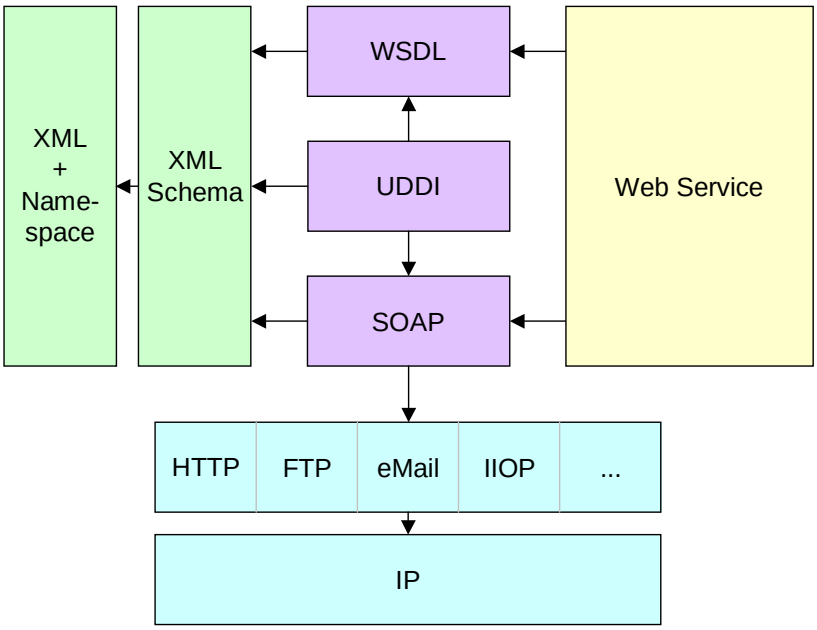

Abbildung 2.20: Abhängigkeiten der Technolgien im Web-Service-Umfeld

2.8.2 Universal Description, Discovery and Integration (UDDI)

Durch die Definition eines WSDL-Dokuments ist eine genaue Spezifikation einer Web-Service-Schnittstelle möglich. UDDI hat sich zur Aufgabe gemacht, einen Mechanismus zur Verfügung zu stellen, der das Auffinden und Nutzen von Web-Services ermöglicht, die von verschiedenen Unternehmen oder Organisationen angeboten werden.

Kern dieser Idee ist ein Verzeichnis, in dem nach den Services gesucht werden kann. Es unterteilt sich in drei Bereiche:

White Pages: Ähnlich einem Telefonbuch werden hier Kontakt-Informationen (Adresse, Telefon, eMail) zu den Anbietern von Web Services abgelegt.

Yellow Pages: Dies entspricht den Gelben Seiten und schlüsselt die Anbieter entsprechend den Branchen auf. Diese Einträge verweisen auf die White Pages.

Green Pages: Technische Details zu den angebotenen Diensten u.a. Verweise auf die WSDL-Dokumente befinden sich in dieser Sektion.

Jeder Dienste-Anbieter im Internet soll seine Web Services mittels des UDDI-Registers veröffentlichen. Jedoch sollte der Eindruck von Abbildung 2.19 vermieden werden, dass

alle Schritte der Service-Bindung automatisch vonstatten gehen können. Das Problem hierbei lässt sich folgendermaßen beschreiben: Zwar kann ein Web-Service automatisiert bei einem UDDI-Register gesucht und gefunden werden und ebenso ist es möglich, diesen Service nach Analyse des WSDL-Dokumentes aufzurufen, allerdings wird in keiner Weise die Bedeutung dieses Aufrufs dem Service-Aufrufenden deutlich werden können. Diese wird sich dem Entwickler einer Software erst durch weitere Konsultationen von Dokumentationen erschließen.

Abbildung 2.20 auf der vorherigen Seite gibt im Vergleich zu Abb. 2.19 auf Seite 34 eine andere Sicht auf die Abhängigkeiten der im Web-Service-Umfeld verwendeten Technologien. Die Pfeile in der Grafik visualisieren hierbei die Abhängigkeit der Technologien voneinander. Die dargestellte Abhängigkeit der Web-Services von WSDL ist so zu verstehen, dass ohne Beschreibung der Service nicht genutzt werden kann. Der Web-Service selbst ist also nur indirekt auf WSDL angewiesen. Die Verbindung von UDDI zu XML-Schema ergibt sich aus der Tatsache, dass Unternehmensinformationen der White Pages und Yellow Pages in einem XML-Format übertragen werden.

Die Grafik verdeutlicht, dass Web-Services indirekt über SOAP, WSDL und UDDI auf zwei Schlüssel-Technologien aufbaut: auf der einen Seite auf Internet-Protokoll-Standards, auf der anderen Seite auf XML. Durch die allgemeine Akzeptanz dieser Standards ist mit einer weiten Verbreitung von Web-Services für die Zukunft zu rechnen.

3 Aktuelle Forschungsfelder

Ich muss nicht wissen, wie es funktioniert.
Ich muss nur wissen, dass es funktioniert.

FRANZ BECKENBAUER

Verteilte Anwendungen sind Forschungsschwerpunkt verschiedener universitärer und institutioneller Forschungsgruppen. Dabei rücken immer mehr die inneren Prozesse, Abläufe und Strukturen in den Vordergrund. Die im letzten Kapitel beschriebenen Middleware-Plattformen und Komponenten-Frameworks gelten dabei als Vertreter der „Black-Box“-Architekturen, bei denen der innere Aufbau weder bekannt noch veränderbar ist. Neue Forschungsvorhaben verfolgen einen „White-Box“-Ansatz, um die inneren Abläufe und Datenflüsse anpassungsfähig zu gestalten.

Zusätzlich existieren Forschungsprojekte, die nicht mittelbar im Kontext von Middleware oder verteilten Anwendungen stehen, deren Erkenntnisse und Konzepte jedoch Eingang in Middleware-Architekturen gefunden haben. Dies sind hauptsächlich Projekte im Umfeld von flexiblen Datenfluss-Architekturen.

Um einen Überblick über all diese Projekte zu liefern, wurde eine Klassifikation vorgenommen, die weder den Anspruch auf Überschneidungsfreiheit noch auf Vollständigkeit erhebt. Sie deckt jedoch die Aspekte ab, die im Rahmen dieser Arbeit von Bedeutung sind. Es sind dies die Kern-Bereiche

- der reflektiven Middlewaren,
- der Multimedia-Frameworks und
- der Protokoll- und Kommunikationsframeworks.

Abb. 3.1 gibt einen Überblick über diese Bereiche.

Weitere benachbarte Bereiche, die im Folgenden nur am Rande behandelt werden, sind die Streaming-Spezifikationen von CORBA und der Bereich der Betriebssysteme (OS). Letztendlich werden im Feld *Sonstiges* Plattformen und Prototypen subsumiert, die nicht eindeutig den oben beschriebenen Kern-Bereichen zugeordnet werden können, aber wichtige Teilaspekte ansprechen.

Innerhalb der Gruppierungen sind *Spezifikationen*, *Technologien*, *Projekte* und *Systeme* aufgeführt. Die durch Pfeile dargestellten Abhängigkeiten werden konkret in den entsprechenden Abschnitten beschrieben.

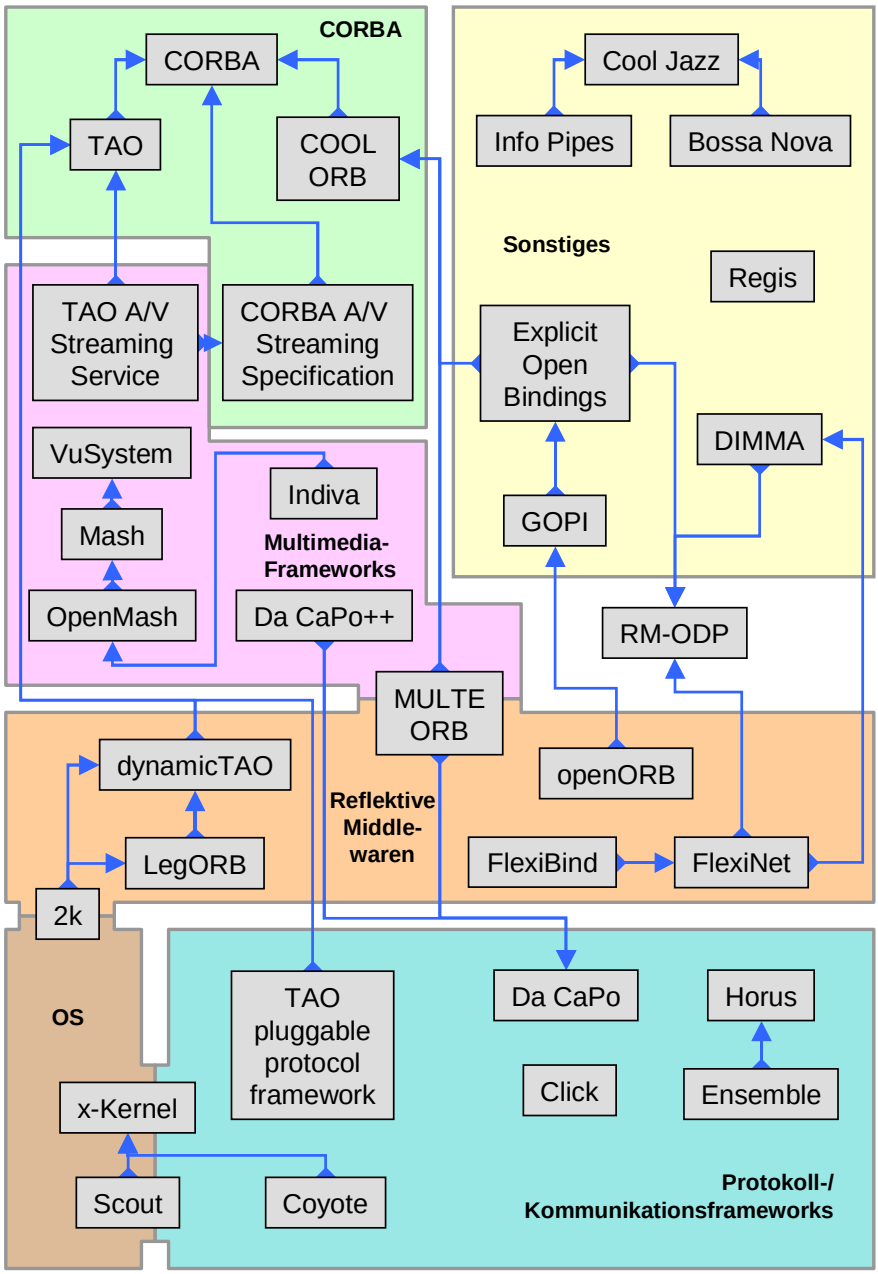

Abbildung 3.1: Überblick über verwandte Technologiebereiche

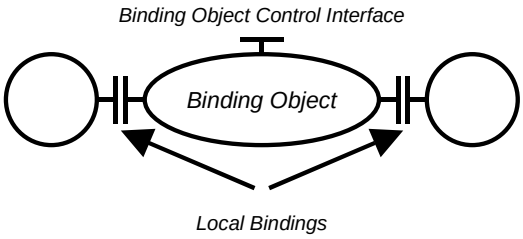

Abbildung 3.2: Explizite Bindungen

Dieses Kapitel führt kurz in die aufgeführten Technologiebereiche ein, beschreibt deren Eigenschaften und benennt einige ihrer wichtigsten Vertreter.

3.1 Reflektive Middleware

Der Bereich der reflektiven Middleware[1] beschäftigt sich mit Technologien, um Middleware-Architekturen flexibel und anpassungsfähig zu machen. Reflektive Systeme besitzen eine Repräsentation ihres Aufbaus (das entscheidend ist für das Verhalten), das mittels so genannter *Inspection* gelesen und mittels so genannter *Adaption* verändert werden kann. Es besteht ein direkter Zusammenhang zwischen der Repräsentation des Systems und dessen Aufbau. Eine Veränderung der Repräsentation führt direkt zu der Veränderung des Systemaufbaus; eine Veränderung des Systemaufbaus schlägt sich sofort in deren Repräsentation nieder. Man bezeichnet diese Art von Repräsentation als *causally connected self representation* (CCSR)[2].

Ziel dieses Ansatzes ist die Schaffung von Systemen, die sich einfacher an verschiedene Einsatzumgebungen anpassen können. Drei Forschungslinien dieser Gattung sollen an dieser Stelle näher erläutert werden: openORB/GOPI, dynamicTAO/LegORB/2K und FlexiNet/FlexiBind/DIMMA.

3.1.1 openORB und GOPI

Das *Department of Computing* der *Lancaster Universität* beschäftigt sich unter anderem mit Middleware-Architekturen. Erste Ansätze basierten auf dem ISO-Referenz-Modell *RM-*

[1] Die deutsche Übersetzung des Begriffs „reflective Middleware“ wurde wegen seiner Nähe zum englischen Original als „reflektive Middleware“ festgelegt, obwohl das Wort „reflektiv“ in der deutschen Sprache nicht existiert.

[2] Coulson (2002)

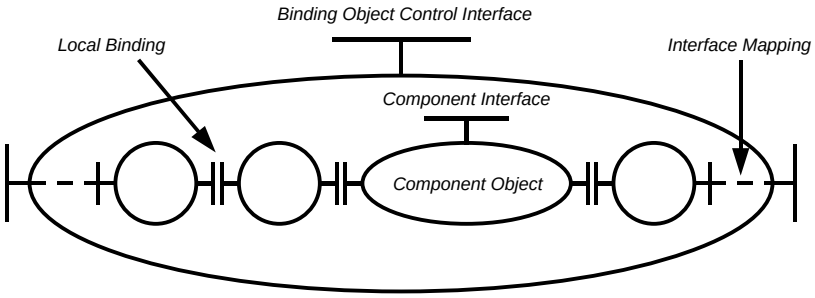

Abbildung 3.3: Offene Bindungen

ODP (Reference Model for Open Distributed Processing[3]). Dieses Modell definiert Schnittstellen (*interfaces*) als Zugriffspunkte zwischen Objekten und Komponenten, die eine Menge von Methoden definieren und die statischen Beziehungen zwischen Objekten auflösen[4]. Ein Objekt kann an ein anderes Objekt gebunden werden (*binding*), wenn eine passende Schnittstelle vorliegt. Der Aufruf von Methoden erfolgt dann nicht direkt über das Objekt, sondern über die Schnittstelle. *Lokale Bindungen* liegen vor, wenn die Schnittstellen im gleichen Adressraum vorliegen. Zusätzlich sind implizite und explizite Bindungen definiert. Bei den *impliziten Bindungen* wird die Komplexität der Bindung gegenüber dem Programmierer verborgen (z.B. beim Methodenaufruf in CORBA oder RMI). Bei *expliziten Bindungen* bleibt die Kontrolle über die Bindung bezüglich Aufbau und Konfiguration in der Hand des Programmierers.

Die CORBA-Spezifikation definiert den entfernten Objekt-Zugriff über implizite Bindungen. Ein Ziel des Projektes war die Einführung von expliziten Bindungen[5]. Diese Fähigkeit wurde durch den Einsatz von Bindungsobjekten erreicht (siehe Abb. 3.2 aus Fitzpatrick u. a., 1998). Der Aspekt der CCSR wurde durch Objekt-Grafen geschaffen, die über Meta-Interfaces angesprochen werden und den darunter liegenden End-zu-End-Kommunikationspfad repräsentieren.

Der Objekt-Graf besteht aus Prozess-Objekten und lokalen Bindungen oder Bindungen über Bindungsobjekte. Durch die Einführung von *Interface-Mapping* gelangt man zu den *offenen Bindungen* (siehe Abb. 3.3 aus Fitzpatrick u. a., 1998), womit externe Interfaces mittels Proxies an die internen Interfaces der Komponenten gebunden werden.

Erste Implementierungen eines Binding-Frameworks basierten auf dem *COOL-ORB*, eines CORBA-ORB's, später auf der Eigenentwicklung der *GOPI-Middleware*[6]. Auf Grund-

[3] Raymond (1998); ISO/IEC 10746-1 (1998); ISO/IEC 10746-2 (1996); ISO/IEC 10746-3 (1996); ISO/IEC 10746-4 (1998); ISO/IEC 13235-1 (1998)

[4] Andersen (2002)

[5] Fitzpatrick u. a. (1998); Blair u. a. (1998); Eliassen u. a. (1999)

[6] Coulson und Baichoo (1999); Coulson u. a. (2002a)

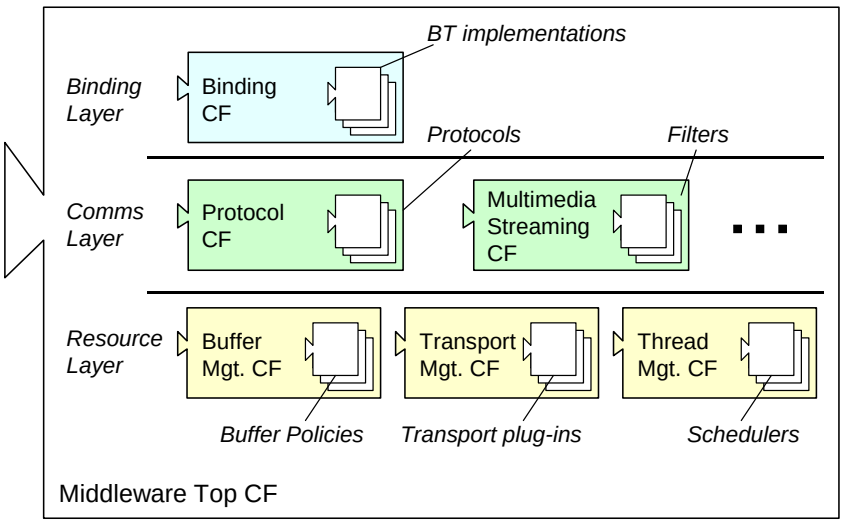

Abbildung 3.4: Die Architektur von OpenORB

lage dieser Erfahrungen wurde die *OpenORB*-Reflektive Middleware[7] entwickelt, in der Version 1 als OOPP (OpenORB Python Prototype) mit dem Ziel der Schaffung architektonischer Abstraktionen für eine reflektive, komponentenbasierte Middleware. Die Reimplementation in C++ der Version 2 hatte die Ziele der Effizienz, der Standard-Konformität (im Bezug auf CORBA) und der Integrität.

Die Architektur von OpenORB ist in Abbildung 3.4 (aus Coulson u. a., 2002b und Clarke u. a., 2001) als Komponenten-Framework (component framework = CF) dargestellt. Es besteht aus den drei Schichten *Binding Layer*, *Communication Layer* und *Resource Layer*. Jede Komponente einer Schicht darf nur auf Komponenten der gleichen oder einer tieferen Schicht zugreifen. Das *Top CF* ist für den Lebenszyklus der Komponenten sowie zur Auflösung der Abhängigkeiten zwischen den Komponenten verantwortlich. Die Architektur ist so aufgebaut, dass neue CFs dynamisch eingeführt werden können, solange dies nicht den Richtlinien (*policies*) des Top-CF widerspricht. Hierzu muss jedes CF der zweiten Ebene zur Design-Zeit seine statischen Abhängigkeiten definieren.

3.1.2 dynamicTAO, LegORB und 2K

Ein weiteren Ansatz zur Entwicklung einer reflektiven Middleware wurde an der Universität in Illinois umgesetzt. Das Ergebnis dieser Arbeit ist die *dynamicTAO*-Middleware[8]. Sie

[7] Andersen u. a. (2000); Parlavantzas u. a. (2000); Clarke u. a. (2001); Coulson u. a. (2002b); Kon u. a. (2002)
[8] Román u. a. (1999); Eliassen u. a. (1999); Kon u. a. (2000c,b, 2002)

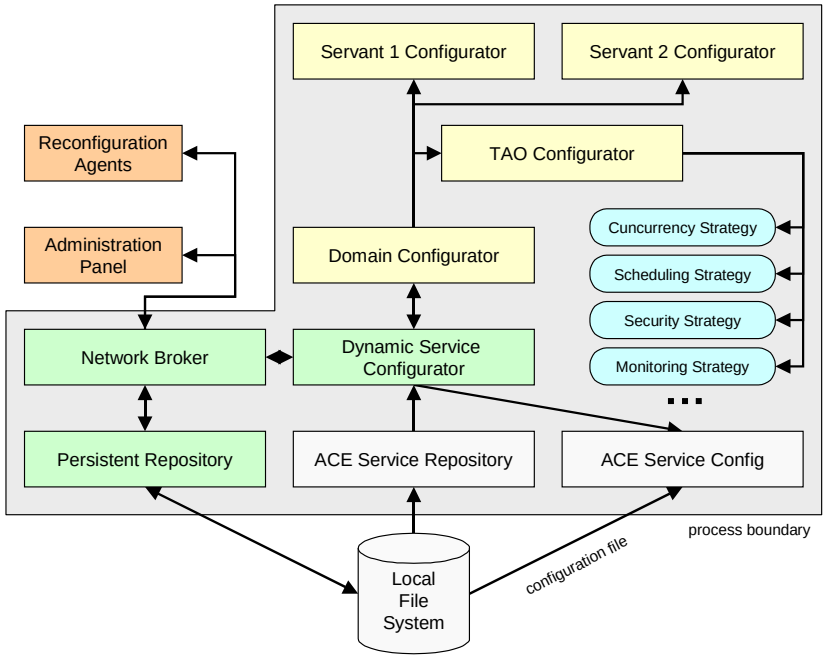

Abbildung 3.5: dynamicTAO Komponenten

basiert auf dem TAO-ORB, einem CORBA-ORB, der um den Aspekt der dynamischen Konfigurierbarkeit zur Laufzeit erweitert wurde.

Die Architektur von dynamicTAO ist in Abbildung 3.5 (nach Kon u. a., 2000c und Román u. a., 1999) dargestellt. Die Abhängigkeiten zwischen ORB-Komponenten und zwischen ORBs und Anwendungskomponenten werden durch Komponenten-Konfiguratoren in Form von C++ Objekten repräsentiert, die diese Abhängigkeiten als Liste von Zeigern zu anderen Komponenten-Konfiguratoren darstellen. Änderungsanforderungen führen zur Überprüfung der Abhängigkeiten unter Verwendung der Komponenten-Konfiguratoren-Objekte. Damit kann das sichere Laden und Entladen von Modulen zur Laufzeit garantiert werden.

In Umfeld von dynamicTAO wurde ein weiterer ORB namens *LegORB* entwickelt, dessen Ziel die Reduktion auf die notwendigsten Fähigkeiten einer dynamisch rekonfigurierbaren Middleware war und der minimale Speicher-Ressourcen in Anspruch nimmt. Beide Technologien sind Bestandteil der Entwicklung des *2K*-Betriebssystems, eines verteilten Betriebssystems[9].

[9] Kon u. a. (1999, 2000a)

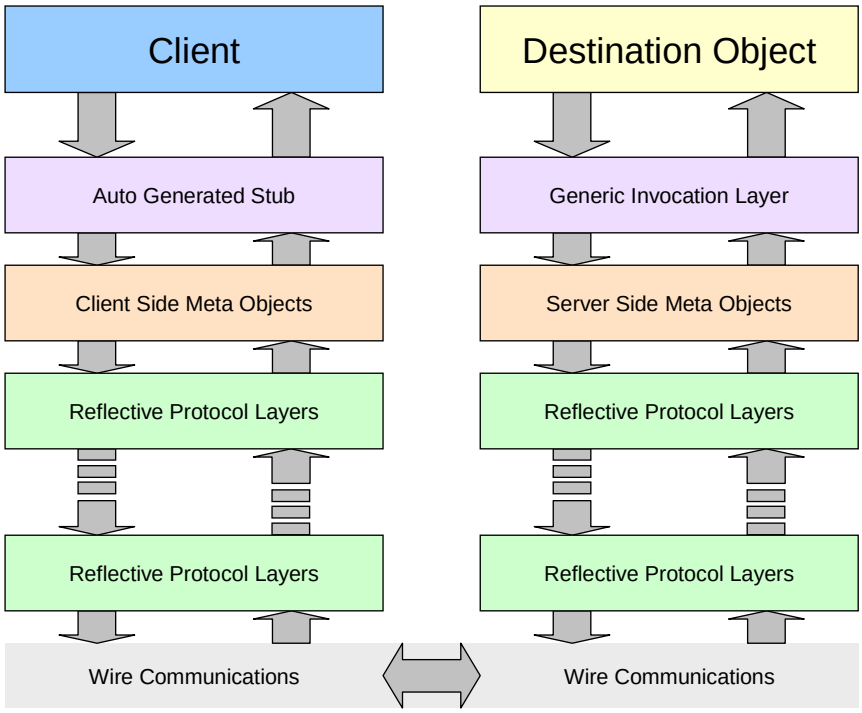

Abbildung 3.6: Ein reflektiver Protokoll-Stack

3.1.3 FlexiNet, FlexiBind und DIMMA

Eine auf Java basierende Middleware-Lösung ist *FlexiNet*[10]. Es ist eine Komponenten-orientierte, rekonfigurierbare und erweiterbare Middleware, die auch die Eigenschaften eines Binding-Frameworks liefert. FlexiNet besitzt eine eigene Form des RPC, die auf so genannten *Binders* basiert und für das unter anderem auch ein IIOP-Binder existiert, mit dem die Zusammenarbeit mit CORBA-Clients und -Servern möglich ist.

Die Entwicklung fußt auf den Erkenntnissen, die aus der Entwicklung der *DIMMA*-Plattform gewonnen werden konnten und basiert unter anderem auch auf den Konzepten der RM-ODP. Bindungen werden bei FlexiNet über lokale Proxy-Objekte hergestellt, die typischerweise Stub-Objekte sind. Abbildung 3.6 (nach Hayton, 1999) zeigt den FlexiNet-Protokoll-Stack. Die Ebenen des Stacks können dabei als Menge reflektiver Meta-Objekte gesehen werden, wobei jedes dieser Objekte die Daten verändern kann, bis schließlich das Zielobjekt mittels der Java-Reflection aufgerufen wird.

[10] Hayton u. a. (1999); Hayton (1999)

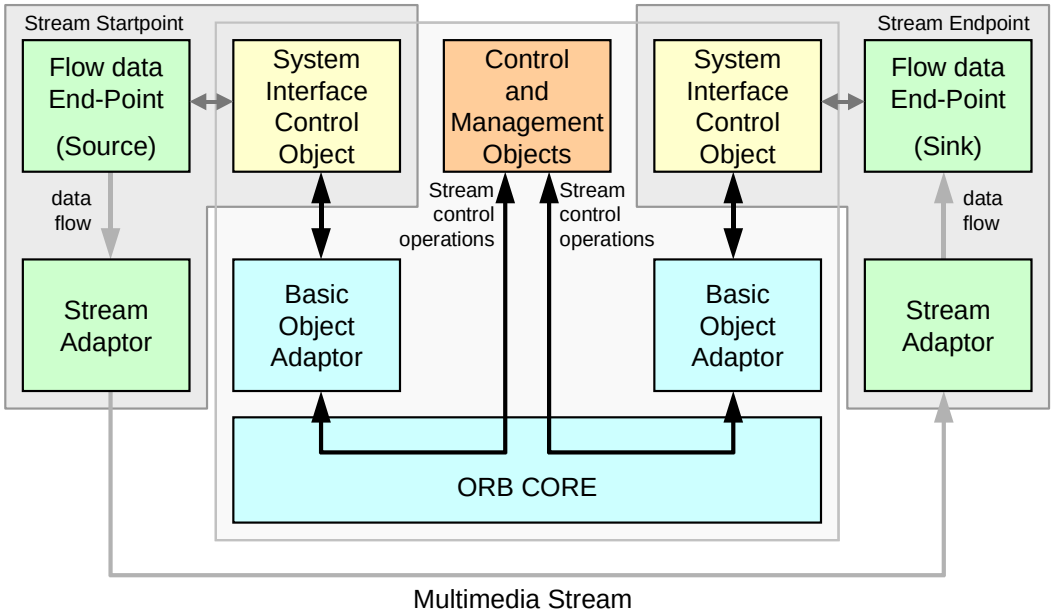

Abbildung 3.7: OMG Streams Architektur

Eine Erweiterung von FlexiNet ist die *FlexiBind*-Architektur[11]. Sie ermöglicht das dynamische Einbinden von Protokollen und Richtlinien (policies) in die Architektur. Sie erlaubt explizite Bindungen und dynamische Adaption von Protokoll-Stacks.

3.2 Multimedia-Plattformen

Eines der Ziele von Multimedia-Frameworks ist die Verarbeitung von Echtzeit-Datenströmen, wie sie z.B. bei Audio- und Video-Übertragungen vorkommen. Ein wichtiger Aspekt ist dabei der *Quality Of Service (QoS)*.

3.2.1 CORBA A/V Streams Spezification und TAO A/V Streams Service

Im Umfeld von CORBA wurde die *Audio/Video-Streams Specification* definiert[12], die Datenströme in CORBA ermöglicht. Abbildung 3.7 (nach Mungee u. a., 1999 und OMG, 2000)

[11] Øyvind Hanssen und Eliassen (1999)
[12] OMG (2000)

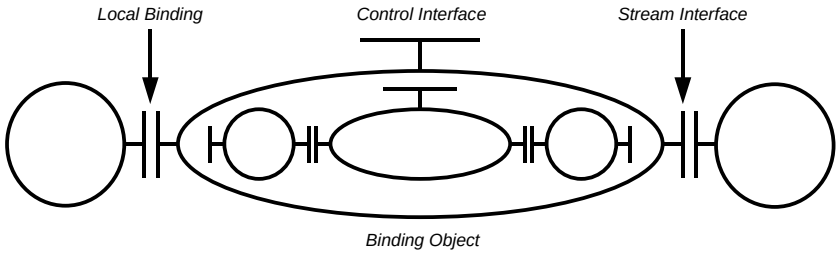

Abbildung 3.8: Stream Interface im Open Binding Modell

zeigt den prinzipiellen Aufbau dieser Technologie. Der Datenstrom ist nicht Teil der CORBA-Mechanismen zum entfernten Objekt-Zugriff, sondern dieses Verfahren dient nur zur *Kontrolle* eines separaten Datenstroms. In der Spezifikation wird zwischen *Flow* und *Stream* unterschieden. Ein Flow ist eine gerichtete 1:1-Kommunikation, ein Stream ist eine Zusammenfassung von einem oder mehreren Flows zwischen zwei Kommunikationspartnern.

Eine Umsetzung der CORBA A/V Streams Specification ist der *TAO Audio/Video Streaming Service*, der auf der Basis des TAO-ORB entwickelt wurde[13].

3.2.2 MULTE ORB

Bei dem *MULTE ORB*[14]. handelt es sich um einen Multimedia-ORB auf der Basis des Protokoll-Frameworks *Da CaPo* (siehe Abs. 3.3.2) und dem COOL-ORB. Gleichzeitig handelt es sich jedoch auch um eine reflektive Middleware.

Das Hauptkonzept sind explizite Stream-Bindungen, Stream-Schnittstellen und Flows. Reflektivität wird mittels eines Meta-Objekt-Protokolls ermöglicht über das die Bindungen und die involvierten Komponenten verfügbar gemacht werden (siehe Abb. 3.8 aus Eliassen u. a., 2000). Eine *Binding Factory (BF)* ist für die Erstellung von Bindungen verantwortlich. Sie verwendet hierzu das *Binding Protocol (BP)*. Manipulationen von existierenden Bindungen, wie z.B. das Hinzufügen, das Entfernen oder das Verändern der Position von Komponenten innerhalb der Struktur wird mittels des *Binding Mutator (BM)* durchgeführt. Letztendlich wird die Menge der Komponenten eines Bindungs-Objekts und der damit verbundene Objekt-Graf als *Configuration* bezeichnet.

Ein weiterer wichtiger Punkt von MULTE ist die Verwendung eines flexiblen Protokoll-Frameworks (in diesem Falle Da CaPo). Die Bedeutung dieser Technologie wird im Abschnitt 3.3 erläutert.

[13] Mungee u. a. (1999)
[14] Eliassen u. a. (2000); Kristensen und Plagemann (2000); Eliassen u. a. (2002)

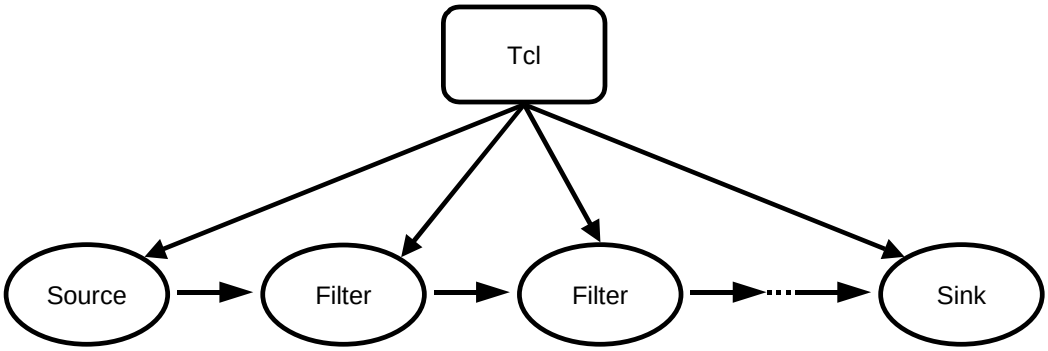

Abbildung 3.9: Die VuSystem Architektur

3.2.3 Mash, VuSystem und Indiva

Die *Mash*-Multimedia-Middleware[15], inzwischen unter dem Namen *OpenMash*[16] bekannt, basiert auf der *VuSystem*-Architektur (siehe Abb. 3.9 aus McCanne u. a., 1997). Multimedia-Daten werden von einem Source-Objekt generiert und durch ein oder mehrere Filter-Objekte geleitet. Optional konsumiert ein Senke-Objekt die gefilterten Daten. Bei den Objekten handelt es sich um C++-Objekte, deren Ablauf durch Tcl-Skripte gesteuert wird.

Eine Technologie, die Mash verwendet, ist *Indiva*, eine verteilte Streaming-Media- und Equipment Kontroll-Middleware[17].

3.3 Protokoll- und Kommunikationsframeworks

Ein Protokoll-Framework erlaubt die dynamische Auswahl, Konfiguration und Rekonfiguration von Protokoll-Modulen. Damit kann die Funktionalität von Protokollen erweitert oder an neue Gegebenheiten des Netzwerkes angepasst werden[18].

An dieser Stelle geht es nicht primär um Middleware, jedoch betrifft diese Technologie einen Aspekt, der auch bei Middlewaren zum Tragen kommen kann und das Streaming von Daten beeinflusst. Nicht ohne Grund basieren einige Multimedia-Frameworks auf Protokoll-Frameworks.

[15] Lindblad u. a. (1994); McCanne u. a. (1997)
[16] `http://www.openmash.org`
[17] Rowe u. a. (2002a,b)
[18] Eliassen u. a. (1999, 2000)

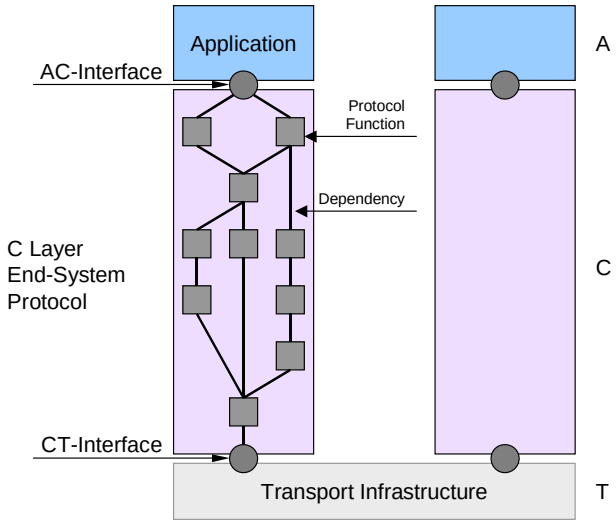

Abbildung 3.10: Das Da CaPo 3-Schichten Modell mit Beispiel-Protokoll-Graf

3.3.1 TAO Pluggable Protocol Framework

TAO's Pluggable Protocol Framework ist eine Erweiterung des CORBA-basierten TAO-ORBs um die Fähigkeit Protokolle hinzufügen zu können. Sie basiert auf zwei Hauptkomponenten: der ORB-Nachrichten-Komponente und der ORB-Transport-Adapter-Komponente. Mit beiden ist es möglich, die Kommunikationsinfrastruktur so zu erweitern, dass statische oder dynamische Bindungen neuer ORB-Nachrichten- oder ORB-Transport-Protokolle möglich sind[19]. Auch sind nun bezüglich verschiedener Anwendungsanforderungen optimierte inter-ORB Protokoll-Stacks möglich.

3.3.2 Da CaPo

Das *Da CaPo*-Projekt der ETH Zürich hatte zum Ziel protokollbedingte Performance-Probleme von End-Systemen aufzulösen[20]. Dazu wird das Kommunikationssubsystems in drei Schichten unterteilt (siehe Abb. 3.10 nach Vogt u. a., 1993). Die End-System-Kommunikation findet über die Transportschicht (T-Layer) statt. Im C-Layer wird die Funktionalität zur Unterstützung der End-zu-End Kommunikation festgelegt. Diese Funktionalitäten werden im AC-Interface der Anwendung (A-Layer) zur Verfügung.

Die C-Layer-Dienste werden als Menge von Protokoll-Funktionen in Form eines Proto-

[19] Kuhns u. a. (1999)
[20] Vogt u. a. (1993)

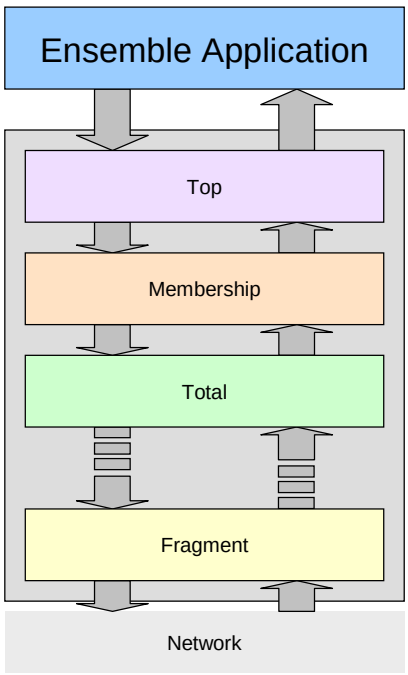

Abbildung 3.11: Die Ensemble-Architektur

koll-Grafen abstrahiert. In einem Konfigurationsprozess (CP) werden diejenigen Protokoll-Module ausgewählt, die bezüglich der Anforderungen der Anwendung am besten passen. Ein Verbindungsmanager (CM) sorgt dafür, dass bei einem neuen Verbindungsaufbau beide Seiten das gleiche Protokoll sprechen.

Das Nachfolge-Projekt *Da CaPo++* basiert auf Da CaPo und versucht ein Applikationsframework konkret für Bank-Umgebungen und Tele-Seminare zu entwickeln[21] oder allgemein auch für Multimedia-Anwendungen. Eine andere Technologie, die auf Da CaPo fußt, ist Mash aus Abschnitt 3.2.3.

3.3.3 Ensemble und Horus

Ensemble ist eine auf dem Gruppen-Kommunikationssystem *Horus*[22] basierende Technologie. Der Architektur (siehe Abb. 3.11 aus Liu u. a., 1999) liegt die Idee eines Protokoll-Stacks zugrunde, der sich aus Mikro-Protokoll-Modulen zusammensetzt und auf verschie-

[21] Stiller u. a. (1997)
[22] Renesse u. a. (1996)

dene Arten geschichtet werden kann[23]. Jedes Modul besitzt ein Top- und Bottom-Interface. Ein Bottom-Interface kommuniziert mit einem darunter liegenden Top-Interface auf Event-Basis mittels Event-Objekten. Das System stellt Mechanismen zur Verfügung, mit denen die Protokoll-Konfiguration auf ihre Benutzbarkeit hin überprüft werden kann.

3.3.4 Click

Der *Click Modular Router* ist eine Technologie zum flexiblen Aufbau und zur Konfiguration von Routern[24]. Das System wird mittels einfacher Elemente zusammengesetzt, um das gewünschte Verhalten zu erzeugen. Elemente besitzen ein oder mehrere Ports, die entweder vom Daten sendenden Push- oder vom Daten empfangenden Pull-Typ sind. Die Ports zweier Elemente können nur miteinander verbunden werden, wenn ein Push-Port mit einem Pull-Port verbunden ist. Datenpakete werden durch die Elemente geleitet, wobei das Element den passenden Ausgangsport auswählt.

Eine Routerkonfiguration wird durch einen gerichteten Grafen mit den Elementen als Knoten repräsentiert. Ein Click-IP-Router besteht beispielsweise aus nur 16 Elementen.

3.3.5 x-Kernel, Coyote und Scout

Bei *x-Kernel* handelt es sich um einen Betriebssystem-Kern, mit dem Netzwerkprotokolle konstruiert und zusammengestellt werden können[25]. Dabei werden Protokolle als Spezifikation von *Kommunikationsabstraktionen* gesehen, die mittels *Nachrichten* zwischen den Protokoll*beteiligten* ausgetauscht werden, deren Zustellung nicht garantiert wird und die entweder synchron oder asynchron ablaufen.

Drei Typen von Kommunikationsobjekten werden zur Verfügung gestellt: *Protokolle*, *Sitzungen* und *Nachrichten*. Die Objekte werden bezüglich statisch/dynamisch und passiv/aktiv klassifiziert. Protokoll-Objekte korrespondieren mit Netzwerk-Protokollen (z.B. IP, UDP, TCP) und sind statisch und passiv. Sitzungen sind Instanzen von Protokoll-Objekten und werden dynamisch erzeugt, sind jedoch passiv. Nachrichten sind aktive Objekte, die zwischen Sitzungs- und Protokoll-Objekten ausgetauscht werden.

Kommunikationsobjekte werden in Form von Pfaden im x-Kernel zusammengefügt. Ein Schlüssel-Aspekt des x-Kernel-Designs ist, dass Prozesse durch Nachrichten assoziiert werden und nicht durch Protokolle. Auf diese Art werden Nachrichten durch den gesamten Protokoll-Stack geführt ohne den größeren Overhead von Kontext-Wechseln[26].

[23] Liu u. a. (1999)
[24] Kohler u. a. (2000); Gottlieb und Peterson (2002)
[25] Hutchinson und Peterson (1991)
[26] Koster (2002)


```
component filter {
    require left1 <port, int>;
            left2 <port, int>;
    provide right <port, int>;
}
```

Abbildung 3.12: Beschreibung einer Filter-Komponente mittels Darwin

Eine Technologie, die auf x-Kernel aufbaut ist *Coyote*, ein System zur Erstellung von feinkörnig-konfigurierbaren Kommunikationsdiensten[27]. Sie setzt sich zusammen aus *Micro-Protokollen*, aus *zusammengesetzten Protokollen*, *Ereignissen* und einem *Laufzeit-Framework*.

Unter Verwendung des x-Kernel-Systems entstand auch eine weitere Technologie: *Scout*, ein pfadbasiertes Betriebssystem[28]. Pfade, in diesem Kontext, sind lineare Datenflüsse, die bei einer Quell-Einheit starten und in einer Ziel-Einheit enden. Pfade werden durch Instanzen von Protokoll-Modulen zusammengesetzt, die *Stages* genannt werden. Module können Teil verschiedener Pfade sein, jedoch gehört ein Stage genau zu einem Pfad. Pfade werden auf Anfrage zur Laufzeit aufgebaut.

3.4 Weitere Projekte und Prototypen

Neben den beschriebenen Technologiefeldern existieren noch weitere Projekte, aus deren Entwicklungen Prototypen entstanden sind. Einige dieser Prototypen wie GOPI (siehe Abs. 3.1.1) oder DIMMA (siehe Abs. 3.1.3) sind in zum Teil noch aktive Folgeprojekte aufgegangen. Im Folgenden werden weitere Prototypen aufgeführt, deren eindeutige Zuordnung zu den zuvor erwähnten Technologiefeldern schwer fällt, aber trotzdem Teilaspekte dieser Felder berühren.

3.4.1 Regis

Regis ist eine Programmierumgebung für die Entwicklung verteilter Anwendungen[29]. Der Schwerpunkt liegt in der Konstruktion von Programmen aus mehreren parallel arbeitenden Komponenten und der Unterstützung durch verschiedene Kommunikationsmechanismen und -primitive. Bei den Komponenten handelt es sich um Prozesse oder Threads auf der Basis von C++-Objekten, die mittels der Konfigurationssprache *Darwin* kombiniert wer-

[27] Bhatti u. a. (1998)
[28] Mosberger und Peterson (1996); Mosberger (1997); Gottlieb und Peterson (2002)
[29] Magee u. a. (1994b,a)

den. Komponenten stellen Dienste zur Verfügung und können selber auch Dienste in Anspruch nehmen. In Abbildung 3.12 (nach Magee u. a., 1994a) stellt die Komponente „filter“ den Dienst „right“ zur Verfügung und benötigt die Dienste „left1“ und „left2“. Durch die Vernetzung von mehreren Komponenten durch die Verbindung von Dienste-Konsument mit den entsprechenden Dienste-Anbietern können komplexe Programme aufgebaut werden.

Neben statischen Netzen bietet Regis auch die Möglichkeit der dynamischen Konfiguration von Netzen an. Damit sind Prozess-Strukturen möglich, die erst zur Laufzeit festgelegt werden.

3.4.2 Cool Jazz, Bossa Nova und Info Pipes

An der Universität Kaiserslautern sind im Rahmen des *Squirrel*-Projekts eine Reihe von Technologien entstanden.

Cool Jazz ist eine konfigurierbare Low-Level-Struktur, auf deren Basis spezifischere Middlewaren aufgebaut werden können.[30]. Cool Jazz unterstützt Nebenläufigkeit, Kommunikation und Signal-Verarbeitung innerhalb eines ereignisbasierten Verarbeitungsmodells.

Aufbauend auf dieser Struktur entstand das *Bossa Nova*-Kommunikationsframework[31]. Seine Flexibilität bezieht sich auf Protokoll-Strukturen, Protokoll-Granularität und Nebenläufigkeits-/Multiplexing-Strukturen.

Bossa Nova basiert auf einem Komponenten-Modell, in dem Komponenten logische Einheiten mit definierten Schnittstellen sind. Protokoll-Komponenten fassen Protokoll-Grafen der Einheiten zusammen. Schließlich werden Bindungen durch Multi-Grafen realisiert, also Grafen in denen zwei Knoten auch durch mehr als eine Kante miteinander verbunden sein dürfen. Das Framework nimmt für sich in Anspruch, einen hohen Grad an Flexibilität bei der Konfiguration und Rekonfiguration von Bindungen zu besitzen.

InfoPipes[32] baut ebenfalls auf Cool Jazz auf. Das Projekt hat die Entwicklung einer Middleware-Plattform für die Verarbeitung von kontinuierlichen Informationsflüssen zum Ziel. Infopipes behandelt die Komplexität von Kontrollflüssen und Nebenläufigkeiten, die bei der Verarbeitung von Datenfluss-Pipelines entsteht. Diese Pipelines bestehen aus einer *Informationsquelle*, möglichen *Transport-Einheiten*, *Filtern*, *Puffern*, *Transformationseinheiten* und Pumpen sowie einer *Informationssenke*.

Einfache Komponenten dieser Pipeline haben einen Informationseingang und einen Informationsausgang. Komplexere Komponenten, wie Verzweigungs- oder Zusammenführungseinheiten können mehrere Ein- oder Ausgänge besitzen.

Auf Grundlage der InfoPipe-Technologie wurde ein Prototyp entwickelt, der Video-Daten überträgt und anzeigt.

[30] Kramp und Koster (1999)
[31] Kramp und Coulson (1999)
[32] Koster (2002); Koster u. a. (2001, 2003)

Teil II

Der Smart Data Server: Konzeption einer neuen Middle-Tier Architektur

4 Zielvorgaben

Wenn wir die Ziele wollen, wollen wir auch die Mittel.

IMMANUEL KANT

Die in Teil I vorgestellten Lösungen decken ein großes Spektrum an Technologien ab, die im Kontext von Middleware-Plattformen und daran angrenzender Bereiche stehen. Trotzdem gibt es Gründe für die Entwicklung einer neuen Middleware. Dieses Kapitel beschreibt zunächst die Defizite an den bestehenden Lösungen und definiert anschließend neue Ziele. Jedes Ziel für sich betrachtet mag zunächst offensichtlich scheinen; die Kombination ist entscheidend.

Status Quo: Die Entwicklung von entfernten Funktionsaufrufen hin zum Zugriff auf entfernte Objekte und Komponenten ist eine konsequente Weiterentwicklung bestehender Technologien. Neben den Vorteilen bei der Entwicklung von verteilten Anwendungen führt dies jedoch auf der Server-Seite zu gesteigertem Verwaltungsaufwand. Während bei den entfernten Funktionsaufrufen das Problem hauptsächlich in der Suche nach der entsprechenden Funktion besteht, kommen bei entfernten Objekten und Komponenten Probleme hinzu, die mit deren Lebenszyklus einhergehen. Dies betrifft das dynamische Einbinden, die Instanziierung der Objekte/Komponenten, deren Verwaltung, ggf. mit persistenter Auslagerung sowie die Freigabe des Speichers, wenn keine Referenz mehr auf das Objekt/die Komponente besteht.

Technologien, die den Zugriffs auf Komponenten unterstützen, wie beispielsweise Enterprise Java Beans/J2EE, DCOM oder CORBA-basierte Server (z.B. MULTE ORB, dynamicTAO) müssen hierfür Server-Ressourcen zur Verfügung stellen, sei es durch zusätzliche Verwaltungseinheiten im Server, die als weitere Prozesse oder Threads laufen oder durch Belegung von Speicher durch die Instanziierung von Objekten oder Komponenten.

In vielen Fällen reicht der funktionale Ansatz jedoch vollständig aus, um praxisbezogene Lösungen zu entwickeln. Vergleichbar mit der MySQL-Datenbank, bei der anfangs bewusst auf die wichtige Eigenschaft der Transaktionsfähigkeit verzichtet wurde, öffnet eine solche Reduzierung auf hinreichende Eigenschaften den Weg für Optimierungen der internen Prozesse, Abläufe und Ressourcen.

$\Longrightarrow$ **Ziel 1:** Schaffung einer Middleware-Plattform auf der Basis entfernter Funktionsaufrufe.

Status Quo: Die Beschränkung auf entfernte Funktionsaufrufe darf nicht zulasten der Nutzungsfähigkeit der Middleware gehen. Im Kontext der entwicklerzentrierten Sichtweise auf die Middleware müssen alle notwendigen Strukturen in der Middleware vorliegen, die einen transparenten Zugriff auf das Umfeld der Middleware ermöglichen. Der Einsatz der Middleware mit der Umsetzung ihrer neuen Konzepte muss unter realen Bedingungen in Industrieprojekten möglich sein.

Viele Forschungsplattformen und Prototypen bleiben diesen Beweis schuldig.

$\Longrightarrow$ **Ziel 2:** Schaffung einer praxistauglichen Middleware.

Status Quo: Eine Entwicklung hin zu komplexeren Strukturen zeigt sich auch bei den Nachrichtenformaten, mit denen die Anfragen kodiert und zwischen Client und Server ausgetauscht werden. Die Spezifikation von SOAP beispielsweise beträgt inzwischen mehrere Dokumente. Je komplexer die übertragenen Nachrichten sind, umso mächtiger müssen die Kodierer- und Dekodiereinheiten sein. Dementsprechend effizient und klein können Kodierer und Dekodierer bei einfachen Spezifikationen entwickelt werden.

Web Service-fähige Server (z.B. .NET) müssen inzwischen neben den Bibliotheken zur Behandlung von SOAP auch die damit verbundenen Bibliotheken für XML, XML-Schema, HTTP und ähnlichem vorhalten. Auf der Seite des Servers mag dies unproblematisch sein. Diese Bibliotheken müssen jedoch auch in den Clients vorliegen. In einem Umfeld, in dem es auf die Größe einer Anwendung ankommt, kann das Verhältnis von eigentlichem Anwendungscode zu den benötigten Bibliotheken entscheidend sein (beispielsweise bei Applets).

$\Longrightarrow$ **Ziel 3:** Schaffung eines einfachen und leichtgewichtigen RPC-Nachrichten-Austauschformats und Nachrichten-Transport-Protokolls.

Status Quo: Die Beschränkung der Anfragen auf entfernte Funktionsaufrufe eröffnet einen weiteren Aspekt von Middleware-Architekturen: die Vernetzung von mehreren Servern. Moderne Server-Architekturen bieten Load-Balancing-Mechanismen zur Verteilung von Lasten zwischen mehreren Servern an, jedoch macht sich der zuvor beschriebene Aufwand bei der Verwaltung von Objekten oder Komponenten auch hier bemerkbar. Bei rein funktionalen Aufrufen sind wesentlich einfachere Anfrage-Weiterleitungsmechanismen denkbar, die den Aufbau von Middleware-Netzen erleichtern.

$\Longrightarrow$ **Ziel 4:** Schaffung eines einfachen Mechanismuses zur Weiterleitung von Anfragen über mehrere Middleware-Server.

Status Quo: Entfernte Zugriffe von einem Client auf eine Middleware folgen meist dem Anfrage/Antwort-Prinzip. Die Übertragung von größeren Daten-Mengen in Form von Streaming-Daten ist in diesen Fällen nicht Teil der Anfrage, sondern erfolgt dann über einen zweiten dedizierten Streaming-Kanal, der durch weitere Anfragen gesteuert wird. Ein Beispiel hierfür ist der TAO A/V Streams Service auf Basis der CORBA A/V Streams Specification.

Das Streaming von Daten während der Anfrage-Erstellung mit möglicherweise gleichzeitiger Antwort-Entgegennahme liegt nicht vor. Vorteilhaft wäre dies beispielsweise dann, wenn schon Teilergebnisse erstellt werden können, bevor die Anfrage vollständig übertragen wurde. Die Datenmengen, die im Server vorgehalten werden müssen, können so optimiert werden. Die fehlende Fähigkeit zur Bearbeitung derartiger Anfragen liegt unter anderem an Beschränkungen des Nachrichtenformat, beispielsweise bei XML-RPC, SOAP oder CORBA.

Andere Technologien sehen die streamingbasierte Anfrage nicht vor, wie beispielsweise RMI, SUNs RPC oder der RPC-Mechanismus von DCE. Hier muss der Datenstrom als Serie von Anfragen simuliert werden. Eine Asynchronität von Anfrage-Datenstrom und Antwort-Datenstrom[1] ist auf diese Weise schwer zu implementieren.

$\Longrightarrow$ **Ziel 5:** Schaffung eines streamingfähigen RPC-Mechanismuses zur parallelen Anfrageerstellung und Antwortentgegennahme.

Status Quo: Neben einem streamingfähigen Protokoll und Nachrichtenaustauschformat muss der Server Strukturen besitzen, die solche Anfragen behandeln können. Protokollframeworks wie Da CaPo, Ensemble oder das TAO Pluggable Protocol Framework geben einen ersten Hinweis auf die Richtung, in der mögliche Lösungen liegen, setzen ihren Schwerpunkt jedoch in der Abarbeitung von Protokollen, einem Teilaspekt von Middlewaren. Ähnlich verhält es sich bei Scout oder Coyote, die als Inspiration dienen können, aber nicht als Strukturen für Middlewaren konzipiert sind.

Multimedia-Plattformen sind zwar auf das Streaming von Daten ausgelegt, behandeln sie jedoch meist nicht als Teil eines Anfrage- und Antwort-Vorganges, sondern leiten sie durch eine Reihe von Filtern, beispielsweise bei Mash oder VuSystem.

Die reflektiven Middlewaren wie OpenORB oder FlexiNet können in diesem Punkt auch nicht überzeugen. Ihr Schwerpunkt liegt in der Bereitstellung neuer Bindungstypen und in der Reflektivität des inneren Aufbaus, aber weniger in der Behandlung von streamingbasierten Funktionsaufrufen.

$\Longrightarrow$ **Ziel 6:** Schaffung einer Serverstruktur, die das Abarbeiten von streamingbasierten Anfragen ermöglicht.

Status Quo: Reflektive Middleware-Architekturen sind zur Zeit ein stark beachteter Zweig der Middleware-Forschung. Der Ansatz der *causally connected self representation* (CCSR), die den Aufbau eines Servers mit einer Struktur assoziiert, deren Veränderung auch den Aufbau des Servers beeinflusst, ist richtungsweisend. Allerdings wird diese Flexibilität meist durch die Festlegung von statischen Abhängigkeiten der Server-Komponenten beschränkt. Dies führt zu komplexeren Mechanismen beim Aufbau der Server-Struktur und deren Überprüfung auf Konsistenz sowie zu Limitationen in den Komponenten selbst. Hier

[1] Es besteht kein Zusammenhang zwischen dem Zeitpunkt der Datenübertragung zum Server und der Datenübertragung vom Server.

wäre eine noch stärkere Entkopplung der Server-Komponenten untereinander durch die Beseitigung der statischen Abhängigkeiten wünschenswert.

Das Ziel der reflektiven Middlewaren zur Schaffung einer einfach erweiterbaren und anpassbaren Serverstruktur, um an unterschiedliche Einsatzumgebungen angepasst werden zu können, bleibt dabei unberührt. Die Behandlung von Datenströmen (Ziel 6) muss jedoch berücksichtigt werden.

$\Longrightarrow$ **Ziel 7:** Schaffung einer flexiblen und erweiterbaren Server-Struktur mit maximal unabhängigen Server-Komponenten.

In den nun folgenden Kapiteln werden Konzepte und Lösungen vorgestellt, die diese Ziele in einer Middleware-Architektur, dem Smart Data Server, umsetzen.

5 Entfernte Aufrufe

Denn es ist zuletzt doch nur der Geist, der jede Technik lebendig macht.

JOHANN WOLFGANG VON GOETHE

Damit ein Client Funktionen des Servers in Form eines RPCs nutzen kann, muss dieser dem Server den Nutzungswunsch in einer geeigneten Form mitteilen können. Entsprechendes gilt für die Antwort des Servers an den Client. Die Kommunikation vollzieht sich dabei auf zwei Ebenen (siehe Abb. 5.1): Auf der Ebene der zu übertragenden Nachricht, in der die Frage und die Antwort kodiert ist, sowie der Ebene des Sitzungssprotokolls, mit dem die Nachricht übertragen wird. Die Trennung in Nachricht und Protokoll ist sinnvoll, um einen späteren Austausch einer der beiden Kommunikationsebenen ohne Beeinträchtigung der anderen Ebene zu ermöglichen.

In Rahmen der SDS-Entwicklung sprachen mehrere Punkte des Anforderungskatalogs für die Eigenentwicklung des Nachrichtenformats und des Sitzungssprotokolls. Einerseits bestand die Anforderung für einen leichtgewichtigen, einfach zu implementierenden RPC, dessen Code-Basis selbst ein Applet in seiner Größe nicht wesentlich vergrößert. Andererseits sollte der RPC-Mechanismus streamingfähig sein. Dies ist so in bestehenden Lösungen wie CORBA oder RMI nur durch Umwege zu erreichen. Letztendlich war ein Ziel, den RPC-Mechanismus sowie die internen Datenflüsse des Servers, die bei der Abarbeitung des RPCs involviert sind, aufeinander abzustimmen und zu optimieren. Dies erfordert die Möglichkeit zu Anpassungen und Veränderungen im Nachrichtenformat oder Sitzungsprotokoll, die bei existierenden und standardisierte RPC-Lösung nicht besteht.

Dieses Kapitel beschreibt zunächst in Abschnitt 5.1 das Sitzungsprotokoll sowie in Abschnitt 5.2 den Aufbau der Anfrage- und Antwort-Nachricht. Die Client- und Server-seitigen Entwicklungsschritte werden in Abschnitt 5.3 beschrieben. Mit welchen Klassen von Fehlern während des RPCs zu rechnen ist, zeigt Abschnitt 5.4. Schließlich endet das Kapitel mit Abschnitt 5.5 und der Streaming-Variante des RPCs aus Sicht des Clients.

5.1 Das Sitzungsprotokoll

Das Protokoll zum Austausch der Anfrage- und Antwort-Nachrichten zwischen Client und Server heißt *IPTP (Information Packet Transfer Protocol)*[1]. Es orientiert sich im Aufbau an

[1] Roth u. a. (1999a); Huang u. a. (2002)

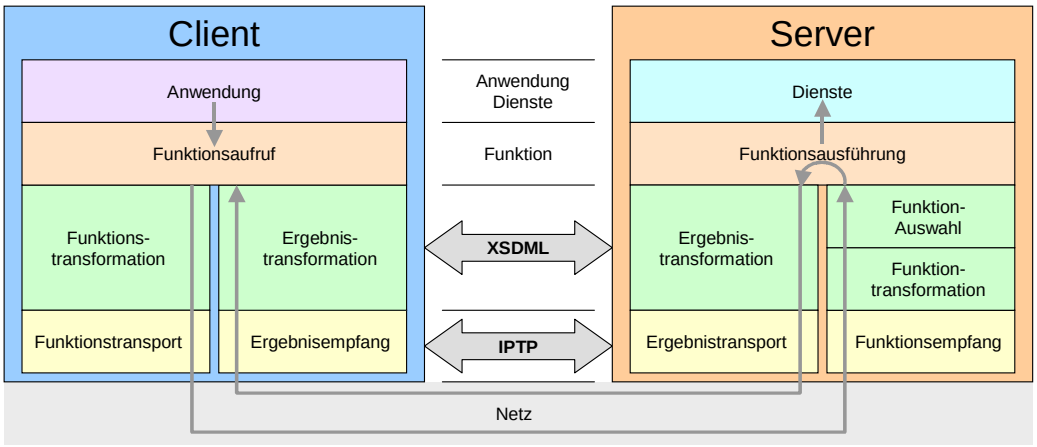

Abbildung 5.1: Protokoll und Nachricht des Servers

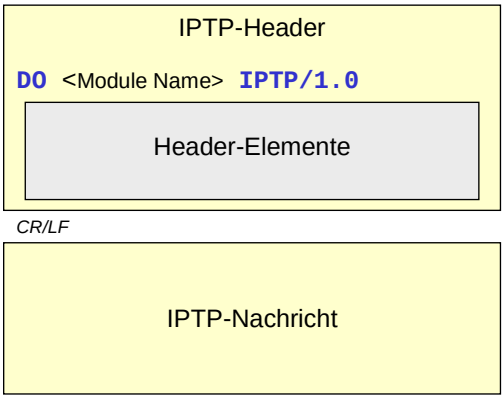

Abbildung 5.2: IPTP-Anfrage

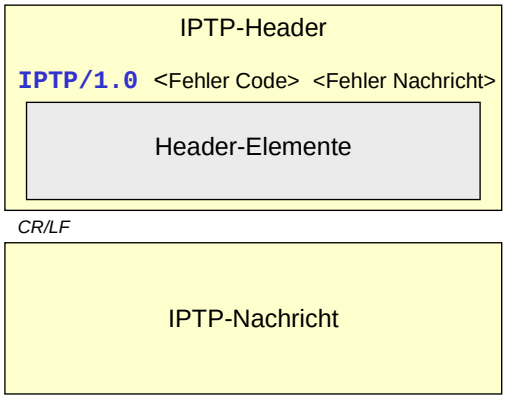

Abbildung 5.3: IPTP-Antwort

HTTP. Dabei beschränkt es sich in seinem Umfang nur auf einen kleinen Teil der HTTP-Spezifikation, ist jedoch im Gegensatz zu HTTP ein statusbehaftetes Protokoll.

Das Protokoll ist Text-basiert und gliedert sich in zwei Bereiche, den IPTP-Header und die IPTP-Nachricht (siehe Abb. 5.2/5.3). Beide Bereiche sind durch eine Leerzeile voneinander getrennt.

```
DO MyModule IPTP/1.0
CONNECTION: CLOSE
USER: ClientUser
CONTENT-TYPE: text/xsdml
CONTENT-LINES: 72
```

Code 5.1: IPTP-Header der Anfrage

Die angesprochenen Funktionen des Servers werden in Modulen bereitgestellt. In einer Anfrage folgt in Zeile 1 des Headers auf die Anweisung `DO` der Modul-Name und die Angabe der Protokoll-Version (`IPTP/1.0`) (siehe Code 5.1). Zur Zeit ist die Version 1.0 die einzig gültige und spezifizierte. Die folgenden Zeilen beinhalten die Header-Elemente mit Informationen in der Form `Schlüsselwort: Wert`. Die Reihenfolge dieser Elemente ist nicht festgelegt. Tabelle B.1 im Anhang B auf Seite 145 führt die möglichen Elemente auf.

```
IPTP/1.0 200 OK
DATE: 2003-06-18 11:57:02 CEST
SERVER: SDS1
SESSION-ID: 20030618115701898-4b9aa32d0790
CONTENT-TYPE: text/xsdml
CONTENT-LINES: 84
```

Code 5.2: IPTP-Header der Antwort

Die Antwort des Servers wird mit einer Status-Zeile eingeleitet (siehe Code 5.2). Sie liefert die Protokoll-Version der Antwort (`IPTP/1.0`), gefolgt von einem Fehlercode und einer Fehlerbeschreibung. Die Tabelle B.2 führt die aktuell unterstützten Fehlercodes auf.

Neben der reduzierten Anzahl von Header-Elementen unterscheidet sich IPTP in zwei Punkten wesentlich von HTTP: Zum einen wird die Nachrichtengröße nicht in Bytes durch `CONTENT-LENGTH` angegeben, sondern durch `CONTENT-LINES` in Zeilen. Dies ist möglich, da die gesamte Kommunikation zwischen Client und Server auf Text-Basis durchgeführt wird, wobei Zeilengrenzen (*CR/LF*) die Informationen des IPTP-Headers sowie die IPTP-Nachricht unterteilen. Moderne Programmiersprachen unterstützen den Entwickler bei der Zeilen-orientierten Verarbeitung von Datenströmen, was eine Implementierung des Protokolls erleichtert.

Typ	XML-Tag	Bedeutung	Beispiel
`Boolean`	<b>	Wahr oder falsch	0
`Integer`	<i>	Ganzzahl mit Vorzeichen	-1234
`Real`	<r>	Gleitkomma-Zahl mit Vorzeichen	1.234
`String`	<s>	ASCII-String	dies ist ein String
`Base64`	<b64>	Base64-kodierte Binärdaten	SWNoIGxpZWJlIElyaXM=
`Null`	<n>	NULL-Wert	null

Tabelle 5.1: Skalare Datentypen des SDS

Zum anderen ist mit der Einführung der `SESSION-ID`, ein (im Vergleich zu Cookies) einfacher Mechanismus für die sitzungsbehaftete Kommunikation zwischen Client und Server.

Neben den definierten Header-Elementen können weitere Elemente frei definiert werden. Da während der Abarbeitung einer Anfrage im Server alle Header-Elemente stets abrufbar sind, steht damit die Möglichkeit offen, auf bestimmte Aspekte der Server-internen Bearbeitung Einfluss nehmen zu können. Dies geschieht allerdings nur dann, wenn eine Bearbeitungskomponente des Servers dies auch vorsieht.

5.2 Die RPC-Nachricht

Die IPTP-Nachricht dient der Kodierung der RPC-Anfrage und -Antwort, die zwischen Client und Server innerhalb einer IPTP-Sitzung ausgetauscht wird. Für diesen Zweck wurde eine eigene XML-Sprache entwickelt: XSDML (XML-based Smart Dataserver Markup Language). Mit ihr ist es möglich, gewünschte Funktionen auf dem Server anzusprechen und die dazu notwendigen Parameter und Daten zu kodieren. Entsprechendes gilt für die Ergebnisse des Servers. Da die Kodierung des RPC ausschließlich auf XSDML basiert, wird die IPTP-Nachricht im Folgenden als XSDML-Nachricht bezeichnet.

5.2.1 Datentypen

Innerhalb der Anfrage und Antwort wird zwischen Parameter und Streaming-Daten unterschieden. Parameter definieren einen festgelegten Satz von Informationen, während Streaming-Daten eine nicht quantifizierbare Menge von Informationen gleicher Art sind.

In beiden Fällen liegen drei Arten von Datentypen zu Grunde: *Skalare*, *Listen* und *assoziative Arrays*[2].

[2] Roth u. a. (1999b)

Die skalaren Datentypen sind in Tabelle 5.1 aufgeführt und stellen die Grunddatentypen dar. Der Wertebereich und die Genauigkeit von `Integer` und `Real` sind nicht vorgegeben, die Empfängerseite ist angehalten, den ihr zur Verfügung stehenden Datentyp auszuwählen, der die Zahl mit den gegebenen Informationen abbilden kann.

Da das Nachrichtenformat auf Text basiert, müssen Binärdaten als Base64-codierter Text übertragen werden. Hierfür steht ein eigener Datentyp zur Verfügung.

Eine Besonderheit stellt `Null` dar, die kein eigener Datentyp, sondern ein spezieller Wert ist: den NULL-Wert. Somit ist die Übermittlung von NULL-Referenzen möglich, was z.B. bei XML-RPC nicht möglich ist.

```
<list>
    <le> ... </le>
    <le> ... </le>
    <le/>
</list>
```

Code 5.3: XSDML Liste

Ein komplexer Datentyp ist die Liste (siehe Code 5.3: `list`). Die Reihenfolge der Listenelemente (List-Element = `le`) ist von Bedeutung. Jedes Listenelement selbst kann wiederum skalar, eine Liste oder ein assoziatives Array sein. Im Falle von `<le/>` ist das Listenelement NULL.

```
<map>
    <mi name="key1"> ... </mi>
    <mi name="key2"> ... </mi>
    <mi name="key2"/>
</map>
```

Code 5.4: XSDML Assoziatives Array

Der zweite komplexe Datentyp ist das assoziative Array (siehe Code 5.4: `map`). Hier ist die Reihenfolge der Array-Elemente (Map-Item = `mi`) nicht entscheidend. Wichtig ist, dass jedes Element mit einem eindeutigen Namen referenziert wird. Entsprechend der Listenelemente können die Array-Elemente allen gültigen Datentypen entsprechen. Das Array-Element, das mit `<mi name="..."/>` referenziert wird, entspricht der NULL.

5.2.2 Aufbau einer XSDML-Nachricht

Eine Client-Anfrage unterteilt sich in fünf Bereiche (siehe Code C.1 im Anhang C auf Seite 147), deren Reihenfolge festgelegt ist:

- Der Spezifikation der gewünschten **Funktion** (`function`),
- einem **Parameter-Bereich** (`reqParaOut`),
- einem **Ergebnis-Rückgabebereich**, in dem die gewünschten Rückgabewerte spezifiziert werden können (`resRsltIn`),
- einem **Streaming-Daten-Rückgabebereich**, in dem das Format der gewünschten Streaming-Daten definiert werden kann (`resDataIn`)
- und einem **Streaming-Datenbereich** (`reqDataOut`).

Der Parameter-Bereich kann entweder eine Liste oder ein assoziatives Array sein. Versucht der Server eine passende Funktion durch Reflection zu ermitteln, so steht als Information nur deren Signatur, also die Anzahl der Parameter sowie deren Datentypen zur Verfügung. Die Bezeichner der Parameter können nicht ermittelt werden. Die Angabe der Parameter als assoziatives Array mit Parameter-Bezeichnern ist aus diesem Grunde nicht möglich. In diesem Fall müssen die Parameter in der geforderten Reihenfolge (entsprechend der Signatur der Funktion) als Liste übertragen werden.

Werden die Parameter (wie im Code-Beispiel) als assoziatives Array übermittelt, müssen nicht notwendigerweise alle möglichen Parameter übermittelt werden. Nicht spezifizierte Parameter werden auf der Server-Seite durch Default-Werte belegt oder führen zu einem Fehler.

Zur besseren Ausnutzung der Funktionen wird im Anfrage-Dokument die Struktur der Antwort im Ergebnis-Rückgabebereich definiert. Nur die Bezeichner des assoziativen Arrays sind in diesem Falle von Bedeutung, die Array-Elemente sind NULL.

Den Ergebnis-Rückgabebereich als assoziatives Array zu definieren, macht aus zwei Gründen Sinn. Bei Aufruf der Funktion auf der Client-Seite wird die Rückgabe-Struktur erzeugt und ist nach deren Beendigung entsprechend gefüllt. Zum anderen ist eine Eigenschaft von assoziativen Arrays die Eindeutigkeit der Bezeichner. Es bietet sich an, auf Client- und Server-Seite Hash-Funktionen zu verwenden, womit der Zugriff auf die Array-Elemente über die Bezeichner schneller durchgeführt werden kann als bei der Durchsuchung von Listen.

Der Streaming-Datenbereich dient der Übertragung von Datenmengen, deren Größe nicht festgelegt ist und die als Datenströme vom Server ausgewertet werden können. Der Streaming-Daten-Bereich ist als Liste von Datensätzen definiert. Da der Streaming-Datenbereich immer der zuletzt übertragene Bereich ist, ist bei dessen Erreichen die Funktion durch die Parameter und die Rückgabebereiche soweit spezifiziert, dass eine Ausführung auf jedem einzelnen Datensatz möglich ist. Die Anfrage muss also nicht notwendigerweise vollständig in den Server übertragen werden, bevor eine Ausführung der Funktion initiiert wird, sondern die Funktion kann nur so viele Daten des Streaming-Datenbereichs auswerten, wie sie aktuell benötigt.

Zum Beispiel muss zur Berechnung des Durchschnitts von 1 Mio. Zahlen deren Gesamtsumme durch die Anzahl der Zahlen geteilt werden. Die Funktion muss somit immer nur den nächsten Wert auslesen und zu der bisherigen Gesamtsumme hinzu addieren sowie den Nummern-Zähler um eins erhöhen. Der Server wird also nur mit jeweils einem Datensatz belastet und nicht etwa mit 1 Mio. Zahlen. Sollen jedoch die 1 Mio. Zahlen in umgekehrter Reihenfolge wieder ausgegeben werden, so ist ein optimiertes Einlesen der Werte nicht möglich und alle Zahlen müssen im Server vorliegen.

Da als Rückgabe unter Umständen auch Streaming-Daten erwartet werden, kann in einem weiteren Bereich die Struktur der Datensätze der Listen-Elemente definiert werden. Als Beispiel sei hier eine Anfrage von Kursdaten einer gewünschten Aktie genannt. Möglicherweise wünscht man sich neben dem Preis der Aktie auch Indikatoren, wie z.B. den 38-Tage gleitenden Durchschnitt oder tageshöchst und -niedrigst-Stände. Als Antwort auf die Anfrage wird im Streaming-Datenbereich dann eine Liste von assoziativen Arrays übertragen, wobei jedes Array-Element die gewünschten Informationen beinhaltet.

Ähnlich der Anfragestruktur ist die Antwortstruktur aufgebaut (siehe Code C.2), die der Server liefert. Hier sind drei Bereiche definiert:

- Die Bezeichnung der aufgerufenen **Funktion** (`result`),
- der **Streaming-Datenbereich** (`resDataOut`), wiederum zur Übertragung größerer Datenmengen als Liste und
- der **Rückgabe-Bereich** (`resRsltOut`) mit dem assoziativen Array der gewünschten Ergebnisse.

Im Gegensatz zur Aufrufstruktur folgt der Rückgabebereich auf den Streaming-Datenbereich. Dies ist sinnvoll, da Ergebnisse meist erst nach der Übertragung aller Daten vorliegen.

5.3 Aufruf einer entfernten Funktion

Wichtig für den entfernten Funktionsaufruf sind die Übergabepunkte zwischen der Client-Anwendung und RPC-Infrastruktur sowie zwischen RPC-Server und der Funktion. Dieser Abschnitt beschäftigt sich mit den notwendigen Aktionen des Entwicklers.

Da es sich bei der Middleware um einen Java-Server handelt, sind die aufgeführten Code-Fragmente in dieser Sprache aufgeführt.

5.3.1 Client-Seite

Der Aufruf einer Funktion besitzt nicht die gleiche Transparenz, wie sie z.B. durch RMI oder CORBA gegeben ist, da Teile des Marshalling (also des Verpackens der Anfrage) vom

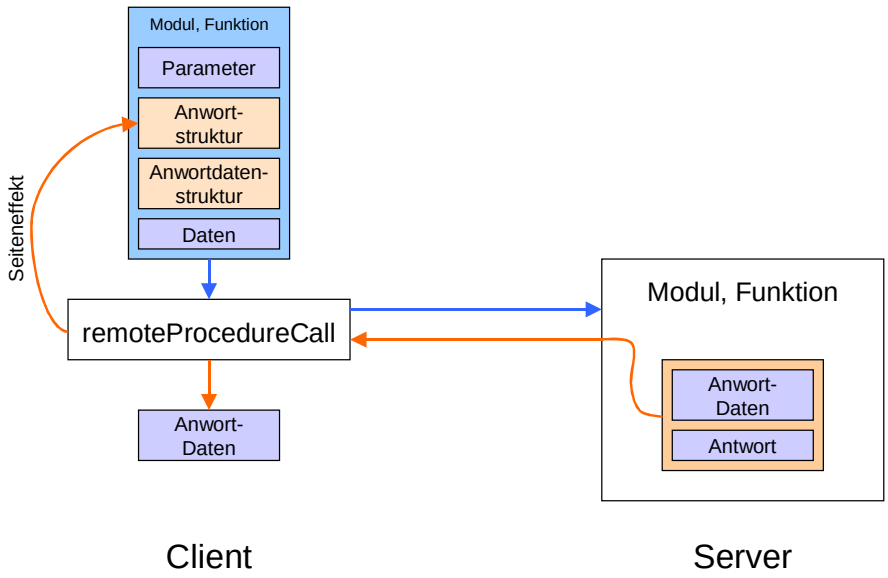

Abbildung 5.4: Entfernter Funktionsaufruf

Anwendungsentwickler selber übernommen werden müssen. Demgegenüber steht der Gewinn an Flexibilität bei der Bestimmung der Antwortstruktur oder der Parameterzusammensetzung. Ebenso müssen keine Interfaces definiert und mittels eines Generierungsprogramms in Stubs oder Skelleton überführt werden.

Als Schnittstelle zum RPC dient die Klasse `SDSBasicConnection`. Sie übernimmt einige grundlegende, immer wiederkehrende Abläufe und unterstützt bei der Abarbeitung von Fehlern. (siehe Abs. 5.4 auf Seite 70). Die zentrale Methode von `SDSBasicConnection` ist `remoteProcedureCall` (siehe Code 5.5). Abbildung 5.4 visualisiert deren Stellung zwischen Client und Server.

```
public List remoteProcedureCall (
   String module,
   String function,
   Map reqParaOutMap,
   Map resRsltInMap,
   Map resDataInMap,
   List reqDataOutList )
   throws SDSConnectionException
```

Code 5.5: Methode `remoteProcedureCall`

resDataInMap	**Rückgabe-Liste**
`NULL`	`NULL`
`{("key", NULL)}`	`[val1, val2, val3,...]`

`{`
` ("key1", NULL)`
` ("key2", NULL)`
` ...`
`}`
	`[`
	` {("key1", val1.1), ("key2", val2.1), ...},`
	` {("key1", val1.2), ("key2", val2.2), ...},`
	` {("key1", val1.3), ("key2", val2.3), ...},`
	` ...`
	`]`

Tabelle 5.2: Aufbau der Rückgabedaten in Abhängigkeit zur Vorgabe

Die Methode *remoteProcedureCall* basiert auf den Java-Sockets. Sockets stellen eine End-zu-End-Verbindung zu dem Socket des Servers her und liefern jeweils einen Eingabe- und Ausgabestrom, über den Daten in beide Richtungen übertragen werden können. Verbindungen, die auf diese Art hergestellt werden, garantieren, dass bei der Übertragung zum jeweils gegenüberliegenden Kommunikationspartner keine Daten aus dem Datenstrom verloren gehen können und im Fehlerfall entsprechende Exceptions ausgelöst werden.

Neben dem Bezeichner der aufzurufenden Funktion (`function`) und dem Modul des Servers, das diese Funktion zur Verfügung stellt (`module`), folgen alle Angaben, die auch bei der Spezifikation des Nachrichtenformates XSDML eingeführt wurden. Listen werden durch `java.util.List`, assoziative Arrays durch `java.util.Map` abgebildet.

Die zu übertragenen Parameter werden mit `reqParaOutMap` definiert, die Struktur der gewünschten Antwort mit `resRsltInMap`, die Struktur der Listenelemente der Streaming-Antwortdaten mit `resDataInMap`[3] sowie die Streaming-Daten, die zur Funktion übermittelt werden sollen mit `reqDataOutList`. Die Übertragung der Daten zum Server als Stream wird dabei von der Methode übernommen. Der Antwort-Datenstrom wird komplett eingelesen und als Rückgabewert der Methode geliefert. Die Streaming-Variante dieser Methode erfolgt in Abschnitt 5.5 auf Seite 71.

Der Rückgabewert der Methode hängt von der Struktur des Objekts `resDataInMap` ab, mit dem die Streaming-Eingabedaten spezifiziert werden. In Tabelle 5.2 sind alle drei Möglichkeiten hierzu beispielhaft aufgeführt. Ist das Objekt `NULL`, so sind keine Streaming-Daten gewünscht. Ist nur eine Informationsart gewünscht, so werden diese in einer Liste übermittelt. Sind verschiedene Informationsarten gewünscht, ist jedes Listenelement ein Datensatz (eine Map) mit den gewünschten Informationen. Dies bedeutet aber auch: Die Gesamtzahl der Informationen pro Art darf sich nicht unterscheiden und muss für einen Datensatz immer vollständig vorliegen.

Die Struktur der gewünschten Antwort (`reqRsltInMap`) wird in einem Map-Objekt

[3] Die Methode liefert als Rückgabewert ein Objekt vom Typ `java.util.List`.

abgebildet. Die Schlüssel legen fest, welche Informationen gewünscht sind. Die Methode `remoteProcedureCall`, hat einen Seiteneffekt bezüglich dieses Parameters. Die geforderten Rückgabewerte sind nach der Ausführung mit den Ergebnissen belegt. Sie können, wie gewohnt, über die Schlüssel aus der Map ausgelesen werden. Wurde bei der Initialisierung der ursprünglichen Map jedoch neben den Schlüsseln auch schon leere Listen oder Maps übergeben (weil die Antworten den entsprechenden Datentyp besitzen), so sind diese nach der Ausführung mit den gewünschten Informationen belegt.

Im Beispiel-Code D.1 im Anhang D auf Seite 151 ist die Map `infosMap` nach Aufruf von `remoteProcedureCall` mit allen verfügbaren Informationen über „Peter“ gefüllt und muss nicht erst über

```
infosMap = (Map) resRsltInMap.get ("Infos");
```

ermittelt werden. Diese Möglichkeit besteht nur bei Listen oder Maps, nicht jedoch bei skalaren Datentypen, wie z.B. bei Integer, Real oder String. Diese müssen weiterhin über die `get`-Methode von `resRsltInMap` abgerufen werden. Der Grund hierfür ist, dass in Java keine Möglichkeit existiert, entsprechende Objekte mit neuen Werten zu belegen. So ist z.B. `integerObject.setValue(18);` nicht möglich.

5.3.2 Server-Seite

Auf der Server-Seite wird die gewünschte Funktion mit der Signatur aus Beispiel-Code 5.6 aufgerufen. Sie unterscheidet sich gegenüber dem Aufruf auf der Client-Seite von `remoteProcedureCall` nur dadurch, dass zusätzlich der IPTP-Header, die Sitzungs-ID sowie ein Logging-Objekt übergeben wird. Über das Logging-Objekt kann die Funktion bei Bedarf Meldungen (Infos, Warnungen, Fehler) in die Server-Log-Datei schreiben (siehe Abs. 7.1 auf Seite 84).

Die Angabe der Modul- und Funktionsbezeichnung ist an dieser Stelle nicht notwendig, allerdings kann es sinnvoll sein, die eigene Bezeichnung zu kennen, z.B. bei der Erzeugung von Fehlermeldungen.

Die Sitzungs-ID ist notwendig, damit die Funktion die Client-Sitzung ermitteln kann. Dies ist nötig, wenn der Funktionsaufruf Teil einer Reihe von Anfragen an den Server ist und nur im Kontext der anderen Anfragen ausgewertet werden kann. Die ID entspricht im IPTP-Protokoll dem Header-Element `SESSION-ID` (siehe Tab. B.1 auf Seite 145).

Mit der Übergabe der Parameter und der gewünschten Antwort-Struktur jeweils als Map erhält man eine große Flexibilität bei der Ausgestaltung der Funktion. Allerdings muss dafür ein erhöhter Aufwand bei der Auswertung der Parameter und der Antwort-Struktur betrieben werden.

Im ersten Schritt müssen aus der Parameter-Map alle verwertbaren Parameter ermittelt werden. Dabei kann es zu drei Fehlern kommen:

```
public static List getAllInfo (
  String module;
  String function,
  Map reqParaOutMap,
  Map resRsltInMap,
  Map resDataInMap,
  List reqDataOutList,
  Map header,
  TimeStamp id,
  MDSLogger logger )
  throws MDSServerFunctionModuleException
```

Code 5.6: Signatur der aufgerufenen Funktion

- Ein Parameter wurde übergeben, hat aber den falschen Datentyp.
- Ein Parameter wurde nicht übergeben, es existiert kein Default-Wert für den Parameter und der Parameter ist zur Berechnung notwendig.
- Ein unbekannter Parameter wurde übergeben.

In den ersten beiden Fällen muss die Berechnung abgebrochen werden und ein spezielles Fehler-Antwort-Dokument erzeugt werden. Die Notwendigkeit eines Parameters kann sich auch im Kontext anderer Parameter ergeben (z.B.: Wenn Parameter "x"vorliegt, muss auch Parameter "y"vorliegen). Es ist sinnvoll, auch bei Vorhandensein von unbekannten Parametern die Berechnung abzubrechen, da dies darauf hindeutet, dass der Aufrufende die Semantik der Funktion nicht verstanden hat.

Nach Abschluss dieses Schrittes liegt eine gültige Parameter-Menge vor. Entsprechend der Zusammensetzung dieser Menge und der Menge der gewünschten Rückgabewerte muss nun entschieden werden, welche Berechnungen durchgeführt werden müssen. Oft können an dieser Stelle Optimierungen durchgeführt werden[4]. Es ist nicht sinnvoll, zunächst alle möglichen Rückgabewerte zu erzeugen und dann nur die gewünschten Werte zu übertragen. Vielmehr sollten die nicht gewünschten Werte erst gar nicht berechnet werden.

Zur Übermittlung von Fehlermeldungen auf Modul- und Funktionsebene werden auf der Client-Seite der Antwort-Struktur (`resRsltInMap`) zwei weitere Felder automatisch hinzugefügt:

`SDS:STATUSCODE` liefert einen frei definierbaren Fehlercode, wobei einige Fehlercodes schon vordefiniert sind (siehe Tab. D.1). Die dazu gehörende Fehlermeldung wird in `SDS:STATUS` erwartet. Auf der Server-Seite ist das Modul gehalten, beide Felder in

[4] Haffner u. a. (2000b); Haffner (2001)

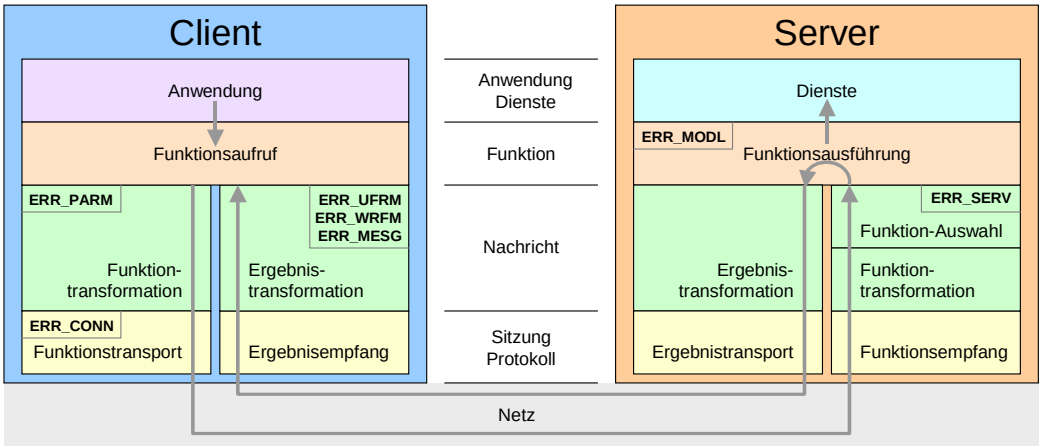

Abbildung 5.5: Fehlerklassen und der Ort ihrer Erzeugung

der Antwort-Struktur zu belegen. Im Fehlerfall wird im Allgemeinen die ursprünglich gewünschte Antwort-Struktur nicht erzeugt (da dies meist nicht mehr möglich ist), sondern ausschließlich die Struktur mit Fehlercode und Fehlermeldung.

5.4 Fehler-Klassen

Auf der Client-Seite können mit der Verwendung der Klasse `SDSBasicConnection` und der Methode `remoteProcedureCall` mehrere unterschiedliche Fehler in Form eines eigenen Exception-Objekts (`SDSConnectionException`) ausgelöst werden. Sie sind in Tabelle D.2 aufgeführt. Abbildung 5.5 zeigt, an welcher Stelle in der Client-Server-Kommunikation der Ursprung des Fehlers liegt. Durch die Kapselung der Fehler durch eine eigenständige Exception-Klasse kann von diesem Ursprung abstrahiert werden. Beispielsweise werden alle Fehler, die im Header von IPTP übermittelt werden (siehe Tab. B.2 auf Seite 145), durch die Fehlerklasse `ERR_CONN` abgebildet. Der ursprüngliche Fehlercode und die Fehlermeldung des Servers sind weiterhin in diesem Exception-Objekt abrufbar.

Die Exception-Klasse `SDSConnectionException` bietet neben ihrer eigenen Fehlerklasse auch den Ursprungsfehlercode und die Ursprungsfehlermeldung. In Abhängigkeit von der eigenen Fehlerklasse handelt es sich dabei entweder um die Fehlercodes auf IPTP-Ebene oder um die Fehlercodes auf Modul-Ebene.

```
public ClientStreamInOut remoteProcedureCall (
   String module,
   String function,
   Map reqParaOutMap,
   Map resRsltInMap,
   Map resDataInMap )
   throws SDSConnectionException
```

Code 5.7: Methode `remoteProcedureCall` (Streaming Version)

5.5 Client-seitiges Streaming von Daten

Das bisher beschriebe Verfahren zur Ausführung einer entfernten Funktion macht auf der Client-Seite bisher nur beschränkten Gebrauch der Streaming-Fähigkeit. So werden die Streaming-Daten jeweils komplett an die `remoteProcedureCall`-Methode übergeben und auch wieder empfangen. Eine Version, die das Streaming der Daten besser unterstützt, ist in Code 5.7 beschrieben. Neu gegenüber der bisher beschriebenen Variante ist das Fehlen von `reqDataOutList` für die Übertragung der Streaming-Daten zum Server und das `ClientStreamInOut`-Objekt als neuer Rückgabewert.

In der `remoteProcedureCall`-Methode aus Code 5.5 auf Seite 66 wird der aufrufende Prozess so lange blockiert, bis das Ergebnis der Anfrage vorliegt oder ein Fehler aufgetreten ist. In der Version aus Code 5.7 erhält der Prozess sofort die Kontrolle. Der weitere Aufruf-Vorgang wird über das `ClientStreamInOut`-Objekt gesteuert. Die folgenden Methoden werden zur Verfügung gestellt:

`void sendStreamData(Object o);` sendet ein Objekt in den ausgehenden Datenstrom.

`boolean hasNextStreamData();` teilt mit, ob ein Objekt im eingehenden Datenstrom vorliegt.

`Object readStreamData();` liest das nächste Objekt des eingehenden Datenstroms. Dieser Aufruf blockiert so lange, bis ein Objekt im Datenstrom vorliegt.

`void finishSendStream();` beendet den ausgehenden Datenstrom.

`boolean inStreamHasFinished();` teilt mit, ob der eingehende Datenstrom beendet ist.

`boolean responseFinished();` teilt mit, ob die Antwort vollständig empfangen wurde. Per Seiteneffekt ist dann auch `resRsltInMap` mit gültigen Rückgabewerten besetzt.

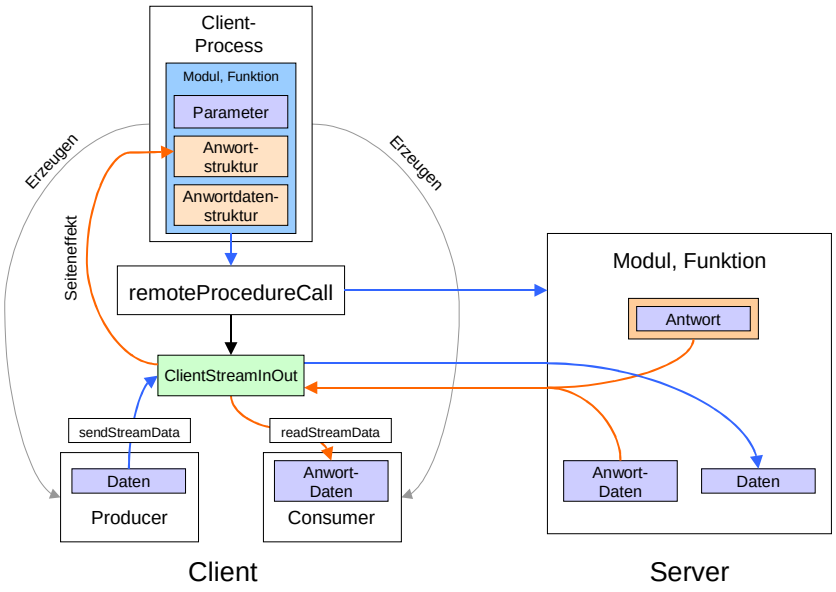

Abbildung 5.6: Entfernter Funktionsaufruf mit Streaming

`boolean rpcFinished();` teilt mit, dass der gesamte RPC (Anfrage und Antwort) abgeschlossen ist, also nach `finishSendStream` und mit `resposeFinished`.

`void waitUntilRpcFinished();` blockiert solange, bis der RPC abgeschlossen ist.

Eine mögliche Vorgehensweise bei der Ausnutzung dieser Streaming-Fähigkeiten ist im Anhang D in Code D.2 auf Seite 153 beschrieben und in Abb. 5.6 dargestellt:

Nach Ausführung von `remoteProcedureCall` werden zwei Prozesse gestartet. Ein Producer-Process, der die Daten für den Ausgabedatenstrom erzeugt, und ein Consumer-Prozess zur Auswertung des Eingabedatenstroms. Der Master-Prozess wartet nach dem Starten dieser Prozesse anschließend, bis der RPC-Vorgang abgeschlossen wurde. Dann ist auch über den Seiteneffekt wieder `resRsltInMap` als Ergebnis der Anfrage belegt.

Das Streaming der Daten auf der Server-Seite wird in Abschnitt 8.3 auf Seite 104 beschrieben, nachdem der Aufbau des Servers näher erläutert wurde.

6 Serverseitige Anfragebehandlung

Die logische Einfachheit ist der einzige Weg,
auf dem wir zu tiefen Erkenntnissen geführt werden.

ALBERT EINSTEIN

Im Kapitel 5 wurde der Aufruf einer entfernten Funktion auf Protokoll- und Nachrichten-Ebene sowie aus Sicht des Anwendungsentwicklers betrachtet. Diese Sicht gibt keinerlei Aufschluss über den Aufbau des Servers und seine internen Prozesse. Dieses Kapitel beschreibt nun die Komponenten, die zwischen Entgegennahme der Anfrage und Ausführung der Funktion involviert sind.

In einer ersten vereinfachten Darstellung ist der Smart Data Server in drei logische Schichten unterteilt, denen unterschiedliche Aufgaben zugewiesen sind (siehe Abb. 6.1):

- Die **Sitzungsschicht** (*Session Layer*) beinhaltet Module, die ausschließlich zur Abarbeitung von Sitzungen (Socket-Sitzung, Benutzer-Sitzung, zeitgesteuerte Sitzung) bestimmt sind.
- Die **Funktionsschicht** (*Function Layer*) repräsentiert die Funktionalität/Dienste des Servers in (Funktions-)Modulen gegenüber der Client-Anwendung. Die Module dieser Schicht sind kein integraler Bestandteil des Servers, da diese nur im Kontext einer Client-Anwendung bei Bedarf dem Server hinzugefügt werden.
- Beide Schichten greifen auf Module zu, die unter der **Diensteschicht** (*Service Layer*) subsumiert werden. Sie repräsentieren serverinterne Dienste und Funktionalitäten als Middleware-Infrastruktur. Da auf diese Dienste von der Funktions- und Sitzungsschicht zugegriffen wird, ist sie zwischen den andern beiden Schichten angesiedelt.

Diese Einteilung wird in diesem sowie dem nächsten Kapitel genauer erläutert werden, wobei immer nur der in dem aktuellen Kontext wichtige Ausschnitt betrachtet wird.

In Abschnitt 6.1 wird zunächst beschrieben, wie der Server eine Benutzersitzung, also den Anfrage-/Antwort-Vorgang, behandelt. Die Erlaubnis zur Ausführung einer entfernten Funktion kann durch eine Autorisierung überprüft werden. Abschnitt 6.2 erläutert diesen Vorgang. In Abschnitt 6.3 wird ein wichtiger Aspekt bei der Ausführung einer entfernten Funktion beschrieben, die Lokalisierung der Funktion und des Moduls, in dem diese Funktion definiert ist. Neben der Ausführung von Funktionen über eine Anforderung eines Clients kann diese auch über ein Zeit-Ereignis initiiert werden. Abschnitt 6.4 zeigt die Unterschiede beider Varianten auf.

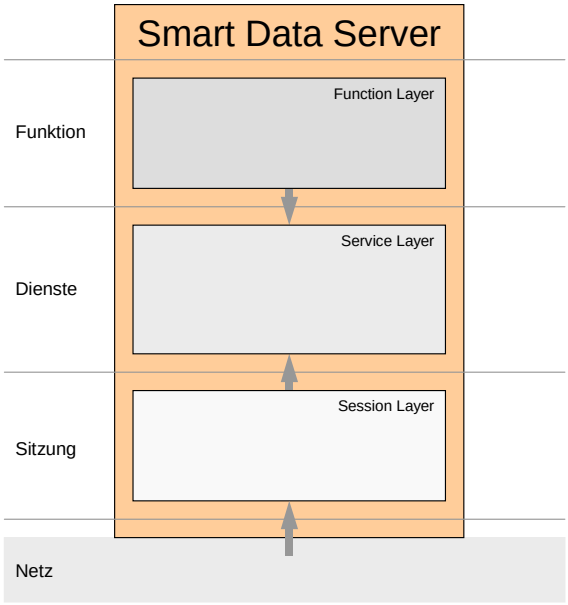

Abbildung 6.1: Schichteneinteilung des Smart Data Servers

6.1 Sitzungsabwicklung

Die Kommunikation zwischen Client und Server erfolgt innerhalb von Sitzungen (Sessions), die den Zeitraum der RPC-Anfrage und RPC-Antwort beschreiben. Auf der Server-Seite werden zwei Arten von Sitzungen unterschieden: *Socket-Sitzungen* und *Benutzer-Sitzungen*.

Auf der Basis von TCP/IP wird zwischen Client und Server eine Socket-Sitzung aufgebaut, die einen bidirektionalen Datenstrom zwischen Client und Server definiert. Diese Verbindung kann für eine oder mehrere RPC-Anfragen aufrecht erhalten werden. Das Kommunikationsprotokoll zwischen Client und Server bietet hierzu die Möglichkeit anzuzeigen, ob die Verbindung nach Abschluss des RPCs weiterhin bestehen bleiben soll.

Üblicherweise sind Funktionsanfragen für sich genommen unabhängig. Bei der Ausführung hat die Vorgeschichte der Anfragen, also welche Anfragen in der Vergangenheit in welcher Reihenfolge ausgeführt wurden, keine Bedeutung. Eine Benutzer-Sitzung stellt nun mehrere Anfragen in einen gemeinsamen Kontext, den eines identifizierten Benutzers (Clients). Eine aufgerufene Funktion kann diesen Kontext bei der Ergebnis-Erstellung berücksichtigen. Hierzu wird zu Beginn der Benutzer-Sitzung eine eindeutige Sitzungs-ID (Session-ID) definiert. Zukünftige Anfragen mit der gleichen Sitzungs-ID gehören zur selben Benutzer-Sitzung.

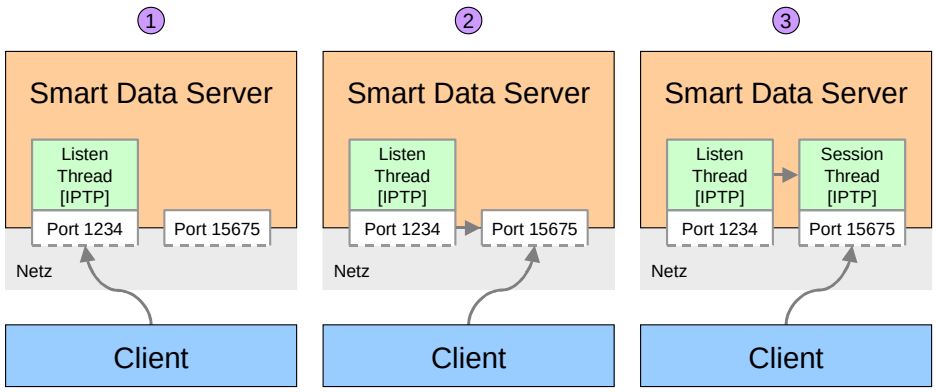

Abbildung 6.2: Delegieren des Ports zum Session-Modul

Zur Entgegennahme von Benutzeranfragen existiert auf der Server-Seite das *Listen*-Modul. Es ist möglich, verschiedene Instanzen dieses Moduls beim Start des Servers zu erzeugen. Jede Instanz ist als eigenständiger Thread jeweils für einen eigenen Netzwerk-Port verantwortlich, der mit einem Kommunikationsprotokoll assoziiert. Zur Zeit kann nur das IPTP an einen Port gebunden werden, die Flexibilität ist jedoch gegeben, in zukünftigen Versionen andere Protokolle festzulegen.

Alleinige Aufgabe des Listen-Moduls ist es, auf Socket-Verbindungen an dem definierten Port zu warten (siehe Abb. 6.2: 1). Im Falle des Verbindungsaufbaus wird der Eingangs-Port auf TCP-Ebene auf einen freien Port umgeleitet, damit der ursprüngliche Port für weitere Anfragen wieder frei wird (2). Die Socket-Informationen der Verbindung und das assoziierte Protokoll wird anschließend an eine neue Instanz des *Session*-Moduls übergeben, das die weitere Abarbeitung der Anfrage übernimmt (3).

Das Session-Modul erfüllt mehrere Aufgaben (siehe Abb. 6.3). Zum einen übernimmt es die Kommunikation mit dem Client unter Verwendung des festgelegten Protokolls. Dies betrifft auf der einen Seite die Entgegennahme der Anfrage sowie das Übertragen des Ergebnisses. Jede Kommunikation mit dem Client wird mit einer eindeutigen Session-ID assoziiert, die der Client in weiteren Anfragen zur Identifikation einer User-Session verwenden kann.

Eine weitere Aufgabe betrifft die Überprüfung der Authentizität der Anfrage. Da im Sitzungsprotokoll die Angabe des angesprochenen Moduls aufgeführt ist, muss die Nachricht zur Überprüfung noch nicht dekodiert werden. Dieser Aspekt ist wichtig, falls die Nachricht verschlüsselt ist und die Anfrage nicht vom Server selbst beantwortet wird, sondern an einen weiteren SDS weitergeleitet wird.

Da das Session-Modul selbst keine Informationen über die Gültigkeit der Anfrage verwaltet, wird ein weiteres Modul, das *Auth*-Modul, angesprochen. Dieses führt die Überprü-

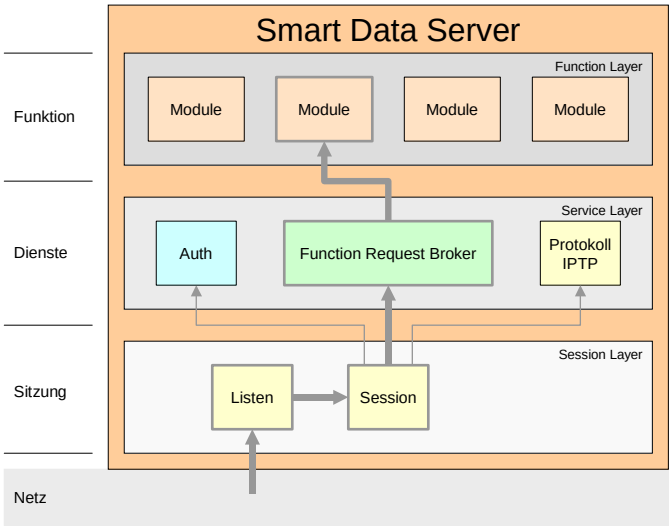

Abbildung 6.3: Sitzungsbearbeitung

fung auf Grundlage des angesprochenen Moduls, der Session-ID und der Socket-Informationen[1] durch (siehe Abs. 6.2). Damit ist eine Abstufung der Rechte für verschiedene Ports möglich. Im Falle der Ungültigkeit der Anfrage sorgt das Session-Modul für die Übertragung einer IPTP-Fehlernachricht an den Client.

Ist die Anfrage erlaubt, so wird die Anfrage an den *Function Request Broker* übergeben, der für das Auffinden des angesprochenen Moduls und die Ausführung der gewünschten Funktion verantwortlich ist. Das Session-Modul überträgt das Ergebnis der Funktionsausführung mittels IPTP an den Client.

Eine letzte Aufgabe des Session-Moduls betrifft die Überprüfung der Gültigkeit von Socket- und Benutzer-Sitzungen. Sind beide über einen längeren (definierbaren) Zeitraum inaktiv, so wird im Falle der Benutzer-Sitzung diese aus dem Kreis der gültigen entfernt, im Falle der Socket-Sitzung diese beendet. Ziel dieser Maßnahme ist ein sinnvolles Ressourcen-Management.

Nicht weiter verwendete Benutzer-Sessions deuten darauf hin, dass die Client-Anwendung beendet ist. Für nicht aktive Socket-Sessions kann es Gründe auf der Client- oder auf der Server-Seite geben. Eine Möglichkeit ist, dass der Client in einen inkonsistenten Zustand geraten ist und nicht mehr in der Lage ist, die Anfrage weiter zu generieren oder deren Ergebnis entgegen zu nehmen. Eine andere Möglichkeit ist, dass das Modul mit seiner Funktion nicht mehr zur Erzeugung der Antwort in der Lage ist.

[1] Eigener Port, fremde IP-Nummer

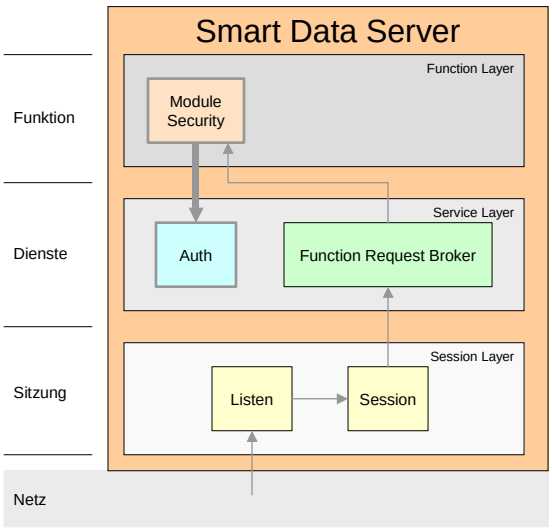

Abbildung 6.4: Autorisation/Authentifikation

Der Schwellwert, wann eine Socket- oder Benutzer-Sitzung abgebrochen wird, sollte so gewählt sein, dass Funktionen, deren Ergebnis-Berechnung längere Zeit in Anspruch nehmen, nicht vor deren Beendigung abgebrochen werden.

6.2 Autorisation/Authentifikation

Das Auth-Modul dient zur Überprüfung, ob eine Anfrage ausgeführt werden darf oder ob die Berechtigung hierfür fehlt. Grundlage hierfür ist die Session-ID, die Port-Nummer, an dem die Anfrage entgegen genommen wurde sowie die IP-Nummer des anfragenden Clients.

Zunächst wird überprüft, ob schon eine Autorisierung mit der gegebenen Session-ID stattgefunden hat. In diesem Fall muss die IP-Nummer des Clients mit der IP-Nummer des Clients zum Zeitpunkt der Autorisierung übereinstimmen. Die IP-Nummer des Clients spielt bei Modulen mit beschränkter Verfügbarkeit eine weitere Rolle. Module können mit IP-Listen assoziiert werden, wobei nur Clients aus dem IP-Nummernkreis der Liste die Erlaubnis bekommen, Funktionen des Moduls anzusprechen. Dieses Verfahren stellt zunächst nur eine erste Sicherheitsbarriere dar. Dem Autor ist bekannt, dass mittels IP-Spoofing eine Möglichkeit existiert, sich unter einer fremden IP-Nummer auszugeben.

Eine Möglichkeit zur Klassifizierung von Anwendergruppen ist die Beschränkung der ansprechbaren Modulen in Abhängigkeit der Ports über die sie aufgerufen werden. Ver-

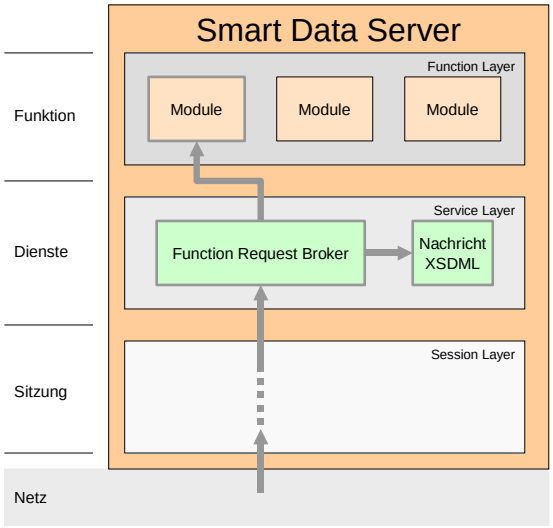

Abbildung 6.5: Modul-Aufruf

schiedene Anwendergruppen greifen auf unterschiedliche Ports des Servers zu und erhalten nur eine Auswahl an möglichen Funktionen.

Um eine Autorisation durchführen zu können, muss die Login-Funktion in einem speziellen *Security*-Modul mittels einer normalen IPTP-Anfrage angesprochen werden (siehe Abb. 6.4). Die Login-Funktion nutzt dann die Autorisierungsfunktion des Auth-Moduls. Nach erfolgreicher Autorisation wird die Session-ID in die Liste der autorisierten Sitzungen eingetragen. Die Autorisierungsfunktion nutzt die interne Datenbank-Schnittstelle *Datastore* (siehe Abs. 7.2), um die Gültigkeit der Login-Information in einer Datenbank abzurufen.

Zur Zeit findet die Authentifizierung über Name und Passwort statt. Diese können sicher mittels asymmetrischer Schlüssel der IPTP-Nachricht zwischen Client und Server übertragen werden.

6.3 Auffinden von Modul und Funktion

Eine zentrale Rolle im Smart Data Server spielt der *Function Request Broker*. Er stellt die Verbindung zwischen der Anfrage und dem gewünschten Modul mit seiner Funktion (siehe Abb. 6.5). Zwei Arten von Modulen können im Server definiert werden[2]:

Interne Module sind solche Module, deren Code im Server vorliegt.

Externe Module müssen über einen weiteren Smart Data Server angesprochen werden.

[2] Roth u. a. (1999a, 2000)

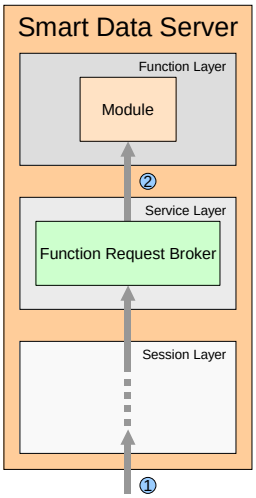

Abbildung 6.6: Internes Modul

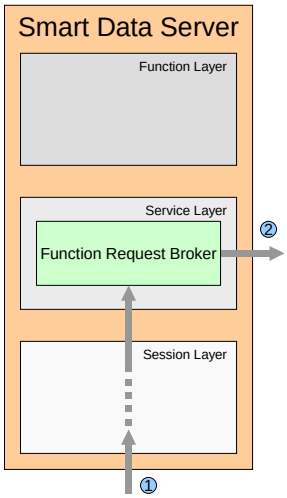

Abbildung 6.7: Externes Modul

6.3.1 Interne und externe Module

Bei den internen Modulen (siehe Abb. 6.6) wird die Modul-Bezeichnung mit einer Java-Klasse assoziiert. Diese Verknüpfung wird in der Konfiguration des Servers festgelegt. Bei Start des Servers werden alle Module geladen und initialisiert. Dies ist möglich, da alle Module ein einheitliches (SDS-Modul-)Interface implementieren müssen. Dieser Vorgang ist vergleichbar mit dem Einbinden eines Datenbank-Treibers in ein System.

Jedes Modul ist für eine Menge von Funktionen verantwortlich. Dem Modul wird mit der Anfrage die IPTP-Nachricht übergeben. Das Modul ist nun grundsätzlich selbst dafür verantwortlich, die Nachricht mittels des *XSDML*-Moduls zu dekodieren und die eigentliche Funktion, wie in Abschnitt 5.3 auf Seite 65 beschrieben, aufzurufen. Da dieser Vorgang für die meisten Module gleich ist, stellt der Function Request Broker eine Möglichkeit zur Verfügung, dies mittels der Reflection-API zu übernehmen. Die Flexibilität eines Moduls, die IPTP-Nachricht in einer beliebigen Art zu interpretieren, bleibt von dieser Option jedoch unberührt.

Neben internen Modulen besteht die Möglichkeit, ein Modul als extern zu definieren (siehe Abb. 6.7). In diesem Fall wird das Modul nicht mit einer Java-Klasse assoziiert, sondern mit einem weiteren Smart Data Server. Diese Verknüpfung ist statisch, d.h. die IP-Adresse und Port-Nummer des zweiten SDS müssen beim Start des Servers bekannt sein.

Bei Aufruf eines externen Moduls wird die IPTP-Nachricht unverändert und, ohne zuvor dekodiert worden zu sein, an den zweiten SDS weiter geleitet. In gleicher Weise wird bei der Erstellung des Ergebnisses die IPTP-Nachricht des zweiten SDS durch den ursprünglichen SDS ohne Bearbeitung an den Client geleitet. Der Client kann nur am IPTP-Header

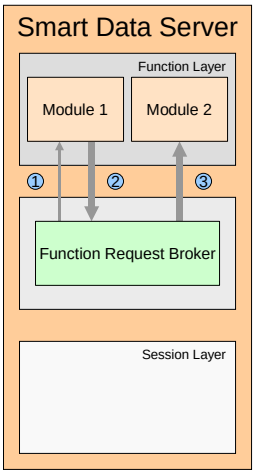

Abbildung 6.8: Intern-interner Aufruf

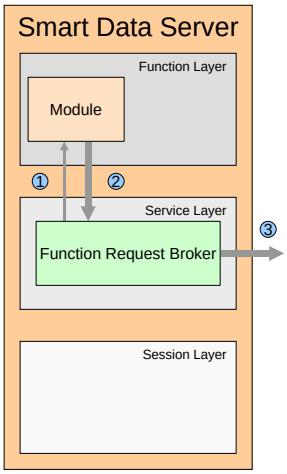

Abbildung 6.9: Intern-externer Aufruf

erkennen, dass die Anfrage nicht durch den angesprochenen SDS, sondern durch einen oder mehrere weitere SDS geleitet wurde. Das SERVER-Element des Headers ist eine Liste der involvierten Server.

6.3.2 Intern-interne und intern-externe Aufrufe

Jedes Modul darf bei der Kenntnis von anderen Modulen, Funktionen dieser Module über den Function Request Broker intern aufrufen. Dabei muss die Anfrage entsprechend dem IPTP-Protokoll aufgebaut werden. Im Allgemeinen reicht ein verkürzter Header mit der Angabe des aufzurufenden Moduls und der Kodierung der Nachricht aus.

Beim Aufruf sind zwei Fälle zu unterscheiden: intern-interne und intern-externe Aufrufe (siehe Abb. 6.8 und Abb. 6.9). Im ersten Fall ist das aufgerufene Modul als intern definiert, im zweiten Fall als extern. Ein Modul wird auch extern angesprochen, wenn das Modul in der Konfiguration des Servers gleichzeitig intern und extern definiert ist[3] und das interne Modul sich quasi selbst aufruft. Ansonsten hat der interne Aufruf stets Priorität vor dem externen Aufruf.

Das Verhalten des Systems, bei Selbst-Aufrufen nicht das interne Modul, sondern das externe anzusprechen, kann folgendermaßen begründet werden: Es ist nicht davon auszugehen, dass der Selbst-Aufruf zur Durchführung einer Rekursion initiiert wurde. Normale Rekursionen sind einfacher durch den direkten Methodenaufruf möglich. Es muss also davon

[3] In der Konfiguration verweist eine Definition auf die Java-Klasse, eine zweite Definition verweist auf den zweiten SDS.

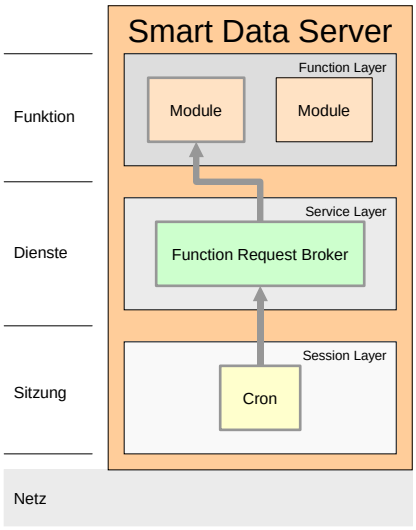

Abbildung 6.10: Zeit-gesteuerte Aufrufe

ausgegangen werden, dass der Entwickler diesen Weg nur deshalb wählt, um die Funktion in einem Modul eines zweiten Servers anzusprechen. Ein Grund hierfür kann sein, dass der Funktion dort eine andere Datenbasis vorliegt.

6.4 Zeitgesteuerte Aufrufe

Neben der Möglichkeit, Funktionen durch externe RPC-Anfragen auszulösen, bietet der Server auch zeitgesteuerte Ausführungen an. Hierfür ist das *Cron*-Modul verantwortlich (siehe Abb. 6.10). Die Funktionen werden in diesem Falle bei Auftreten eines Zeit-Ereignisses ausgeführt.

Der Server unterstützt zwei Arten von Ereignissen: Ereignisse, die durch Erreichen eines Zeitpunktes als *At-Jobs* ausgelöst werden sowie Ereignisse, die nach Ablauf eines Zeitraums (Intervall) als *Every-Jobs* ausgelöst werden.

At-Jobs definieren einen Zeitpunkt durch Angabe von Minute, Stunde, Tag, Monat und Wochentag. Jede dieser Angaben kann durch eine Wildcard (*) ersetzt werden, womit diese Information zur Bestimmung des Zeitpunktes nicht herangezogen wird. Zum Beispiel definiert die Angabe `Minute=0, Stunde=*, Tag=1, Monat=*, Wochentag=*`, jede volle Stunde am jeweils Ersten eines Monats.

Every-Jobs definieren einen Zeitraum durch die Angabe von Minuten. Das Ereignis wird somit nach Ablauf der angegebenen Anzahl von Minuten ausgelöst, bezogen auf den Zeit-

punkt des zuvor ausgelösten Ereignisses.

Da das Cron-Modul nur periodisch das Eintreten eines Zeit-Ereignisses überprüft, wird der darauf folgende Überprüfungszeitpunkt zum Auslösen des Ereignisses gewählt.

Unabhängig von der Art des Jobs startet das Cron-Modul mit Erreichen des Zeit-Ereignisses die gewünschte Funktionalität eines Moduls. Hierzu wird der Function Request Broker herangezogen, der das Modul als Thread lädt und startet[4]. Die SDS-Modul-Schnittstelle, die bei einer normalen Ausführung des Moduls herangezogen wird, bleibt unangetastet. Dem Modul kann neben der Angabe der gewünschten Funktion ein Satz von Parametern mitgegeben werden, der sich aus der Server-Konfiguration ergibt.

Die zeitgesteuerten Funktionen unterscheiden sich erheblich von den Funktionen, die durch eine Client-Anfrage an den Server gestellt werden. In den zeitgesteuerten Anfragen ist das IPTP-Protokoll nicht involviert, ebenso müssen demzufolge keine Nachrichten dekodiert werden. Eine Antwort wird durch die Funktion nicht erzeugt, vielmehr ist der Seiteneffekt der Funktion von Bedeutung, der z.B. durch Änderungen in einer Datenbank oder durch Versenden von eMails entstehen kann.

Da die Module als Thread gestartet werden, ist die Entwicklung derartiger Module mit Sorgfalt zu gestalten. Java hat alle Funktionen, die einen Thread extern beenden lassen, als *Deprecated* quasi verboten. Somit muss die Funktion in einem Modul selber für ihr Ende sorgen. Ein anderes Problem ergibt sich aus der Laufzeit der Funktion. Führt ein neues Zeit-Ereignis zur Erzeugung eines weiteren Threads, noch bevor der alte Thread fertig ist, muss das Modul selbst dafür Sorge tragen, dass sich beide Threads nicht gegenseitig stören.

Natürlich ist es möglich, auch Funktionen des Servers über die Cron-Funktion des Betriebssystems auszulösen. Eine solche Vorgehensweise würde allerdings die Entwicklung von Client-Anwendungen erfordern, die bei Eintreten des Cron-Ereignisses gestartet werden, um per RPC-Anfrage die entsprechende Funktion im Server auszuführen. Eine solche Anwendung müsste auf die Antwort des Servers warten, die möglicherweise aber irrelevant ist[5]. Dies alles ist bei serverinternen Cron-Jobs nicht notwendig.

[4] Das Modul muss somit das `Runnable`-Interface implementieren.

[5] Beispielsweise bei Anfragen, bei denen nur der Seiteneffekt der Anfrage wichtig ist.

7 Server Infrastruktur

Durch Dienen zu herrschen ist das Geheimnis des Erfolgs.
Wahres Herrschen ist Dienen.

I GING
CINESISCHES WEISHEITSBUCH

Eine Middleware-Architektur zeichnet sich nicht alleine dadurch aus, RPC-Anfragen abarbeiten zu können. Als Mittler zwischen Front-End und Back-End, also zwischen Anwendungen und der vorhandenen Infrastruktur (Datenbanken, eMail-Systemen, o.ä.), bietet die Middleware eine Plattform, die den Entwickler von Funktionen und Modulen bei dem Zugriff auf das Back-End unterstützt.

Java liefert hierfür zwar eine Reihe von APIs, mit dem diese Zugriffe standardisiert durchgeführt werden können, der Anspruch einer Plattform geht jedoch weiter. Der Zugriff auf das Back-End soll für den Entwickler transparent sein. Details sollen soweit wie möglich verborgen bleiben, um einen hohen Grad an Flexibilität bezüglich möglicher Änderungen im Back-End zu erhalten. Im Gegensatz zur direkten Nutzung der Java-APIs (die nicht verhindert werden kann), betreffen solche Änderungen dann nur noch die Konfiguration des Servers, nicht jedoch die Komponente, die auf das Back-End zu greift.

Im Smart Data Server wurde dies durch Einführung einer Serviceschicht erreicht, deren Umfang sich auf die am häufigsten gewünschten Dienste beschränkt. Abbildung 7.1 zeigt nun detaillierter, wie sich die Serviceschicht in die Grundstruktur des Smart Data Servers einfügt[1].

Dieses Kapitel beschäftigt sich mit den Diensten der Serviceschicht. Einige Module dieser Schicht wurden schon im Kapitel 6 eingeführt: Das Autorisierungsmodul *Auth* und der *Function Request Broker* sowie die Module, die das Protokoll IPTP und die übertragene XSDML-Nachricht abbilden. Allein mit diesen Modulen ist ein Betrieb des Servers möglich. Zusätzliche Module, die den Zugriff auf die Back-End-Infrastruktur bieten, sind das Modul für Datenbankzugriffe (siehe Abs. 7.2) und eMail-Dienste (siehe Abs. 7.3). Ein Dienst, dessen Nutzen nicht unterschätzt werden darf, ist der Logging-Dienst (siehe Abs. 7.1). Dieser Dienst ist unerlässlich für den Betrieb des Servers, da er Meldungen aller aktiven Komponenten des Servers protokolliert und damit den inneren Zustand sichtbar macht.

Der Begriff „Dienst“ hat im Kontext der Web-Services eine andere Bedeutung als in der SDS-Serverstruktur. Ein Web-Service-Dienst ist vergleichbar mit den Funktionen, die ein SDS-(Funktions-)Modul zur Verfügung stellt.

[1] Roth (2000b); Roth u. a. (1998, 1999a)

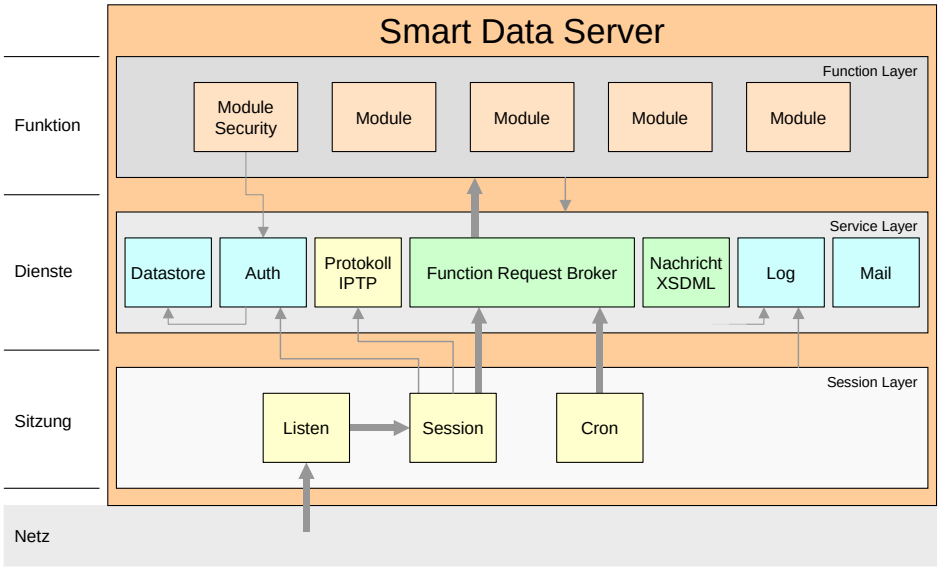

Abbildung 7.1: Aufbau des SDS

7.1 Logging

Ein wesentlicher Dienst des Servers ist der Log-Dienst. Er dient jeder Art von Modul im Server zur Protokollierung von Informationen, Status-Meldungen, Fehlermeldungen oder Warnungen im Server-Log oder weiterer frei definierbarer Log-Dateien.

Wenn ein Modul den Log-Dienst nutzen will, instanziiert dieses ein Log-Objekt unter Verwendung einer Kennung, die dazu dient, Einträge der Log-Datei einem Modul zuordnen zu können. Wird zusätzlich ein optionaler Ausgabestrom angegeben, so werden die Log-Einträge nicht in die Standard-Log-Datei geschrieben, sondern in diesen Ausgabestrom. An Stelle des Ausgabestroms ist auch eine Kennung erlaubt, die in der Konfiguration des Servers mit einer Log-Datei assoziiert wird.

Jede Nachricht, die geloggt wird, muss mit einem Log-Level angegeben werden[2]. Tabelle E.1 im Anhang E führt die möglichen Level auf. Die Level 0-3 erfordern im Allgemeinen keine Aufmerksamkeit und sind nur informativ, die Level 4 und 5 sind Fehler, die Aufmerksamkeit erfordern, aber nicht die Integrität des Servers stören und die Level 6 und 7 erfordern sofortiges Handeln, z.B. Neustart des Servers. Dies kann jedoch nicht verallgemeinert werden und hängt von der Art des Fehlers und dem Modul ab, das den Fehler meldet.

[2]Wird dieser nicht angegeben, wird der Standard-Level implizit verwendet.

In der Server-Konfiguration ist die Angabe eines Schwellwertes möglich. Nachrichten mit einem niedrigeren Level werden nicht protokolliert. Damit kann die Größe der Log-Datei in Abhängigkeit des Informationsbedürfnisses beeinflusst werden.

Eine Zeile in der Log-Datei führt folgende Informationen auf:

- Den Log-Level.
- Den Zeitpunkt, zu dem die Nachricht geloggt wurde, mit Angabe von Datum/Uhrzeit und Zeitzone.
- Die Kennung, die bei Instanziierung des Logging-Objektes verwendet wurde. Im Allgemeinen handelt es sich hierbei um die Angabe des Moduls, ggf. ergänzt durch die Angabe der aktuellen Funktion.
- Die Nachricht.

Der Logging-Mechanismus unterscheidet sich durch seine einfache und auf das Wesentliche reduzierte Art bzgl. Instanziierung und Aufruf von der Logging-API von Java, die erst mit der Version 1.4 eingeführt wurde, oder von *Log4J* des Apache Jakarta Projekts[3].

7.2 Datenbank-Zugriff

Generell steht jeder Java-Anwendung der Zugriff auf Datenbanken mit der JDBC-API zur Verfügung. Der SDS besitzt jedoch seine eigene Datenbank-Schnittstelle. Hierfür waren mehrere Gründe ausschlaggebend, die zur Entwicklung des *Datastore*-Modul führten.

Der Zugriff auf eine oder mehrere Datenbanken erfolgt nicht über Datenbanktreiber und SQL-Anfragen, sondern über ein Abfrage-Objekt. Dieses muss vor der Ausführung der Anfrage zunächst konfiguriert werden, um festzulegen, welche Daten gelesen oder geändert werden sollen. Diese Konfiguration erfolgt ohne Kenntnis von SQL und ermöglicht Abfragen mit einer größtmöglichen Unabhängigkeit von datenbankspezifischen Dialekten. Die Mächtigkeit dieser Konfiguration ist geringer gegenüber SQL-Anfragen, die auf spezielle Datenbank-Systeme hin optimiert sind. Allerdings garantiert dies die Portabilität beim Wechsel des Datenbank-Systems.

Wird das Anfrage-Objekt an das Datastore-Modul geleitet, so ermittelt dieses aus den gegebenen Informationen, um welche Datenbank es sich konkret handelt: Verbindungsinformationen, wie Benutzerkennung und Passwort sowie zu verwendender Treiber ergeben sich aus der Konfiguration des Servers. Der Anwendungsentwickler hat hierüber keinen Zugriff.

[3] `http://jakarta.apache.org/log4j`

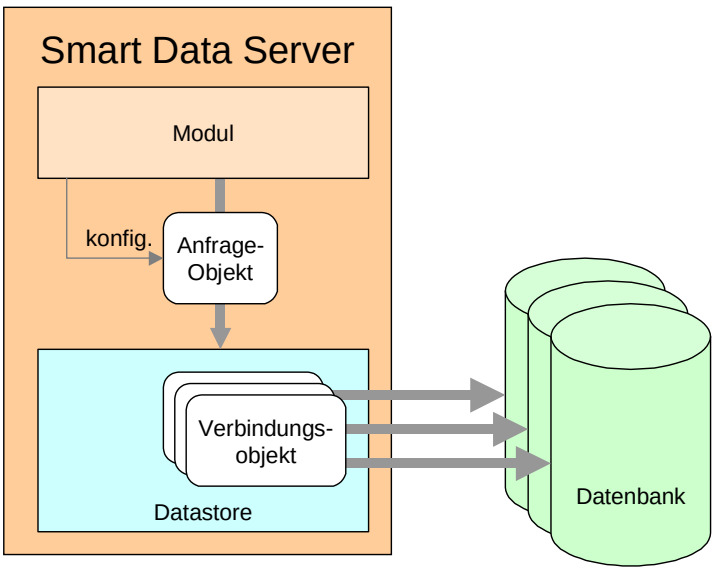

Abbildung 7.2: Zugriff auf Datenbanken über Datastore

Jede Instanz, die Datenbank-Zugriff über das Datastore-Modul erhalten will, muss ein Datastore-Objekt erzeugen. Dieses Objekt verwaltet die Verbindungsobjekte zur Datenbank und sorgt dafür, dass zum Zeitpunkt der Anfrage ein solches verfügbar und auch benutzbar ist (siehe Abb. 7.2). Im Zweifelsfall wird versucht, eine Verbindung erneut aufzubauen, falls eine vermeintlich existierende Verbindung nicht mehr verwendbar ist, z.B. weil inzwischen der Datenbank-Server neu gestartet wurde. Damit ist garantiert, dass ausschließlich schwerwiegende, nicht bearbeitbare Ausnahmesituationen an die aufrufende Instanz weiter geleitet werden.

Jeweils genau ein Thread darf gleichzeitig auf ein Datastore-Objekt zugreifen, da es sonst zu inkonsistenten Ergebnissen kommen kann.

Das Datastore-Modul wurde mit einer praxisgerechten Basisfunktionalität ausgestattet. Mittels einer neuen Klassenbibliothek[4] wird der transparente Zugriff auf Datenbanken in Zukunft noch flexibler möglich sein. Diese Bibliothek überführt beliebige SQL-Anfragen unterschiedlichster Dialekte in herstellerabhängige SQL-Anfragen der gewünschten Zielsysteme, ohne dass der Autor der SQL-Anfragen das Zielsystem kennen muss. Zusätzlich wurde die Anfragesyntax erweitert, um Anfrage über Datenbank- und Herstellergrenzen hinweg durchführen zu können. Die Einbindung des Smart Data Servers in heterogen Datenbanklandschaften wird somit noch einfacher möglich sein.

[4] Mitev (2003)

7.3 eMail

Zum Versenden und Empfangen von eMail existiert in Java die JavaMail-API. Diese ist jedoch nicht Bestandteil des Java Runtime Environment, sondern Teil der Java-Extensions. Das *Mail*-Modul des SDS nutzt diese API. Im Gegensatz zum Datastore-Modul werden jedoch die wesentlichen Merkmale der API nicht verändert. Alleiniger Sinn des Moduls ist das Verbergen der Zugriffsinformationen vor der Anwendung, also der Account- und Passwort-Informationen sowie der Internet-Adressen der angesprochenen SMTP- und POP-Server.

8 Workflow-Programme

Alles fließt.
Panta rhei.

HERAKLID VON EPHESUS

Die bisher beschriebene Architektur einer Middleware liefert eine funktionsfähige und praktikable Plattform zur Entwicklung verteilter Anwendungen. Zunächst stand das einfache Einbinden, Entwickeln und Anwenden von Komponenten im Vordergrund. Server-interne Abläufe spielten nur insofern ein Rolle, um diesem Ziel gerecht zu werden.

Dieses Kapitel beschäftigt sich mit der Optimierung der serverinternen Abläufe und Prozesse. Gründe hierfür sind

- die Fähigkeit zur stetigen Erweiterung der Server-Funktionalitäten,
- Streaming von Daten,
- Aufweichung der Grenzen zwischen Server-Funktionalität und der Funktionen von Modulen.

Dem Konzept der erweiterten Server-Architektur liegt die Idee der so genannten *Workflow-Programme* zu Grunde, mit dem Daten- und Kontrollflüsse gesteuert werden können. Die Workflow-Programme-Technologie wurde speziell für den Smart Data Server entwickelt.

In Abschnitt 8.1 werden die Elemente dieser Programme erläutert. Abschnitt 8.2 beschäftigt sich mit dem Aufbau von Objekt-Pipelines während der Abarbeitung von Workflow-Programmen. Der Aspekt der Fehlerbehandlung spielt hierbei eine wichtige Rolle und wird in Abschnitt 8.2.3 behandelt. Aufbauend auf diesen Strukturen führt Abschnitt 8.3 schließlich in die Umsetzung dieses Konzeptes im Smart Data Server ein.

8.1 Grundstruktur

Der Aufbau von Servern besteht in der Regel aus Funktionseinheiten oder Komponenten, die bestimmte Aufgaben erfüllen sollen. Zwischen diesen Komponenten bestehen Abhängigkeiten, die das Verhältnis von aufrufenden und aufgerufenen Komponenten betreffen,

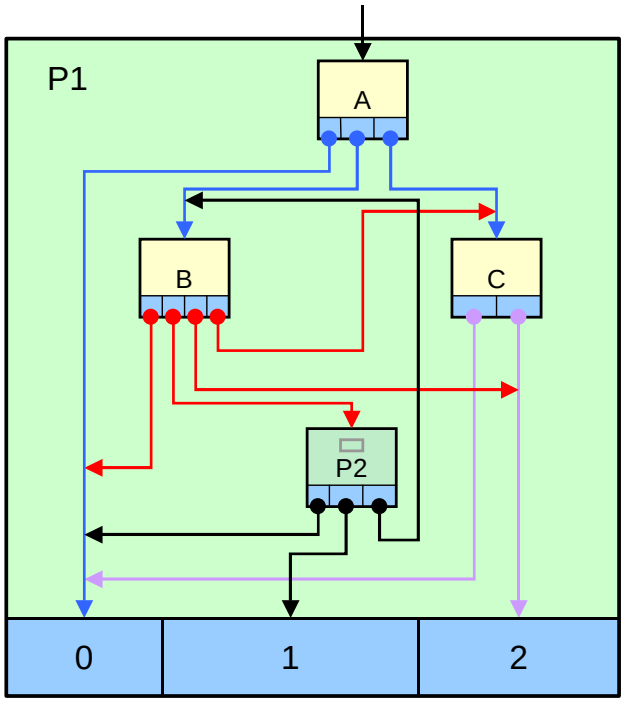

Abbildung 8.1: Beispiel eines Workflow-Programms

sowie der Daten, die erwartet oder geliefert werden müssen. Die Abfolge der Abarbeitung der Komponenten ist dabei mit möglichen alternativen Pfaden fest vorgegeben und ist durch direkte Aufrufe der jeweils anderen Komponente im Programm codiert.

Ein Problem ergibt sich dann, wenn die Server-Struktur später abgeändert oder erweitert werden soll, z.B um neue Features einzubauen. Mühsam müssen die Stellen gesucht werden, bei denen Aufrufe der neuen Komponente oder Abänderungen der Reihenfolge von Aufrufen möglich sind. Mit steigender Komplexität wird diese Vorgehensweise immer schwieriger und aussichtsloser.

Ziel des hier vorgestellten Ansatzes ist es, direkte Abhängigkeiten zwischen Komponenten so weit wie möglich zu vermeiden und nur als Aufruf von Basis-Komponenten (z.B. Logging, eMail) zu erlauben. Weiterhin muss den impliziten Abhängigkeiten zwischen den Komponenten Rechnung getragen werden: Bei der Ausführung von Komponenten müssen unter Umständen Daten vorliegen, die andere Komponenten erstellt haben.

Um dieses Ziel zu erreichen, wurde die Technik der *Workflow-Programme*[1] entwickelt.

[1] Roth u. a. (2002, 2001a); Huang u. a. (2002, 2003)

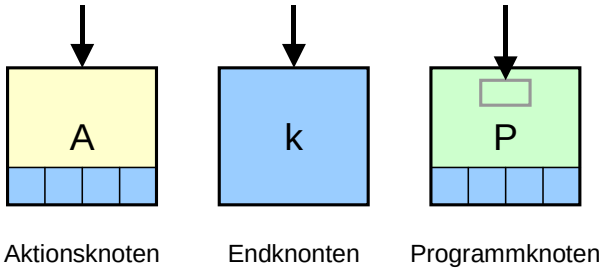

Abbildung 8.2: Knotentypen

Es handelt sich um eine spezielle Art von Automaten. Abbildung 8.1 zeigt ein Beispiel für ein Workflow-Programm. Es setzt sich aus einem oder mehreren Knoten und Kanten zwischen den Knoten zusammen.

8.1.1 Knoten und Kanten

Drei Knotentypen sind definiert.

Aktionsknoten werden mit Komponenten assoziiert, die bei Erreichen des Knotens ausgeführt werden. Nach deren Abarbeitung liefert diese Komponente eine Zahl (aus $\mathbb{N}_0$) als Rückgabewert, die als spätere Entscheidungsgrundlage für den Fortgang im Workflow-Programm dient. Die Menge der möglichen Rückgabewerte muss endlich und fest zwischen 0 und einem maximalen n liegen, wobei jedem möglichen Rückgabewert eine frei definierbar Bedeutung zugeordnet wird. Der Wert dient nicht zur Übermittlung von Daten wie „Alter“ oder „Größe“, sondern ist für grundsätzliche Ergebnisse der Berechnung wie „Erfolg“ oder „Misserfolg“ gedacht. Zum Beispiel könnte der Rückgabewert für eine Autorisierungskomponente folgendermaßen definiert sein:

0= Interner Fehler bei der Überprüfung der Autorisierung
1= Autorisierung OK
2= Autorisierung nicht OK

Eine Komponente kann ohne Einschränkung (ausschließlich) als Aktionsknoten an verschiedenen Stellen innerhalb eines Workflow-Programms verwendet werden.

Endknoten repräsentieren die Terminierung des aktuellen Workflow-Programms. Sie werden mit einer Zahl k (aus $\mathbb{N}_0$) statisch assoziiert, die als Rückgabewert des Workflow-Programms verwendet wird.

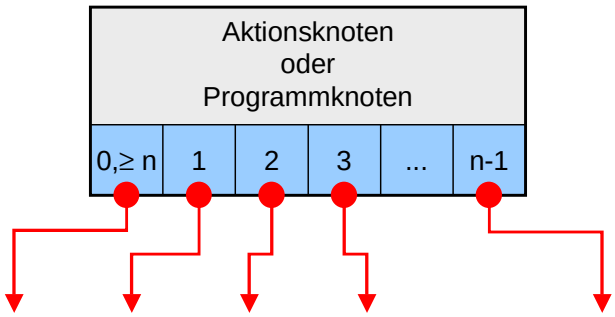

Abbildung 8.3: Rückgabewerte

Programmknoten repräsentieren Workflow-Programme. Als Teil eines bestehenden Workflow-Programms repräsentieren sie Unterprogramm-Aufrufe, mit denen auch Rekursionen möglich sind. Programmknoten oder Workflow-Programme können nur mit Erreichen von Endknoten beendet werden.

Programmknoten besitzen einen definierten Start-Knoten, der nicht selbst Programmknoten sein darf. Diese Einschränkung ist notwendig, um zu garantieren, dass zumindest ein Aktionsknoten oder ein Endknoten erreicht wird. Ein Beispiel, das mit dieser Einschränkung verhindert wird, lautet: Startknoten von Programm *P1* ist Programm *P2*. Startknoten von Programm *P2* ist Programm *P1*.

Start-Knoten sind, im Gegensatz zu den End-Knoten, kein eigener Knotentyp. Sie definieren ausschließlich den Startpunkt innerhalb eines Workflow-Programms oder Programmknotens.

Ein Programm-Knoten oder Workflow-Programm ist also eine Menge von Knoten, einem definierten Start-Knoten aus dieser Menge (ohne die Programmknoten) sowie Kanten zwischen den Knoten.

Abbildung 8.2 zeigt die visuelle Repräsentation der verschiedenen Knoten.

Die Kanten zwischen den Knoten definieren den Kontrollfluss im Workflow-Programm und sind letztendlich für die Abfolge der Ausführung der Komponenten verantwortlich. Jeder Programm- oder Aktionsknoten muss mindestens Ausgangspunkt einer Kante sein. Gehen von einem Knoten n Verbindungen aus, so wird die k-te Verbindung mit dem Rückgabewert (k-1) assoziiert (siehe Abb. 8.3).

Eine besondere Stellung nimmt die erste Kante ein. Diese steht nicht nur für den Rückgabewert 0, sondern auch für alle Rückgabewerte, die $\geqslant n$ sind. Zusammen mit der Bedingung, dass von jedem Knoten mindestens eine Kante ausgehen muss, ist garantiert, dass nach der Abarbeitung eines Knotens immer ein Folgeknoten definiert ist. Ein Workflow-Programm

kann durch den Aufbau den Workflow-Programm-Grafen niemals in einen nicht definierten Zustand gelangen.

Da die erste Kante für alle nicht definierten Rückgabewerte verantwortlich ist, steht dieser Übergang für ein unerwartetes Verhalten eines Knotens, was bei der Entwicklung der Komponente und der Spezifikation des Grafen berücksichtigt werden muss.

8.1.2 Workflow-Manager

Die Ausführung von Workflow-Programmen wird durch einen *Workflow-Manager* vorgenommen. Dem Workflow-Manager wird hierzu das Workflow-Programm sowie ein vorinitalisierter Datenpool übergeben, auf den im nächsten Abschnitt genauer eingegangen wird. Die Vorgehensweise des Workflow-Managers ist sehr einfach:

1. Ermittle den Startknoten.
2. Ist der ermittelte Knoten ein Endknoten, beende die Ausführung und liefere den Wert des Endknoten als Rückgabewert des Workflow-Programms.
3. Ist der ermittelte Knoten ein Aktionsknoten, führe den Code der assoziierten Komponente aus.
4. Ist der ermittelte Knoten ein Programmknoten, starte einen Workflow-Manager mit dem Programmknoten als Workflow-Programm sowie dem aktuellen Datenpool.
5. Ermittle aus dem Rückgabewert des Aktionsknotens oder Programmknotens und der von diesem Knoten ausgehenden Kanten den nächsten Knoten.
6. gehe zu Punkt 2.

Diese beschriebene Vorgehensweise wird im folgenden weiter präzisiert und ergänzt.

8.1.3 Datenpool

Bei der Ausführung eines Workflow-Programms ergibt sich durch die Rückgabewerte der aufgerufenen Komponenten und den definierten Kanten eine Abfolge von Komponenten-Aufrufen. Die Grundlage für die Ausführung sind Eingabedaten. Ebenso können Komponenten selbst auch Daten erzeugen. Während der Ausführung eines Workflow-Programms existiert ein Datenpool, aus dem Daten zu den Komponenten geleitet werden können oder der mit weiteren Daten einer Komponente erweitert werden kann.

Abbildung 8.4 zeigt die Abfolge von Komponenten eines Beispiel-Workflow-Programms und der dabei involvierten Daten. Vor Ausführung des Programms durch den Workflow-Manager wird der Datenpool durch das aufrufende Programm initial mit den Variablen S und T

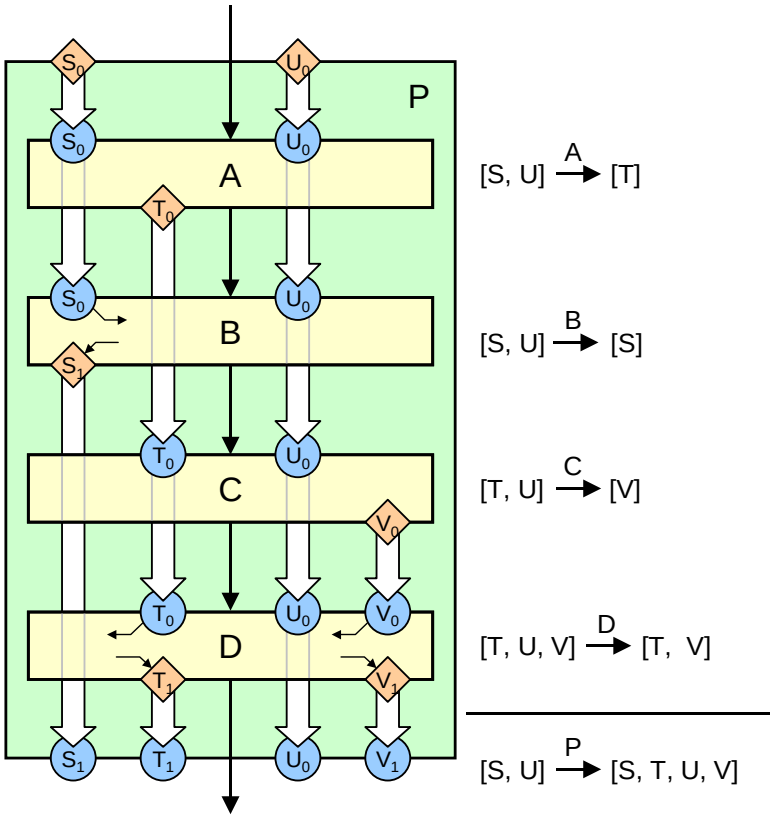

Abbildung 8.4: Datenflüsse

belegt. Jede Komponente muss definieren[2], welche Variablen es aus dem Datenpool benötigt und welche es selbst liefert. In der Abbildung benötigt beispielsweise die Komponente *B* die Daten der Variablen *U* ausschließlich lesend. Im Gegensatz dazu wird die Variable *S* lesend benötigt und möglicherweise während der Abarbeitung verändert. An der Variablen *T* ist die Komponente *B* nicht interessiert.

Komponenten besitzen keinen direkten Zugriff auf den Datenpool. Vielmehr ist es so, dass vor deren Ausführung alle gewünschten Daten durch den Workflow-Manager über für die Komponente definierte Interfaces zur Komponente übertragen werden (siehe Abb. 8.5: (1)) und nach deren Ausführung (2) die festgelegten Daten ausgelesen werden (3).

[2] in der Praxis durch Interfaces (siehe Abs. 8.1.4)

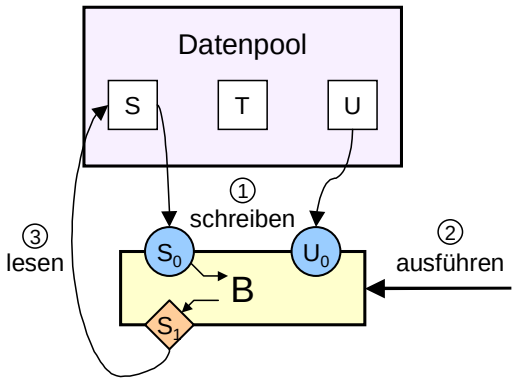

Abbildung 8.5: Datenpool

8.1.4 Namensschema

Durch Festlegung der Workflow-Programme wird die Reihenfolge bestimmt, in der die Komponenten nacheinander abgearbeitet werden. Eine Komponente selber hat kein Wissen darüber, welche Komponenten vor der eigenen Ausführung aktiv waren oder welche Komponenten folgen werden. Einzig die Variablen, die eine Komponente im Datenpool erwartet, führt zu einer gewissen Abhängigkeit zwischen den Komponenten, da andere Komponenten erwartete Variablen zuvor erzeugt haben müssen[3]. Welche Komponente dies tut und ob die Variable von anderen Komponenten inzwischen verändert wurde, darf die Bedeutung des Datums einer Variablen nicht verändern. Beispielsweise darf eine Variable *X* nicht zunächst Altersangaben, später Größenangaben beinhalten.

Der Bezeichnung einer Variablen im Datenpool kommt somit eine besondere Stellung zu. Sie ist fest mit einer Bedeutung assoziiert und wird von der Komponente bestimmt, die die Variable erzeugt. Als Konsequenz dürfen zwei Komponenten nicht zufälligerweise eine gleiche Bezeichnung für Variablen festlegen und unterschiedlich interpretieren.

In der Praxis wurde die Möglichkeit zur Mehrfachbelegung von Variablenbezeichnungen durch die Einführung eines eindeutigen Namensschemas gelöst. Komponenten, die Variablen vom Datenpool lesen oder ändern wollen, definieren zu diesem Zweck Interfaces mit `get`-Methoden für Variablen, die in den Datenpool geschrieben werden sollen und `set`-Methoden für Variablen, die aus dem Datenpool gelesen werden sollen (siehe Abb. 8.6).

Ein oder mehrere solcher Interfaces definieren eindeutig das Ein- und Ausgabeverhalten der implementierenden Komponente. Wie allgemein üblich beziehen sich die Methoden

[3] Wenn sie nicht vor der Ausführung des Workflow-Programms initialisiert wurden (siehe Abb. 8.4, Variable *S* und *U*).

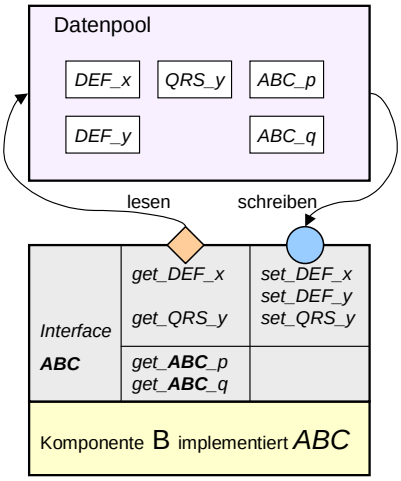

Abbildung 8.6: `set`-/`get`-Methoden

`getX` und `setX` auf die Variable `x`[4]. Der Workflow-Manager ermittelt zu einer Komponente alle Interfaces, die das `WorkflowModuleIOBehaviour`-Interface erweitern und ermittelt von diesen Interfaces alle `set`- und `get`-Methoden, die mittels Reflection ausgeführt werden. Folgende Regel gilt:

Eine im Interface definierte `set`-Methode einer Komponente wird nur dann vom Workflow-Manager zum Schreiben einer Variablen verwendet, wenn die entsprechende Variable im Datenpool vorliegt. Die `get`-Methode wird zum Lesen nur dann ausgeführt, wenn eine der beiden Bedingungen erfüllt ist:

- Zu jeder `get`-Methode einer Variablen ist auch eine entsprechende `set`-Methode im Interface definiert. Damit wird garantiert, dass nur zuvor gelesene Variablen verändert werden können.
- Die Bezeichnung des Interfaces ist Bestandteil der Variablenbezeichnung[5]. Diese Regel betrifft die neuen Variablen.

Ein Interface
`component.bankstuff.BncIntf`

[4] Man beachte: Variablen beginnen in Java per Konvention meist mit kleinen Buchstaben, zur besseren Lesbarkeit wird der erste Buchstabe der Variablenbezeichnung in der `set`- und `get`-Methode jedoch groß geschrieben.

[5] Genauer: Die Variablenbezeichnung setzt sich zusammen aus einem Underscore-Zeichen, der Interface-Bezeichnung inklusive Package-Bezeichnung (Punkt wird jeweils durch Underscore ersetzt), einem weiteren Underscore-Zeichen und schließlich einer beliebigen Zeichenfolge.

darf beliebige Methoden der Form
`get_component_bankstuff_BncIntf_`<*Variablensuffix*>
definieren, um neue Variablen dem Datenpool hinzuzufügen. Die neuen Variablen haben in diesem Beispiel die Bezeichnungen
`_component_bankstuff_BncIntf_`<*Variablensuffix*>.

Wenn keine zwei Komponenten im Workflow-Programm dasselbe `WorkflowModuleIOBehaviour`-Interface implementieren, kann garantiert werden, dass keine zwei Methoden neue Variablen mit derselben Bezeichnung definieren. Per Konvention müssen die Interfaces eindeutig zu Komponenten zugeordnet werden. Einzig, wenn in Zukunft eine Komponente durch eine neue Version mit anderem Namen ersetzt wird oder eine andere Komponente deren Funktion übernimmt, darf diese Regel übergangen werden. Allerdings ändert dies nichts an den Bezeichnungen der Variablen im Datenpool, da die Interface-Definitionen bestehen bleiben. Dies wäre nicht möglich, wenn allein die Komponenten-Bezeichnung in das Namenenschema eingegangen wäre.

8.2 Pipelining

Mit den zuvor beschriebenen Workflow-Programmen ist ein flexibler und erweiterbarer Aufbau eines Servers möglich. Allerdings ist die Abarbeitung eines Workflow-Programms sequentiell. Zwei Komponenten eines Workflow-Programms können nicht gleichzeitig aktiv sein.

Ein Ziel des neuen Middleware-Ansatzes war jedoch das Streaming von Daten. Hier ist die gleichzeitige Ausführung von mindestens zwei Komponenten sinnvoll, um Daten vom Client lesen und gleichzeitig senden zu können. Dies erfordert den Einsatz von Threads. Um diesen neuen Aspekte umsetzen zu können, wurde der bestehende Ansatz erweitert. Er besteht nun aus zwei Teilen:

- Das statische Workflow-Programm dient der sequentiellen Abarbeitung von Komponenten. Auf Grundlage der Rückgabewerte wird entschieden, welche Komponente als nächstes ausgeführt wird. Dies entspricht der bisherigen Konzeption aus Abschnitt 8.1.

- Während der Abarbeitung des Worfklow-Programms wird sukzessive eine dynamische Objekt-Pipeline, bestehend aus Pipeline-Threads aufgebaut. Ein Pipeline-Thread kann dabei als Objekt-Produzent (mit einem Objekt-Ausgang) und/oder Objekt-Konsument (mit einem Objekt-Eingang) in Aktion treten. Konsumenten lesen aus dem Objekt-Eingang einen kontinuierlichen Objekt-Strom, den ein Produzent zuvor in seinen Objekt-Ausgang geschrieben hat.

  Bei Erreichen einer Komponente kann der Workflow-Manager in Abhängigkeit von der Art der Aktionskomponente, die Objekt-Pipeline um maximal einen Pipeline-

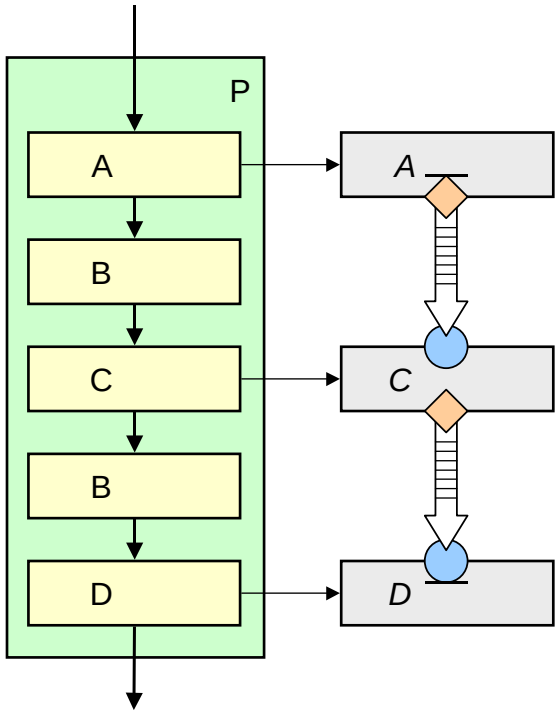

Abbildung 8.7: Aufbau einer Objekt-Pipeline

Thread erweitern. Nachdem das statische Workflow-Programm beendet ist, kann die Objekt-Pipeline noch aktiv bleiben, beispielsweise als Bestandteil des Servers, bis alle Streaming-Daten vom Client gelesen und gesendet wurden.

8.2.1 Pipeline-Elemente

Abbildung 8.7 zeigt ein Beispiel, wie neben der Ausführung der Komponente (links) durch den Workflow-Manager eine Objekt-Pipeline (rechts) aufgebaut wird. Deutlich wird auch, dass nicht jede Komponente zu einer Erweiterung der Pipeline führt. Eine Komponente legt durch die Implementation eines entsprechenden Interfaces fest, welcher der folgenden Arten es entspricht:

- Die Komponente erzeugt den Beginn einer Pipeline (Objekt-Out-Thread).
- Die Komponente erzeugt das Ende einer Pipeline (Objekt-In-Thread).

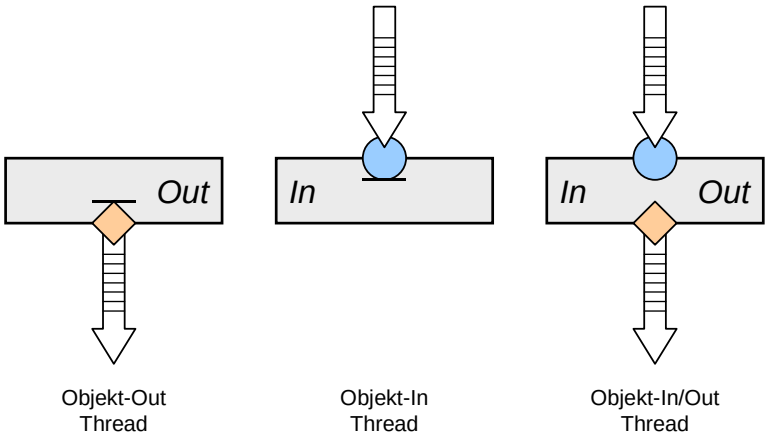

Abbildung 8.8: Thread-Typen

- Die Komponente erzeugt ein Zwischenstück innerhalb einer Pipeline (Objekt-In-Out-Thread).
- Die Komponente erweitert die Pipeline nicht.

Abbildung 8.8 führt die Symbole für die entsprechenden Pipeline-Threads auf.

Mit der Abarbeitung eines Workflow-Netzes muss nicht notwendigerweise eine Pipeline allein aufgebaut werden. Dies können beliebig viele sein. Allerdings muss das Netz so konzipiert sein, dass keine unvollständigen Pipelines erzeugt werden. Der Workflow-Manager überprüft vor der Erweiterung der Pipeline die folgenden Bedingungen:

- Auf den Beginn einer Pipeline oder ein Zwischenstück darf nur das Ende oder ein Zwischenstück folgen.
- Auf das Ende einer Pipeline darf nur der Beginn einer Pipeline folgen.

Alle anderen Abfolgen führen zum Abbruch der Abarbeitung des Workflow-Programms.

Zwischen dem Objekt-Ausgang und dem in der Pipeline folgenden Objekt-Eingang wird ein Objekt-Puffer angelegt, der exklusiv für die beiden beteiligten Pipeline-Elemente existiert. In ihn kann der Objekt-Produzent schon Objekte schreiben, bevor der Objekt-Konsument der Pipeline hinzugefügt wurde. Die Größe dieses Puffers ist zunächst beliebig, kann aber von den Konsumenten in seiner Größe limitiert werden, um den Datendurchsatz zu begrenzen. In diesem Fall blockiert der Produzent bei Erreichen des Limits, bis der Konsument wieder ein Objekt abgearbeitet hat.

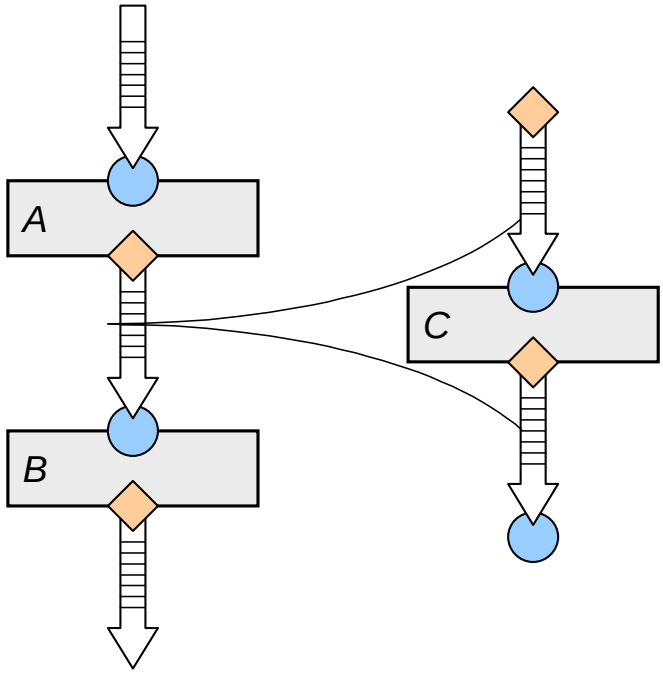

Abbildung 8.9: Erweiterung der Pipeline

Der Produzent kann den Puffer schließen, wenn keine weiteren Objekte mehr erzeugt werden, um sich anschließend zu beenden. Der Konsument kann noch alle im Puffer verbliebenen Objekte lesen und erkennt dann, dass der Puffer geschlossen wurde. Am Ende der Objekt-Verarbeitung baut sich somit eine Objekt-Pipeline vom Anfang der Pipeline bis zum Ende der Pipeline sukzessive ab.

Zwischen Objekt-Produzent und Objekt-Konsument findet neben dem Schreiben und Lesen der Objekte keine weitergehende Kommunikation statt. Der Konsument muss somit die Bedeutung der gelesenen Objekte kennen. Dies hat Konsequenzen bei einer Erweiterung des Workflow-Programms. Führt dies zu einem neuen Zwischenstück zwischen bisher benachbarten Pipeline-Elementen, so darf dies die Struktur und Semantik der übertragenen Objekte nicht verändern (siehe Abb. 8.9).

In der Praxis ist es wahrscheinlicher, dass das Netz um eine Entscheidungskomponente erweitert wird, die den Weg im Workflow-Programm bestimmt. Die Pipeline wird dann in Abhängigkeit von der Entscheidung um ein neues, alternatives Pipeline-Element erweitert (siehe Abb. 8.10). Im Beispiel wurde zwischen *A* und *C* die neue Entscheidungskomponente *B*, die den Objekt-Strom nun auch alternativ zu *D* leiten kann. Dabei ist *B* selbst nicht Teil der Pipeline.

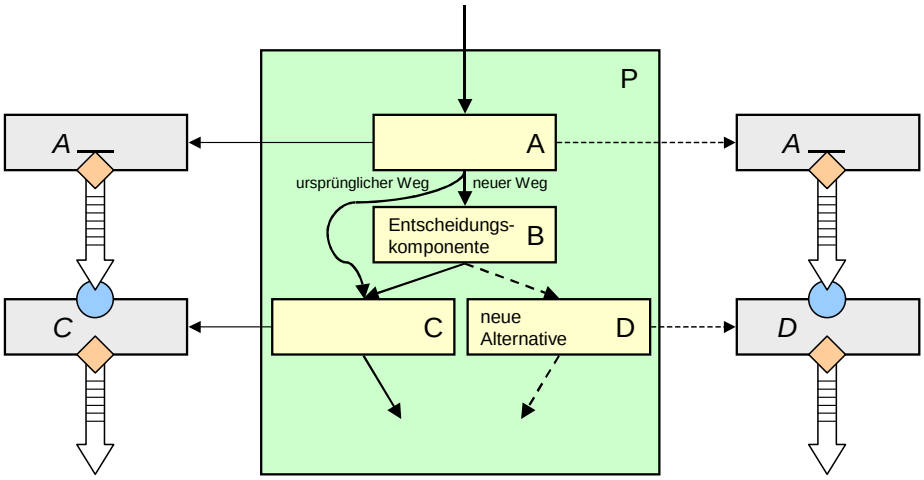

Abbildung 8.10: Alternative Erweiterung der Pipeline

8.2.2 Aufbau der Pipeline

Wichtig bei der Abarbeitung des Workflow-Programms und der Erstellung der Pipeline ist eine genaue Synchronisation zwischen Workflow-Komponente und Pipeline-Thread. Folgende Schritte werden durchgeführt:

Der Workflow-Manager

1. erzeugt eine Komponenten-Instanz;
2. übergibt alle gewünschten Variablen aus dem Datenpool mittels der `set`-Methoden an die Komponente;
3. lässt die Komponente ein assoziiertes Pipeline-Thread-Objekt erzeugen;
4. erzeugt einen Objekt-Puffer;
5. übergibt dem Pipeline-Thread alle benötigten Objekt-Eingangs- und Objekt-Ausgangs-Puffer;
6. startet das Pipeline-Thread als Thread;
7. führt die Komponente aus und bestimmt aus dem Rückgabewert die folgende Komponente;
8. liest mittels der `get`-Methode die definierten Variablen und schreibt sie in den Datenpool.

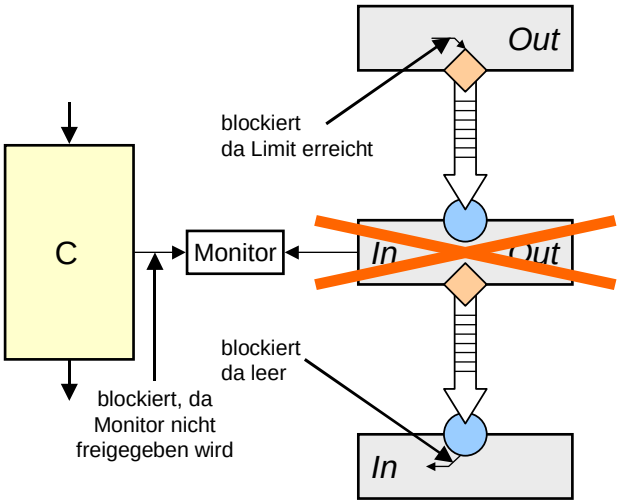

Abbildung 8.11: Problem bei fehlerhaften Pipeline-Threads

Folgendes ist bei Punkt 7 zu beachten: Möglicherweise kann eine Komponente eine Entscheidung über ihren Rückgabewert erst dann fällen, wenn das assoziierte Pipeline-Thread-Objekt eine bestimmte Anzahl von Objekten aus dem Objekt-Eingangspuffer gelesen hat. Hierzu muss die Komponente in Punkt 3 bei der Erzeugung des Pipeline-Thread-Objekts auch ein Monitor-Objekt erstellen und dem Pipeline-Thread übergeben. Die Komponente blockiert dann in Punkt 7 solange an diesem Monitor, bis dieser von dem Pipeline-Objekt freigegeben wird.

8.2.3 Fehlerbehandlung

Wie schon in einem früheren Abschnitt beschrieben, ist es in Java nicht möglich, Threads von außen zu beenden. Jeder Thread muss dies selbst durchführen. Wenn ein Thread jedoch in eine nicht endende Rundlaufschleife gerät oder eine Exception wirft, kann das saubere Ende eines Threads nicht garantiert werden. In dem Falle hat dies jedoch eine schwerwiegende Konsequenz bezüglich der beschriebenen Objekt-Pipeline. Gerät ein Pipeline-Thread in einen inkonsistenten Zustand, wartet das darauf folgende Glied der Pipeline auf eingehende Objekte, die jedoch nicht produziert werden. Das voranstehende Glied der Pipeline blockieren, da dessen Objekt-Ausgangspuffer sein Limit erreicht hat (siehe Abb. 8.11). Alle folgenden Glieder der Pipeline blockieren letztendlich, da auch sie keine Objekte mehr lesen können, alle voranstehenden Glieder blockieren, da auch deren Limits erreicht werden.

Zwei Problemklassen können unterschieden werden:

- Die Klasse der Threads, die in eine permanenten Rundlaufschleife hineinlaufen und für alle Zukunft weder Daten konsumieren, noch produzieren.

- Die Klasse der Threads, die eine Exception auslösen und ihre Arbeit einstellen.

Die erste Problemklasse umfasst Threads mit einem prinzipiellen Fehler im Design der Implementation. Solche Designfehler dürfen auch an keiner anderen Stelle im Server vorkommen. Eine Unterstützung der Server-Infrastruktur ist hier nicht möglich. Diese kann jedoch für die zweite Problemklasse zur Verfügung gestellt werden:

Wird eine Exception innerhalb eines Threads ausgelöst, so gibt es zunächst kein Objekt, das diese Exception mittels `try` und `catch` abfängt. Solche Exceptions werden dann an der Systemkonsole signalisiert. In Java existiert jedoch das Konstrukt der Thread-Group. In einem Thread-Group-Objekt können Threads bei deren Instanziierung registriert werden[6]. Löst ein beliebiger Thread dieser Gruppe eine Exception aus, wird die `uncaughtException`-Methode in dem Thread-Group-Objekt ausgeführt und zwar mit dem Thread sowie der ausgelösten Exception als Parameter.

Im Smart Data Server wird dieses Konstrukt durch Überschreiben dieser Methode ausgenutzt, um die Pipeline sukzessive abzubauen. Ausgehend von einem abgebrochenen Thread werden die benachbarten Objekt-Ein- und Objekt-Ausgabepuffer abgebrochen. Dies unterscheidet sich von dem normalen Beenden eines Objekt-Puffers dadurch, dass bei den benachbarten Objekt-Konsumenten und Objekt-Produzenten Exception ausgelöst werden. Diese Information kann ausgenutzt werden, um sich sinnvoll zu beenden. Der Abbau der Pipeline erfolgt auf diese Art schrittweise in Richtung des ersten und letzten Glieds der Pipeline.

Möglicherweise blockiert auch eine Komponente an einem Monitor, der durch das assoziierte (aber abgebrochene) Pipeline-Thread hätte befreit werden sollen (siehe Abb. 8.11). Um dies zu verhindern, können diese Monitore in der Thread-Group registriert werden. Im Falle einer Exception im Pipeline-Thread werden dann alle registrierten Monitore durch Auslösen einer speziellen Exception freigegeben, die der Komponente den Abbruch des Threads signalisiert. Zusätzlich besteht die Möglichkeit, die `uncaughtException`-Methode selbst zu überschreiben, also eigene Thread-Group-Klassen zu definieren, um die Exceptions durch eigenen Code behandeln zu können. Dabei kann jedes Pipeline-Thread-Objekt einer anderen Thread-Group zugewiesen werden.

Mit den bereitgestellten Mitteln ist der kontrollierte Abbau einer Pipeline ohne großen Aufwand auf der Seite des Entwicklers möglich. Dem Entwickler bleibt es aber überlassen, das Ende eines Pipeline-Elementes selbst durchführen oder dies vom System durchzuführen zu lassen.

[6] Genauer: Bei der Instanziierung eines Threads, kann das Thread-Group-Objekt als Parameter übergeben werden.

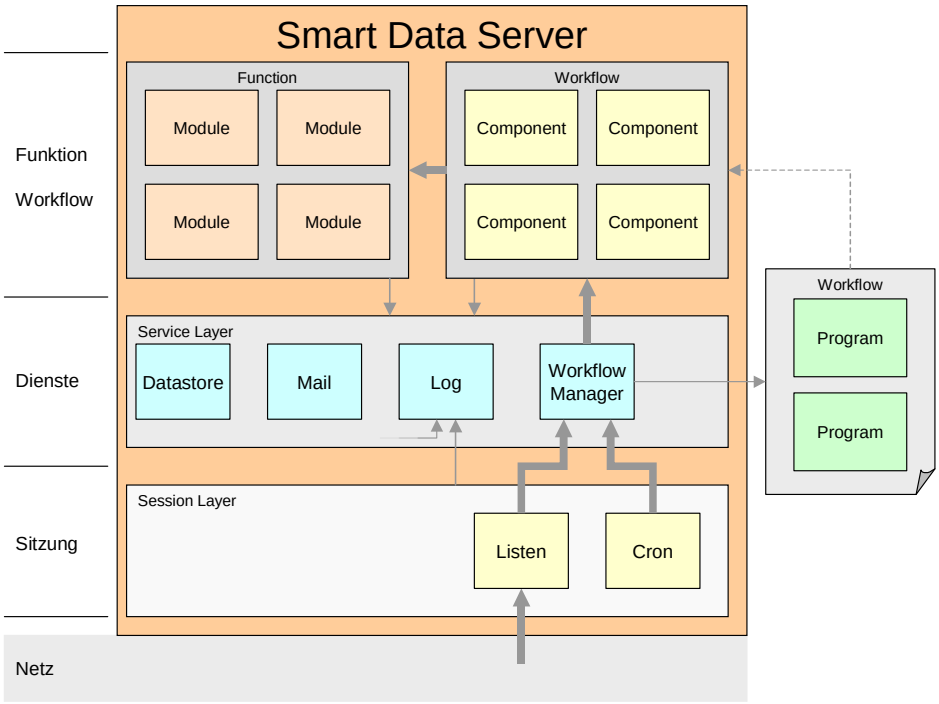

Abbildung 8.12: Workflow-Programme im SDS

8.3 Workflow-Programme im SDS

Die Workflow-Technologie führt im Smart Data Server zu einer Flexibilisierung der internen Abläufe. Abbildung 8.12 zeigt, wie die Workflow-Programme im Server repräsentiert sind:

Zentrale Einheit zur Abarbeitung der Workflow-Programme ist der *Workflow-Manager* des Service Layers. Der Workflow-Manager ist für den Kontrollfluss in den Workflow-Programmen verantwortlich, also für das Ausführen der einzelnen Komponenten im Workflow-Programm und der Ermittlung der darauf folgenden Komponenten. Zusätzlich hat er die Kontrolle über den Datenpool und den Transfer der Variablen zwischen Komponente und Datenpool. Die Objekt-Puffer zwischen den Pipeline-Elementen werden durch den Workflow-Manager angelegt, ebenso ist diese Einheit für die Fehlerbehandlung im Falle von Exceptions in der Pipeline verantwortlich.

Der Workflow-Manager hat Zugriff auf die Workflow-Programme, die Teil der Server-Konfiguration sind und als Textdatei vorliegen. Durch Editieren dieser Datei können die Workflow-Programme und damit das Verhalten des Servers verändert werden. Dort wird die

```
ProgNode p1Prog {
  Code { // Zuordnung eines Bezeichners mit Code:
    aCode = ti.workflow.test.A
    bCode = ti.workflow.test.B
    cCode = ti.workflow.test.C
  }
  Nodes { // Festlegung der Knoten:
    // End-Knoten:
    e0 = EndNode (0)
    e1 = EndNode (1)
    e2 = EndNode (2)

    // Aktionsknoten:
    // Verschiedene Aktionsknoten können mit
    // demselben Code assoziiert werden.
    a = ActionNode (aCode)
    b = ActionNode (bCode)

    // Programm-Knoten:
    // Verschiedene Programmknoten können mit
    // demselben Programm assoziiert werden.
    p2 = ProgNode (p2Prog)

    // Festlegung des Start-Knotens:
    start (a)
  }
  Edge { // Festlegung des Netzes:
    // connect (<Knoten> <result=0> <result=1> ... <result=n>)
    connect (a, e0, b, c)
    connect (b, e0, p2, e2, c)
    connect (c, e0, e2)
    connect (p2, e0, e1, b)
  }
}

ProgNote p2Prog {...}
```

Code 8.1: Beispiel-Workflow-Programm

Beziehung zwischen Aktionsknoten und dem Code der Workflow-Komponente hergestellt (siehe Code 8.1 zu Abbildung 8.1 auf Seite 90). Die Instanziierung der Komponenten wird vom Workflow-Manager durchgeführt.

Der Service Layer beinhaltet ausschließlich Basis-Dienste, die Komponenten und Module zur Erfüllung ihrer Aufgaben verwenden können, also Datenbank-Zugriff, eMail oder

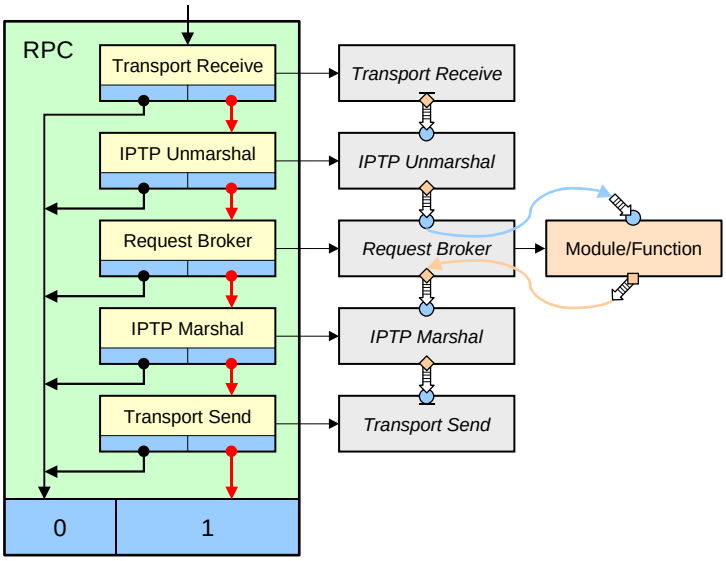

Abbildung 8.13: RPC mittels Workflow-Programm

Logging. Sie sind jedoch nicht Teil des Workflows. Im Session Layer befinden sich nur noch die Initiatoren der Ausführung von Workflow-Programmen. Bei einer Anfrage an einen Server-Socket beispielsweise wird das RPC-Workflow-Programm durch das Listen-Modul (siehe Abs. 6.1 auf Seite 74) gestartet, wobei der Datenpool mit dem Socket-Objekt initialisiert wird.

Das RPC-Workflow-Programm ist in seiner einfachsten Variante in Abbildung 8.13 dargestellt. Zwei Komponenten dienen zur Entgegennahme und zum Versenden der Anfrage und Antwort (Transport Receive/Send), zwei weitere Komponenten dienen zum Ent- und Verpacken der Anfrage und Antwort (IPTP Unmarshal/Marshal) und eine Komponente ist für die Ausführung der Funktion im gewünschten Modul verantwortlich (Request Broker). Diese Komponente verwendet hierzu die Refection-API von Java. Um ein Teil der Pipeline sein zu können, werden die Objekt-Eingangs- und Objekt-Ausgangs-Puffer an die Funktion übergeben. Der Rückgabewerte 0 des RPC-Workflows signalisiert den erfolgreichen Ablauf des Programms, der Wert 0 signalisiert einen nicht behebbaren Fehler.

Der dargestellte RPC-Mechanismus besitzt keine alternativen Pfade und nutzt die Möglichkeiten der Workflow-Programme aus diesem Grund nicht vollständig aus. Abbildung 8.14 zeigt eine weitere Ausbaustufe des RPC-Mechanismus[7], in der neben dem IPTP auch SOAP als Protokoll aufgeführt, eine Authentifikations- und Autorisationseinheit involviert

[7] Zur Vereinfachung ohne Kanten für die Fehlerbehandlung (Rückgabewert 0) und ohne korrespondierende Pipeline.

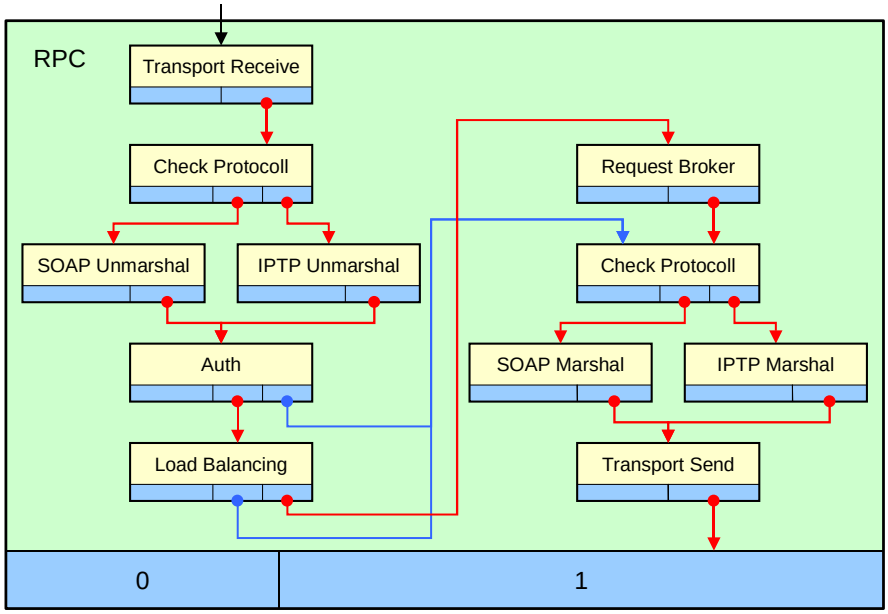

Abbildung 8.14: Erweiterter RPC

und Load-Balancing zwischen mehreren Smart Data Servern zur Lastenverteilung implementiert sind. Es wird deutlich, dass es sich um eine Erweiterung des einfachen RPC-Mechanismus handelt und die dort definierten Komponenten unverändert übernommen werden können.

Abhängigkeiten zwischen Komponenten können nicht vollständig aufgelöst werden, da bestimmte Daten erwartet und produziert werden müssen. Allerdings vermeiden Workflow-Programme explizite Abhängigkeiten (direkte Methodenaufrufe) und sehen nur implizite Abhängigkeiten (Variablen schreiben/lesen) vor. Der Entwickler von Workflow-Programmen und -Komponenten kann diese Fähigkeit nutzen, einfach neue Server-Infrastrukturen zu entwerfen und bestehende zu erweitern.

Ein weiterer wichtiger Punkt ist das Verhältnis von Funktionsmodulen zu Workflow-Komponenten. Funktionsmodule sind der Endpunkt von RPC-Anfragen und sind innerhalb von Workflow-Programmen durch den Request Broker eingebunden. Wenn es um die Erweiterung von Funktionen geht, die über RPC erreichbar sein sollen, betrifft das die Funktionsmodule (siehe Abb. 8.12 auf Seite 104). Durch die Verschiebung von Server-Funktionalitäten in die Workflow-Programme stehen diese Teile des Servers jedoch auch dem Anwendungsentwickler frei zur Verfügung, um Veränderungen und Erweiterungen des Servers vorzunehmen. Die Grenzen zwischen Server-Funktionalität und RPC-Funktionalität sind in diesem Punkt bewusst aufgeweicht worden.

Teil III

Der Smart Data Server in der Praxis

9 Fallstudien

Erfahrung ist der Anfang aller Kunst und jedes Wissens.

ARISTOTELES

Der Smart Data Server ist nicht nur ein theoretisches Konzept. Er existiert als Implementierung und konnte in einer Reihe von Projekten seine Fähigkeiten unter Beweis stellen. Dieses Kapitel beschreibt eine Auswahl dieser Projekte.

9.1 Web-basiertes Personalmanagement

Im Rahmen des Projektes „Teilzeitverfahren für Lehrkräfte“ wurde ein System erstellt, mit dem teilzeitbeschäftigte Lehrer über das Internet ihre Anträge verlängern lassen können[1]. Diese Daten werden dann im weiteren Verlauf von verschiedenen Benutzergruppen genehmigt, bearbeitet und kontrolliert, um dann am Ende in die lokale Datenbank im Intranet übernommen zu werden und einen Bescheid zu erstellen, der dem Antragsteller zugestellt wird. Der SDS agiert dabei als Integrationsplattform zwischen der Datenbank sowie den Benutzergruppen.

Bei der Erstellung stand die Entwicklung von Modulen im SDS sowie deren einfache Anwendung auf der Client-Seite im Vordergrund.

Das Verfahren zur Eingabe von Antragsdaten ist dynamisch. Der Benutzer wird durch eine Reihe von HTML-Dialogen geführt, deren Inhalt sich aus den vorangegangenen Eingaben sowie Daten, die in einer Datenbank vorliegen, zusammensetzt. Auf der letzten Dialogseite bestätigt der Benutzer seine Angaben. Die Aufbereitung der Eingabemasken wird dynamisch mittels Servlets auf dem Web-Server durchgeführt. Dabei werden Daten aus den vorangegangenen Dialogen in einer Servlet-Sitzung zwischengespeichert. Die Servlets greifen auf den SDS zu, um dort die Daten, die für die Dialogerstellung benötigt werden, nachzufragen.

Damit Benutzer über das Internet Formulardaten eingeben können, sah das Konzept vor, einen Web-Server zur Verfügung zu stellen (siehe Abb. 9.1). Nach der Eingabe der Daten mit Hilfe eines Browsers müssen die Daten in das Intranet der Behörde transferiert werden. In dem ersten Konzept stand der Web-Server in einer demilitarisierten Zone zwischen Internet und Intranet der Behörde. Zur Aufbereitung der Formulare und zum Übertragen der

[1] Roth u. a. (2001b); Roth und Meinel (2002); Roth (2001); Roth u. a. (2001a)

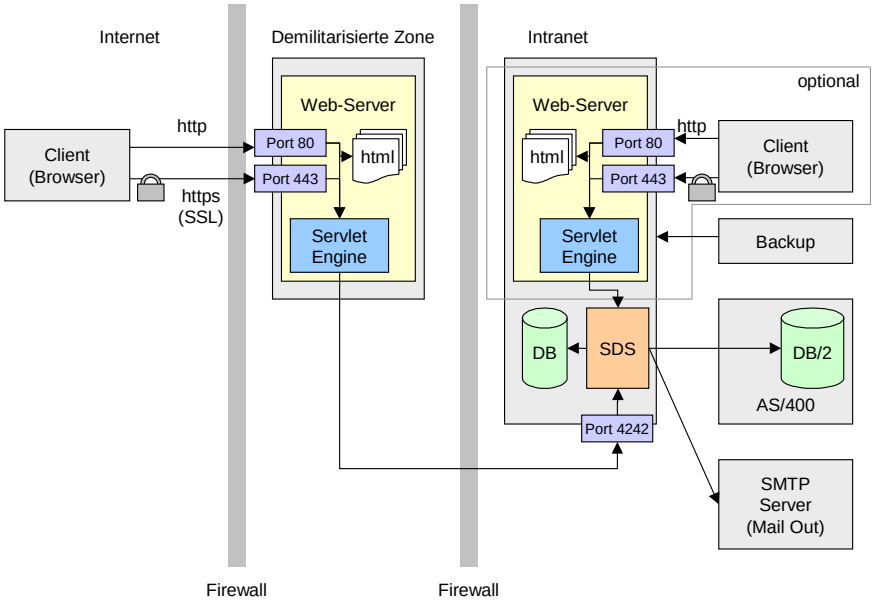

Abbildung 9.1: Entwurf der Topologie

Antragsdaten in die Datenbank des Intranets greifen die Servlets aus der demilitarisierten Zone über einen dedizierten Kanal auf den SDS im Intranet zu. Der SDS im Intranet ist so konfiguriert, dass das Servlet nur die Funktionalität, die zur Dialogerstellung benötigt wird, abrufen kann. So ist garantiert, dass über diesen Weg sensitive Daten des Intranets nicht ins Internet gelangen können.

Mit der Bestätigung der Formulardaten werden diese an den SDS im Intranet übermittelt, der diese zunächst in einer temporären Datenbank zwischenspeichert.

Die im Folgenden am Verfahren beteiligten Benutzergruppen greifen, wenn nötig, über den Internet-Webserver und dessen Servlets auf den SDS im Intranet zu oder über einen weiteren Web-Server im Intranet. Erst im letzten Schritt des Arbeitsablaufes werden die geprüften und bearbeiteten Daten in die juristische Datenbank[2] des Intranets überführt.

Der geplante Vorgang konnte aus technischen Gründen kurzfristig nicht umgesetzt werden. Der Grund dafür war das Fehlen eines ausgereiften Firewall-Konzeptes, das es ausschloss, dass ein außenstehender Rechner (ein Rechner im Internet oder innerhalb der demilitarisierten Zone) auf das Intranet zugreift. Diese Entscheidung war unabhängig von der Tatsache, dass die IP-Nummer des Rechners, der verwendete Port sowie das übertragene Protokoll bekannt waren.

[2]Unter einer *juristischen Datenbank* wird die Datenbank in einem Unternehmen oder einer Behörde verstanden, auf deren Grundlage juristisch bindende Entscheidungen getroffen werden.

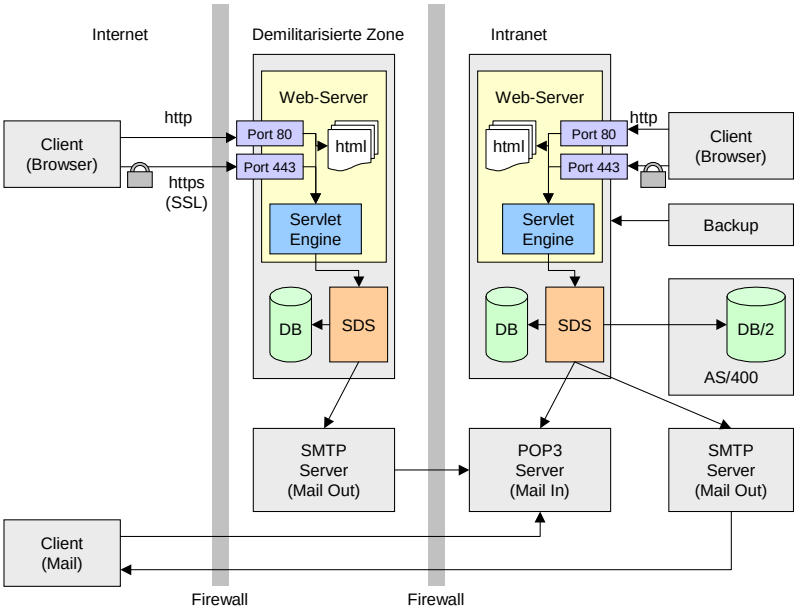

Abbildung 9.2: Implementation-Konzept

Die Entscheidung wurde sehr kurzfristig mitgeteilt, nachdem schon alle Planungen kommuniziert und die Entwicklungen der Servlets sowie weite Teile der Module des SDS zu einem großen Teil fertig gestellt waren.

Ziel der an die neue Infrastruktur angepassten Lösung war die Übernahme der bis dahin implementierten Code-Basis. Diese Anforderung konnte in vollem Maße erfüllt werden. Die notwendigen Erweiterungen blieben überschaubar.

In der neuen Variante wurde ein weiterer SDS inklusive lokaler Datenbank auf dem Webserver im Internet installiert (siehe Abb. 9.2). Das Servlet des Web-Servers der demilitarisierten Zone kommuniziert nicht, wie vorgesehen, mit dem Intranet-SDS, sondern nur noch mit dem Internet-SDS. In der lokalen Datenbank des Internet-SDS werden nur die Datenbestände gehalten, die zur Erstellung der Eingabemasken benötigt werden (keine personenbezogenen Daten). Die von den Benutzern eingegebenen Formulardaten werden dort nicht (auch nicht temporär) gespeichert.

Ein entscheidender Punkt beim Zugriff des Servlets auf den SDS ist das Übertragen der vom Benutzer bestätigten Formulardaten. Das Servlet spricht hierzu eine bestimmte Funktionalität des SDS an (Modul `Antrag`, Funktion `InsertAntrag`). Diese Funktion ist in beiden SDS, dem Internet-SDS sowie dem Intranet-SDS, implementiert, jedoch mit unterschiedlichen Seiteneffekten.

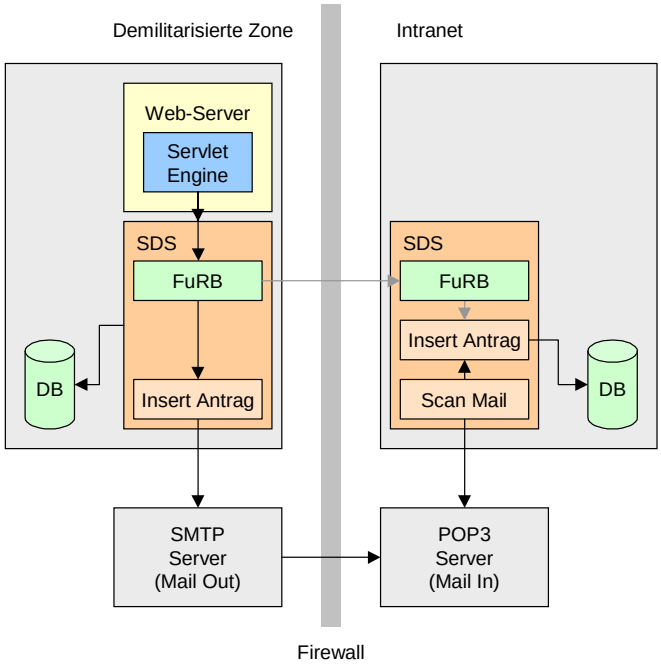

Abbildung 9.3: Zugriff auf die Funktion `InsertAntrag`

Der Intranet-SDS nimmt die Formulardaten wie vorgesehen entgegen und überträgt sie in die lokale Datenbank (siehe Abb. 9.3). Der Internet-SDS hingegen nimmt die Formulardaten und versendet sie per eMail an einen dedizierten Benutzeraccount im Intranet. Diese Art der Kommunikation von außen in das Intranet ist nicht beschränkt. Die Formulardaten werden als Textinformationen übermittelt, d.h., es werden keine Mail-Anhänge verwendet, die möglicherweise zurückgewiesen werden könnten[3].

Auf der Seite des Intranets wertet ein zusätzliches Modul dieses Postfach zyklisch aus, analysiert die Mail-Daten und führt mit den ermittelten Daten die `InsertAntrag`-Funktion des Intranet-SDS aus.

Die Entscheidung für diese Vorgehensweise hat mehrere Vorteile. Neben dem Erhalt der entwickelten Komponente ist diese Lösung auch zukunftssicher. Sollte die Infrastruktur in Zukunft einen Zugriff auf das Intranet über einen dedizierten Kanal (definierte IP, Port, Protokoll) aus dem Internet heraus erlauben, so reichen kleine Änderungen an der Konfiguration des Internet-SDS aus, um den Umweg über eMail zu beseitigen. Der im SDS vorhandene Function Request Broker (FuRB) entscheidet in Abhängigkeit der Server-Kon-

[3] Einige Firmen weisen grundsätzlich eMails mit Anhängen ab, unabhängig davon, ob sie infiziert sind oder nicht.

figuration, ob eine Anfrage intern oder extern von einem entfernten SDS abgearbeitet wird. Da die Syntax der Anfrage an die `InsertAnfrage`-Funktion unverändert bleibt, ist das Weiterleiten der Anfrage an den Intranet-SDS möglich.

Das vorliegende Beispiel macht deutlich, dass eine konkrete Problemlösung für Anwendungen von Behörden und Unternehmungen im Internet/Intranet-Umfeld durch den Einsatz des SDS flexibel angegangen werden konnte. Eine kurzfristige Änderung der Infrastruktur führte nicht zum Scheitern des Projektes, der Mehraufwand hielt sich in Grenzen, ohne die bis zu diesem Zeitpunkt entwickelten Teillösungen in Frage zu stellen. Der Code musste nicht umgeschrieben werden, die Entwicklungsteams für diese Lösungen mussten sich in keiner Weise umstellen oder die neue Infrastruktur bei der Fortführung ihrer Entwicklung bedenken.

In diesem Falle hätten alternative Lösungsansätze auf Grundlage von RMI oder mit direktem Zugriff der Servlets mittels JDBC auf die Intranet-Datenbank zur Neuentwicklung von weiten Teilen der Code-Basis geführt.

So konnte dem Anspruch der SDS-Plattform, kurze Entwicklungszeiten zu ermöglichen, Rechnung getragen werden. Die Lösung konnte fristgerecht in den Live-Betrieb überführt werden. Innerhalb der ersten zwei Wochen stellten über 3000 Lehrer mit der vorgestellten Lösung ihre Teilzeitanträge über das Internet.

9.2 Unternehmerbüro

Die IT-Strukturen in Behörden und Unternehmen sind hochgradig heterogen. Die Gründe dafür sind einleuchtend: IT-Strukturen wachsen und ändern sich. Im Wandel der Zeit werden neue Technologien eingesetzt, ohne die existierende Infrastruktur komplett ersetzen zu können. Investitionsschutz und Migrationskosten spielen dabei eine ausschlaggebende Rolle. Dabei werden die Anforderungen an die IT-Strukturen nicht geringer: Daten verschiedener Datenquellen müssen kombiniert werden, um Synergieeffekte ausnutzen zu können. Eine allgemein gültige Lösung dieser Problematik ist nicht möglich, jede Lösung muss an die individuelle IT-Struktur angepasst werden.

Doch so unterschiedlich die Anforderungen auch sein mögen, es gibt Gemeinsamkeiten, die in einer Integrationsplattform zusammengefasst werden können. Der Smart Data Server bietet sich als eine solche Plattform besonders an, da er einen transparenten Zugriff auf die Infrastruktur der Behörden oder des Unternehmens ermöglicht.

Im Rahmen eines Projektes mit einer kommunalen Stadtverwaltung wurde der SDS zur Verknüpfung von Daten aus verschiedenen Ämtern entwickelt[4]. Abbildung 9.4 zeigt das hierzu entwickelte Konzept.

Dem Amt „Unternehmerbüro“ sollten Daten dreier weiterer Ämter („Vermessungsamt“, „Ordnungsamt“, „Amt für Statistik und Stadtentwicklung“) verfügbar gemacht werden, um

[4] Roth (2000c)

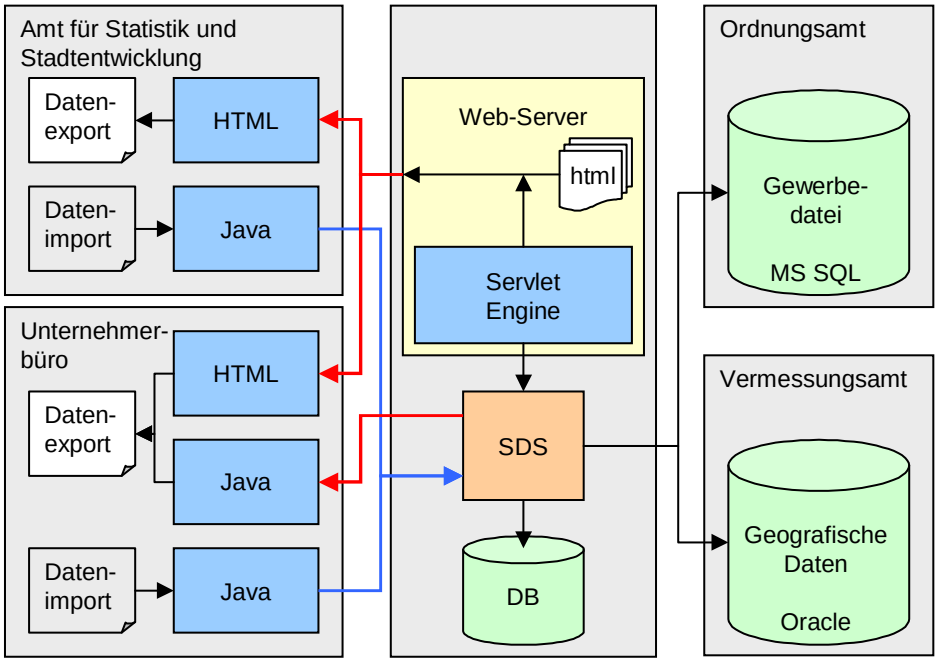

Abbildung 9.4: Kommunale Stadtverwaltung

durch deren Kombination Mehrwerte für das Amt zu erlangen. Die Daten des Amtes für Statistik und Stadtentwicklung lagen bis zu diesem Zeitpunkt nur unstrukturiert in nicht-elektronischer Form vor. Um sie nutzbar zu machen mussten Mechanismen geschaffen werden, diese Daten in eine neue Datenbank einzupflegen. Exportmechanismen sollten es diesem Amt ermöglichen, auf die neu eingespielten, historisierten Daten zurückzugreifen, um als Datenlieferant auch einen Nutzen zu gewinnen.

Das Vermessungsamt und das Ordnungsamt stellten Daten zur Verfügung, auf die nur lesend zugegriffen werden sollte. Die Kombination von Adressdaten des Ordnungsamtes mit geografischen Daten des Vermessungsamtes versprach die größten Synergieeffekte.

Aber auch das Unternehmerbüro selbst kann prinzipiell Datenlieferant sein: Zugriffe auf Daten können als Fallbeispiele für zukünftige Anfragen hinterlegt werden.

Mit dem SDS als Middle-Tier-Architektur war es möglich, eine auf Internet-Technologie basierende Struktur zu entwickeln, die den Zugriff auf Daten verschiedenster Ursprungsquellen transparent verfügbar macht und zwar über Standard-PCs mit Standard-Programmen (Internet-Browser wie z.B. Netscape oder Internet-Explorer).

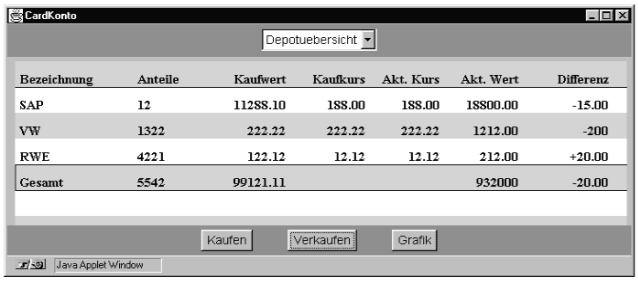

Abbildung 9.5: Depotübersicht

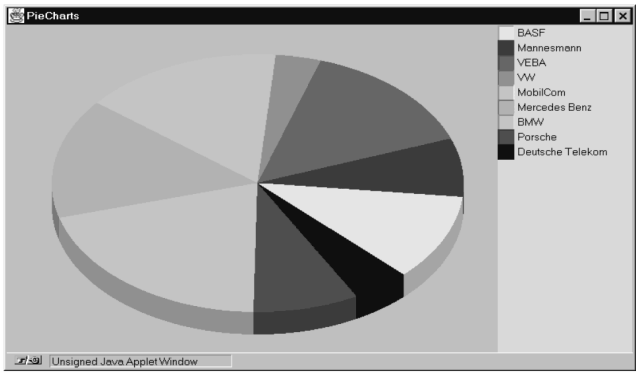

Abbildung 9.6: Kuchendiagramm

9.3 Portfolio-Management-System

Für eine luxemburgische Bank wurde mit Hilfe des Smart Data Server ein *Portfolio-Management-System* gebaut, mit dem Kunden der Bank Transaktionen von Wertpapieren über das Internet simulieren können, um hiervon eigene Entscheidungen (Ankauf, Verkauf) abhängig zu machen.

Aus Sicht des Kunden teilte sich die Anwendung in zwei Bereiche; zum einen in den Bereich der Portfolio-Verwaltung, zum anderen in den der Chart-Analyse.

In der Portfolio-Verwaltung ist der Kauf und Verkauf von Positionen möglich, ebenso die Darstellung des Depot-Bestandes in verschiedenen Darstellungsvarianten:

- In der *Depotübersicht* (siehe Abb. 9.5) werden die aktuell vorhandenen Positionen bewertet und bezüglich des Kaufdatums in Gewinn und Verlust dargestellt. Dabei wird neben den Kursschwankungen bei Positionen in ausländischer Währung auch die Veränderung der Wechselkurse berücksichtigt.

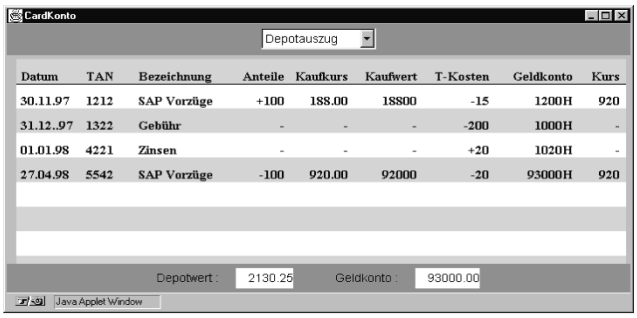

Abbildung 9.7: Depotauszug

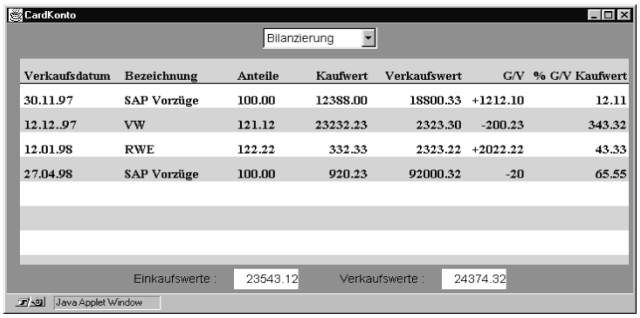

Abbildung 9.8: Bilanz

Der Anteil der einzelnen Positionen am Gesamtportfolio kann in einer weiteren Kuchengrafik abgerufen werden (siehe Abb. 9.6).

- Im *Depotauszug* (siehe Abb. 9.7) werden Veränderungen am Depot protokolliert, also Käufe, Verkäufe, Gebühren und Zinsen.
- Bei der *Bilanzübersicht* (siehe Abb. 9.8) werden die verkauften Positionen bezüglich Gewinn und Verlust bewertet.

Neben der Auflistung der Depotdaten können für alle im System verfügbaren Wertpapiere Charts dargestellt werden, also Grafiken, die den Kursverlauf eines Wertpapiers im Laufe der Zeit abbilden (siehe Abb. 9.9). Neben der reinen Darstellung der Kurse können weitere Indikatoren in den Chart eingeblendet werden. Indikatoren bieten Hilfestellung bei Kauf- und Verkaufsentscheidungen. Zusätzlich besteht die Möglichkeit, den Verlauf zweier Wertpapiere gegenüber zu stellen. In diesem Fall ist weniger der Preis eines Wertpapiers interessant als dessen prozentuale Veränderung.

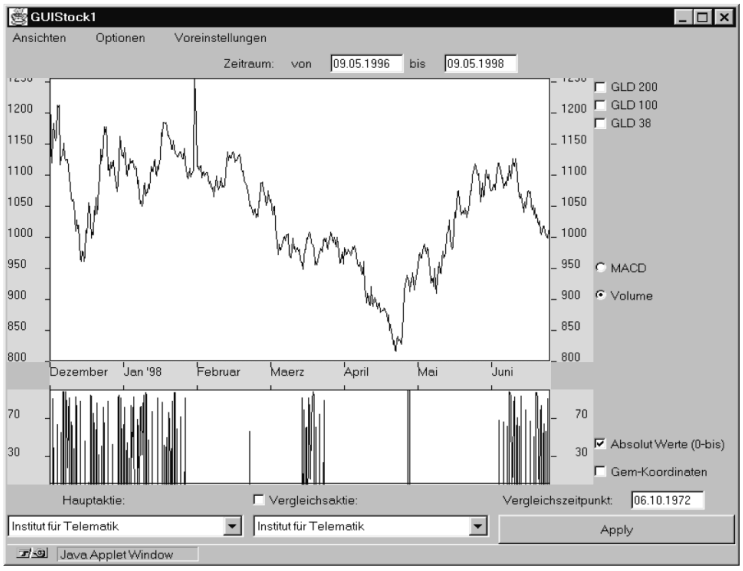

Abbildung 9.9: Aktienverlauf

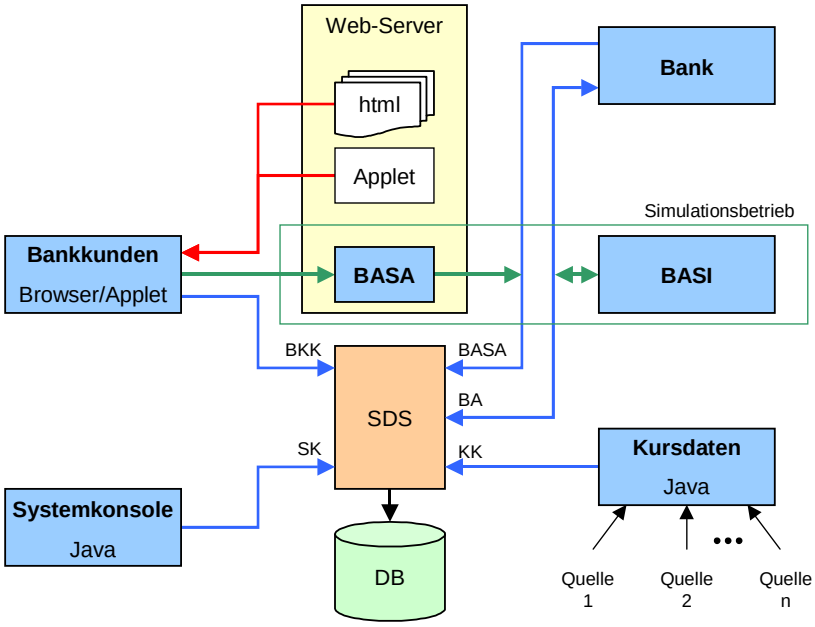

Abbildung 9.10: Portfolio Management System

Zur Umsetzung musste mehreren Randbedingungen Rechnung getragen werden. So lag zu Beginn keine gesicherte Kursdatenversorgung vor, die zur Bewertung von Depots unerlässlich ist. Auch sollte neben dem Realbetrieb ein Simulationsbetrieb möglich sein, in dem Teile der Bank-Infrastruktur durch Simulationseinheiten ersetzt werden.

Das Konzept ist in Abbildung 9.10 visualisiert. Im Zentrum dieser Lösung steht der Smart Data Server, der über fünf verschiedene Informationskanäle angesprochen werden kann. Jedem dieser Kanäle ist ein eigener Aufgabenbereich zugeteilt. Ein Informationskanal entspricht einem Port des Smart Data Servers, der mit unterschiedlichen Sicherheitsniveaus konfiguriert werden kann.

Die Informationskanäle werden wie folgt beschrieben:

BKK: Der *Bankkundenkanal* liefert die Depot- und Chart-Ansicht für die Bankkunden sowie frei verfügbare Daten für alle weiteren Internet-Benutzer. Der Zugriff erfolgt mittels eines Java-Applets.

BASA: Der *Bank-Systemadministrationskanal* dient der Neuanlage von Kundendepots, der Angabe von Kreditlimits, der Einstufung von Risikoklassen und liefert Protokolle des Smart Data Servers für die Bank.

BA: Über den *Bank-Administrationskanal* werden Wertpapier-An- und Verkaufstransaktionen durchgeführt.

KK: Der *Kurskanal* dient der Anlieferung von Kursdaten.

SK: Mittels der Systemkonsole wird der Smart Data Server administriert.

Im Simulationsbetrieb geschieht die Abwicklung einer Transaktion durch ein eigenes *Banksimulationsmodul (BASI)*, das neben Verzögerungszeiten bei An- und Verkauf auch Kursschwankungen zwischen Order und Transaktionsdurchführung realisiert. Die Einrichtung der Depots selbst, die Vergabe der Zugriffsberechtigungen und das Belegen des Kreditlimits kann der Benutzer im Simulationsbetrieb über eine HTML-Seite mit hinterlegten BASA-CGI-Skripten selbstständig durchführen.

Da eine bankseitige Kursdatenversorgung nicht vorlag, wurden für frei zugängliche Quellen (z.B. das Internet) Anwendungen geschrieben, die diese Quellen periodisch auslesen und zum SDS transportieren. Mit der vorhandenen Kursdaten-Einspielfunktion muss bei der Umstellung auf andere Quellen (z.B. Reuters oder Bloomberg) nur der Transformationsadapter zwischen Quelle und SDS neu geschrieben werden. Die Versorgung mit Kursdaten aus mehreren Quellen ist möglich.

Die Anforderungen an das Bankkunden-Applet war die Fähigkeit zur Visualisierung der vorhandenen Information bei minimaler Speicherplatzgröße. Das Applet wurde hierzu so konzipiert, dass es selbst kaum Logik besitzt und diese hauptsächlich auf der Serverseite implementiert ist.

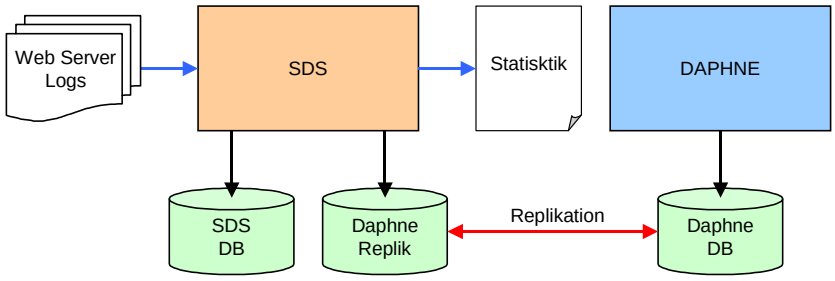

Abbildung 9.11: Log-Analyse

So werden z.B. bei der Abfrage von Indikatoren diese auf der Serverseite berechnet und nur die darstellbaren Werte[5] zum Applet übertragen. Dies macht auch Sinn, da einige Indikatoren einen größeren Zeitraum an Kursdaten zur Berechnung benötigen, als möglicherweise angefordert werden. So benötigt der 200-Tage-gleitende Durchschnitt (200GLD) die letzten 200 Kursdaten zur Berechnung des aktuellen Durchschnitts. Umfasst der angeforderte Zeitraum jedoch nur 100 Tage, so kann für keinen dieser Tage auf der Client-Seite ein 200GLD ermittelt werden. Der Server besitzt jedoch alle zur Berechnung notwendigen Daten und kann für die 100 Tage alle 200GLD berechnen.

Die Größe des Applets konnte auch durch das schlanke Kommunikationsprotokoll und wegen der einfachen Nachrichtenstruktur auf ein Minimum reduziert werden, so dass das Laden des Applets auch über langsame Modem-Verbindungen schnell möglich ist.

9.4 Web-Log-Analyse

Als Server zur Analyse von Web-Log konnte der Smart Data Server eingesetzt werden[6]. Die Besonderheit an dieser Analyseeinheit war die Spezialisierung auf Logs, die durch Web-Sites erzeugt wurden, die mittels des Online-Redaktionssystems „Daphne“[7] erstellt wurden.

Web-Sites auf der Basis von Daphne haben die Besonderheit, dass Dokumente nur Nummern als Dateinamen besitzen und alle Dokumente innerhalb eines einzigen Verzeichnisses liegen. Allein aus den Server-Logs ist es somit unmöglich, die Position eines Dokuments innerhalb der logischen Struktur der Website zu ermitteln.

Diese Information liegt jedoch noch im Redaktionssystem Daphne vor. Wird eine Log-Datei in den Smart Data Server eingespielt, so überträgt dieser im ersten Schritt alle Infor-

[5] bzgl. der Auflösung des Applets
[6] Haffner u. a. (2000a, 2001)
[7] Daphne ist ein Online-Redaktionssystem, dass am Institut für Telematik (Trier) entwickelt wurde (Heuer, 2002).

mationen in verdichteter Form in eine SDS-eigene Datenbank, z.B. mit Browser-Hersteller, IP-Nummer oder der Internetname des Rechners, der das Dokument abgerufen hat.

In einem zweiten Analyseschritt wird nun die Log-Daten der Datenbank mit den Informationen des Redaktionssystems abgeglichen. Dies geschieht nicht auf der Produktivdatenbank, sondern auf einer Replik, um das Hauptsystem zu entlasten.

Aus der Daphne-Datenbank kann aus der Dokumentennummer dessen Ressort (Kategorie) ermittelt werden, also der logische Zusammenhang, in dem sich ein Dokument befindet. Das Ergebnis der Web-Log-Analyse liefert dann aussagekräftige Informationen, z.B. wie oft bestimmte Ressorts/Kategorien besucht oder welche Dokumente überhaupt nicht besucht wurden. Mit üblichen Log-Analyse-Tools hätten die vom Betreiber der Web-Site gewünschten Informationen nicht ermittelt werden können.

Gegenüber einer Neuentwicklung einer Anwendung hielt sich der Entwicklungsaufwand der benötigten Module für den Smart Data Server in Grenzen. Die vorhandene Server-Infrastruktur übernimmt viele Aspekte (z.B. Fehlerbehandlung, Datenbankzugriffe), die bei einer Neuentwicklung stets neu betrachtet werden müssen. Der Zeitpunkt, ab dem sich der Entwickler um das eigentliche Problem der Anwendung kümmern kann, liegt früher. Aus diesem Grunde ist die Gesamtentwicklungszeit kürzer.

10 Laufzeittests

Trois mille six cents fois par heure, la Seconde
Chuchote: „Souviens-toi!“–Rapide, avec sa voix
D'insecte, Maintenant dit: Je sais Autrefois,
Et j'ai pompé ta vie avec ma trompe immonde![1]

L'Horloge, Les Fleurs du mal
CHARLES PIERRE BAUDELAIRE

Nachdem der Smart Data Server bezüglich seines Aufbaus beschrieben wurde und an einigen praktischen Beispielen seine Einsatzfähigkeit zeigen konnte, schließen die Erläuterungen mit Laufzeittests ab. Hierbei muss sich der Server mit anderen Technologien messen. In Abschnitt 10.1 wird zunächst der Testaufbau motiviert, gefolgt von den Ergebnissen in Abschnitt 10.2. Das Kapitel endet mit einigen abschließenden Bemerkungen zu den Ergebnissen in Abschnitt 10.3.

10.1 Test-Aufbau

Die Geschwindigkeit von Servern wird durch verschiedene Faktoren beeinflusst. Im Falle von Middleware-Architekturen ist wichtig, in welcher Zeit der Server Anfragen beantwortet. Bei der Beantwortung einer Anfrage spielen drei Zeiten eine Rolle:

- Die Zeit für die Erstellung der Anfrage-Struktur und deren Transport an den Server.
- Die Zeit für die Berechnung der Antwort.
- Die Zeit für die Erstellung der Antwort-Struktur und deren Transport zum Client.

Von allen drei Zeiten ist die Zeit für die Berechnung der Antwort diejenige, die am wenigsten bestimmt werden kann, da diese stark von den Eingabeparametern abhängt. Wenn andere Systeme in der Antworterstellung involviert sind, hängt die Zeit auch von der Performance dieser Systeme ab. Testszenarien hierfür zu entwickeln, ist schwer möglich. Anderseits kann jedoch auch davon ausgegangen werden, dass identische Berechnungen auf

[1] *// Dreitausend- und sechshundertmal tickt die Sekunde // Allstündlich: Denk daran! - In flinkem Einerlei // Zirpt der Insektenlaut des Jetzt: Ich bin vorbei! // Mein Rüssel pumpte dir dein Leben aus der Wunde! //* (Übersetzung: Sigmar Löffler, Insel Verlag)

demselben System unter verschiedenen Middleware-Plattformen auch ähnliche Laufzeiten besitzen, wenn sie in der gleichen Programmiersprache geschrieben wurden. Einzig die Speicherauslastung des Systems oder die Anzahl parallel ausgeführter Threads und Prozesse hat hier einen Einfluss auf die Berechnungszeit. Auch in diesem Falle ist es wiederum schwierig, gleiche Lastszenarien aufzubauen und auszuwerten.

Dieses Kapitel beschränkt sich deshalb auf Tests der Geschwindigkeit der Anfrage- und Antwortübertragung. Gemessen werden das Verpacken der Anfrage auf der Client-Seite, die Übertragung der Anfrage vom Client zum Server, das Entpacken der Anfrage auf der Server-Seite, das Verpacken der Antwort durch den Server, die Übertragung der Antwort zu Client sowie das Entpacken der Antwort auf der Seite des Client. Die entfernte Funktion führt keine weitergehenden Berechnungen durch, einzig die Aufruf-Parameter müssen soweit vorliegen, dass eine Berechnung in der Funktion möglich ist. Zur Übertragung von Daten vom Server zum Client müssen diese zum Zeitpunkt der Funktionsausführung vorbereitet vorliegen.

Um aussagekräftige Ergebnisse zu erhalten, werden im ersten Testfall ausschließlich Daten zum Server übertragen, ohne dass Daten in der Antwort geliefert werden (*C2S*[2]). Im zweiten Fall werden keine Daten zum Server übertragen, der dafür aber vorbereiteten Daten zum Client überträgt (*S2C*[3]).

Der Test deckt das Sitzungsprotokoll (siehe Abs. 5.1) und das RPC-Nachrichtenformat (siehe Abs. 5.2), mit dem die Anfragen kodiert und transportiert werden, ab. Die Effektivität dieser Einheiten wird als Ganzes bewertet, eine Unterscheidung zwischen der Effizienz des Nachrichtenformats sowie der Kodier- und Dekodierfähigkeit wird nicht durchgeführt.

Hauptsächlich werden die übertragenen Daten durch zwei Parameter bestimmt: Einerseits durch die Größe der Daten, andererseits durch die Komplexität der Datenstruktur. Da auch hier wiederum beliebige Datenstrukturen denkbar sind, wurden die beiden Parameter in einer einheitlichen Form abgebildet.

Da die Komplexität möglicher Datenstruktur beliebig ist, wurde dieser Test allein auf Übertragung von Listen beschränkt. Diese Datenstruktur bildet zwar nur einen Ausschnitt von möglichen Strukturen ab, die Konsequenzen aus den Laufzeittests lassen sich jedoch in Grenzen auf andere Datenenstrukturen übertragen, wenn man deren Gesamtgröße und Komplexität kennt.

Zwischen Client und Server werden im Test somit nur Listen übertragen. Veränderlich ist die Anzahl der Elemente in der Liste sowie die Größe eines Elements der Liste (ein String mit zufälligen Zeichen der festgelegten Länge). Alle Elemente der Liste haben innerhalb der Datenstruktur die gleiche Größe. Beispielsweise können 32 KBytes Daten als Liste mit 512 Elementen mit jeweils 64 Bytes oder als Liste mit 128 Elementen je 256 Bytes übertragen werden.

[2] Client to Server
[3] Server to Client

Neben dem Smart Data Server wurde die Java-RMI-Technologie getestet (siehe Abs. 2.3.1). Der zweite Konkurrent ist die Apache SOAP-Implementierung[4], die als Servlet innerhalb einer Tomcat-Servlet-Engine[5] läuft (siehe Abs. 2.7). Es liegen mit RMI eine native Lösung und mit SOAP (im Kontext von WebServices) eine zur Zeit sehr beachtete Technologie vor. Beide Lösungen basieren auf Java, was Chancengleichheit garantiert.

Da sich alle drei Technologien bezüglich der Aufrufart unterscheiden, es also nicht möglich ist, als entfernte Funktion die tatsächlich identische Funktion verwenden zu können, musste bei der Entwicklung der Testfunktionen besonderer Wert darauf gelegt werden, keine versehentlichen Vor- oder Nachteile zu implementieren. So müssen beispielsweise alle Elemente der Liste unterschiedliche String-Elemente beinhalten. Würden zwei Elemente auf dasselbe String-Objekt verweisen, würde RMI dies erkennen und nur eine Repräsentation des Strings übermitteln.

Im Falle der Datenlieferung vom Server zum Client musste neben der eigentlichen entfernten Funktion eine Initialisierungsfunktion im Server implementiert werden, mit der die Datenstruktur im Server vorbereitet werden kann. Diese Vorbereitung findet außerhalb der Zeitmessungen statt.

Vor Beginn des Testlaufs werden alle Kombinationen von Elementengröße und Elementenanzahl im Vektor zufällig verwürfelt, um die Server im Laufe des Tests mit immer wechselnden Komplexitäten zu belasten. Für jede dieser Kombinationen wird für SOAP, SDS und RMI jeweils ein Test des Datentransfers zum Server (C2S) und anschließend ein Test des Datentransfers vom Server (S2C) aufgeführt. Da es sich bei den entfernten Funktionsaufrufen um Laufzeiten im Millisekundenbereich handelt, werden jeweils zehn Anfragen zu einer Zeitmessung zusammengefasst.

Aus den Zeitmessungen werden zunächst unter Zuhilfenahme des Ausreißertests von Grubbs[6] Störungen der Messreihe (Festplattenzugriffe, o.ä.) entfernt. Aus den verbliebenen Werten wird der Median (*ME*) als Ergebnis der Zeitmessung gewertet. Zusätzlich wird der Variationskoeffizient (*VK*) bestimmt. Er gibt an, wieviel Prozent vom arithmetischen Mittel die Standardabweichung beträgt. Er stellt einen besseren Wert für die Streuung der Daten dar als die Standardabweichung, da die Laufzeiten für die unterschiedlichen Messungen stark varriieren.

Für die Elementgröße und die Anzahl der Elemente wird eine Reihe von Zahlen zwischen 0 und 512 herangezogen[7]. Als größtes Datenpaket werden 512 Elemente à 512 Bytes übertragen. Dies entspricht 256 KBytes.

Bei der Durchführung des Testprogramms führt Just-In-Time-Compiler von Java anfangs zu größeren Laufzeitmessungen. Diese Schwankungen spielen durch die Gesamtlaufzeit des Tests keine signifikante Rolle.

[4] Version 2.3.1
[5] Version 4.0.4
[6] Signifikanzniveau P=95%
[7] 0, 8, 16, 24, 32, 48, 64, 96, 128, 192, 256, 384, 512

Bei den Testsystemen handelt es sich einerseits um einen Pentium III mit 850 Mhz, 256 MBytes Hauptspeicher, Java 1.4.2 und Windows 2000 (SP4) (System 1 = *S1*), andererseits um einen Pentium III mit 600 Mhz, 256 MBytes Hauptspeicher, Java 1.4.2 und Windows XP SP1 (System 2 = *S2*). Client und Server laufen auf demselben Rechner, womit Netzschwankungen ausgeschaltet sind. Die Anfragen durchlaufen trotzdem vollständig den IP-Stack. Um eine gleichbleibende Testumgebung zu gewährleisten, wurden alle störenden Prozesse des Servers beendet (Virenscanner, Firewalls, etc.). Der Netzwerkanschluss wurde vom LAN getrennt. Alle Serverprozesse laufen mit der Priorität „Höher als Normal“.

10.2 Test-Ergebnisse

Da die Testreihe pro Durchführung eines Testzyklusses (alle Verfahren, beide Richtungen) ca. 15 Minuten benötigt, wurden bei System 1 ca. 250 Zyklen durchgeführt, bei System 2 ca. 50 Zyklen. Die geringere Anzahl der Zyklen bei System 2 ergibt sich dadurch, dass die Tests dort nur nachts durchgeführt werden konnten.

Die Ergebnisse der Messreihen (Median und Variationskoefizient) sind im Anhang F aufgeführt. Aus Platzgründen wurde in den Tabellen die englische Bezeichnung „Item“ für „Element“ gewählt.

Die folgenden Erläuterungen beziehen sich zuerst auf das System 1. Nur bei wesentlichen Unterschieden zu System 2 werden diese gesondert aufgeführt.

10.2.1 RMI

Tabelle F.1 zeigt die Messergebnisse von RMI beim Datentransfer vom Client zum Server. Deutlich erkennbar ist, dass RMI eine Mindestlaufzeit von 30ms aufweist. Bemerkenswert ist, dass sich dieser Wert auch bei komplexeren Anfragen oder größeren Elementen nicht verändert. Erst bei größeren Bytes/Item-Items-Kombinationen steigt die Laufzeit an. Verständlicherweise steigt sie bei Zunahme der Komplexität stärker an als bei entsprechender Zunahme der Elementgröße. Der Grund hierfür ist, dass ein weiteres Element in der Liste neben den Elementdaten zusätzliche Bytes an Strukturinformationen verursacht.

Die Abbildungen 10.1 und 10.2 visualisieren einen Ausschnitt der Messergebnisse. Im ersten Fall sind die Bytes/Item, im zweiten Fall die Anzahl der Items konstant auf 512 gesetzt. Während das Wachstum bei konstanter Bytes/Item mit der Item-Anzahl langsamer steigt, steigt das Wachstum bei fester Item-Anzahl mit steigender Bytes pro Item.

Die Höhe der Variationskoefizienten zwischen 6% und 17% zeigt jedoch, dass die Messwerte teilweise sehr stark um den Mittelwert variieren können. Eine Aussage, dass der Zusammenhang nicht linear ist, kann deshalb nur mit mehr und größeren Stützstellen getroffen werden.

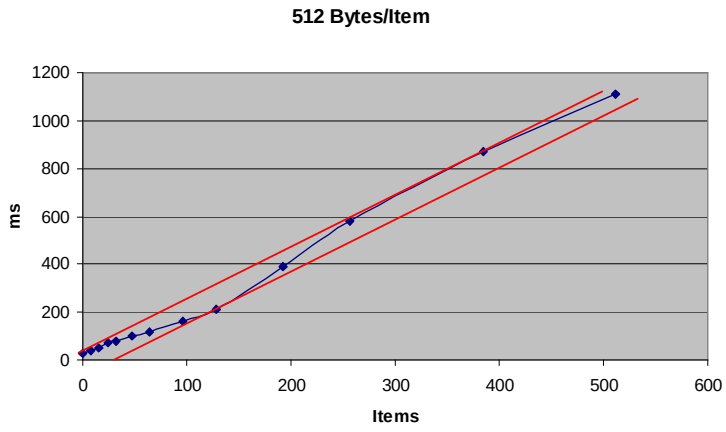

Abbildung 10.1: ME(RMI,C2S,S1), Bytes/Item=512

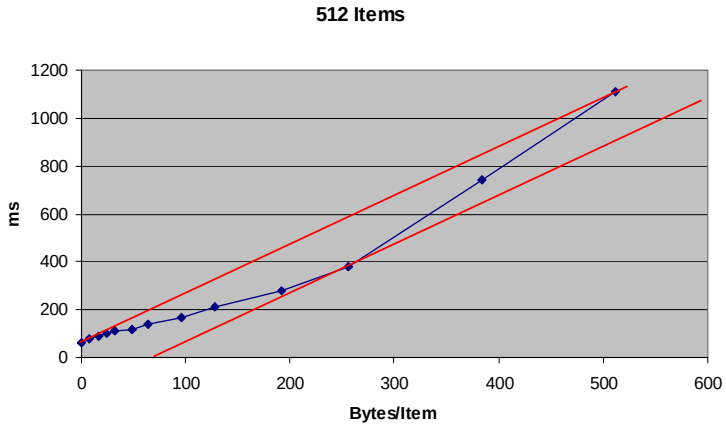

Abbildung 10.2: ME(RMI,C2S,S1), Items=512

Interessant in diesem Zusammenhang ist die Verteilung der Variationskoefizienten (siehe Abb. 10.3), die nicht über die gesamte Matrix gleich ist, sondern ein nicht erklärbares „Tal“ entlang einer Diagonalen 512/24 zu 24/512 besitzt.

Beim Datentransfer zwischen Client zu Server und umgekehrt gibt es keine signifikanten Unterschiede.

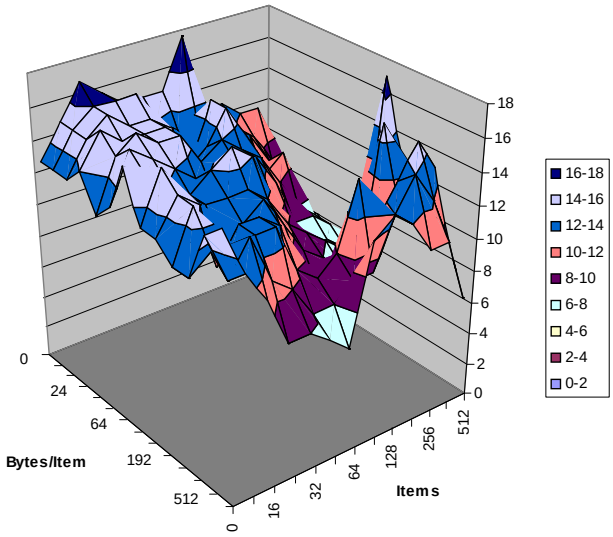

Abbildung 10.3: VK(RMI,C2S,S1)

10.2.2 SOAP

SOAP ist in der Ausführung wesentlich langsamer als RMI (Faktor 10). Die Grundlaufzeit beträgt hier schon 340 Millisekunden.

Der Variationskoefizient ist zunächst sehr gering und steigt erst ab einer höheren Komplexität stark an (siehe Abb. 10.4). Zu einem ähnlichen Zeitpunkt steigt auch die Laufzeit beim Empfang von Daten gegenüber der Sendezeit (siehe Abb. 10.5). Der Grund hierfür wird in der Prozessverwaltung der Windows-Betriebssysteme gesehen. Auf der Server-Seite teilen sich mehrere Threads den gleichen Prozess, der auf der Client-Seite nur von einem Thread in Anspruch genommen wird. Das Verpacken der Anfrage erfordert mehr Aufwand, als das Entpacken, sodass ein vermehrtes Unterbrechen des verpackenden Threads zu einer erhöhten Gesamtlaufzeit führt.

10.2.3 SDS

Die Messergebnisse von SDS sind vergleichbar mit den Ergebnissen von SOAP. Dies ist auch nicht weiter verwunderlich, da beide mit einem Nachrichtenformat auf der Basis von XML die Datenstrukturen verpacken. Allerdings benötigt der SDS bei geringem Datenvolumen wesentlich weniger Zeit zur Funktionsausführung (Faktor 3.4). Hingegen steigt die Laufzeit des SDS bei Zunahme der Anzahl der Listenelemente stärker an als SOAP. Dies führt dazu, dass SOAP bei einer bestimmten Anfragekomplexitäten schneller ist als der SDS

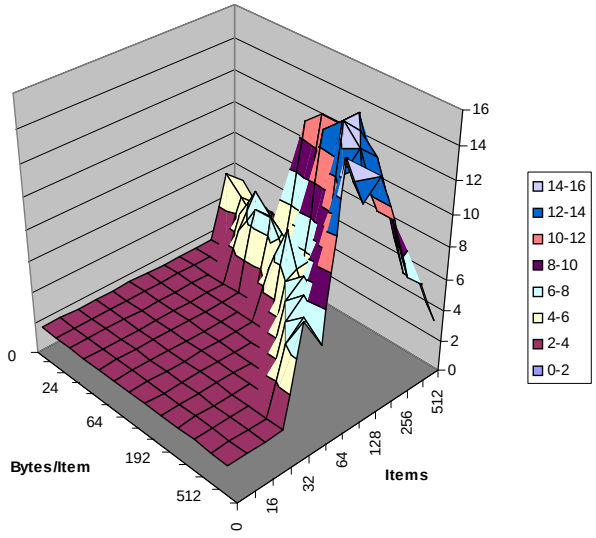

Abbildung 10.4: VK(SOAP,C2S,S1)

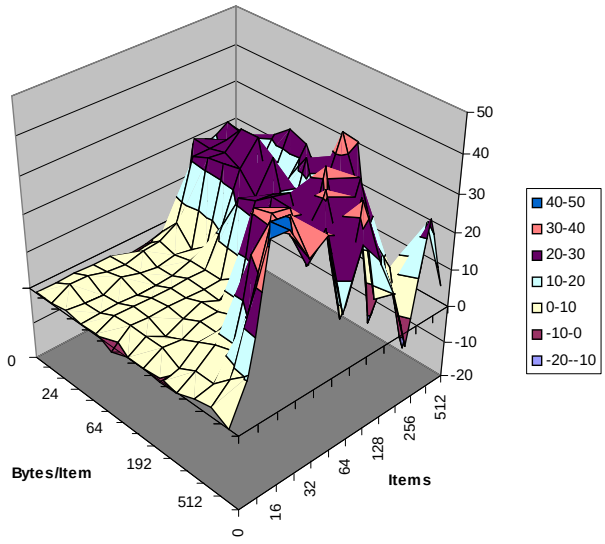

Abbildung 10.5: ME(SOAP,S2C,S1)/ME(SOAP,C2S,S1)

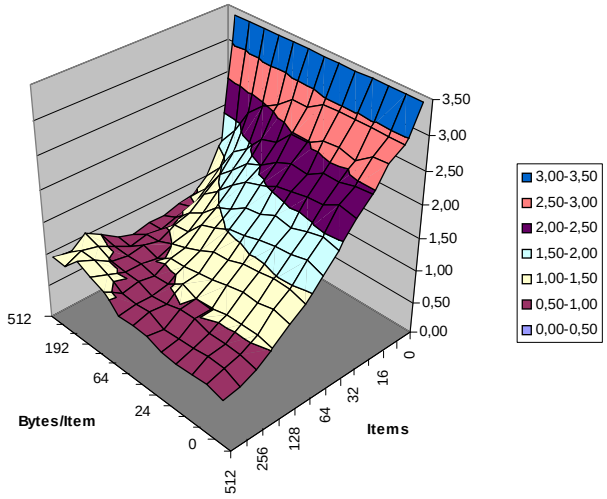

Abbildung 10.6: ME(SOAP,C2S,S1)/ME(SDS,C2S,S1)

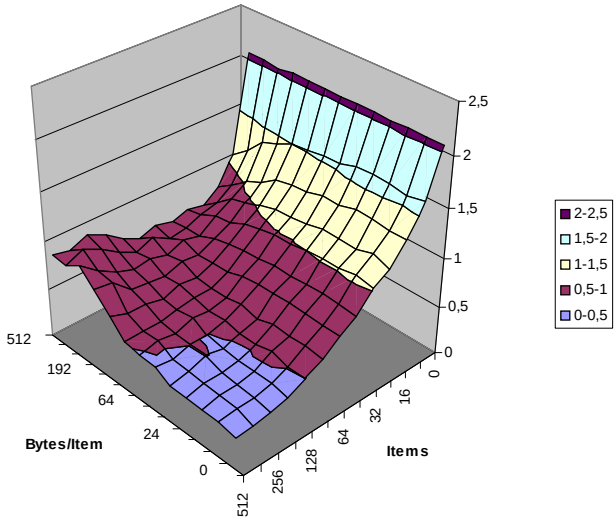

Abbildung 10.7: ME(SOAP,C2S,S2)/ME(SDS,C2S,S2)

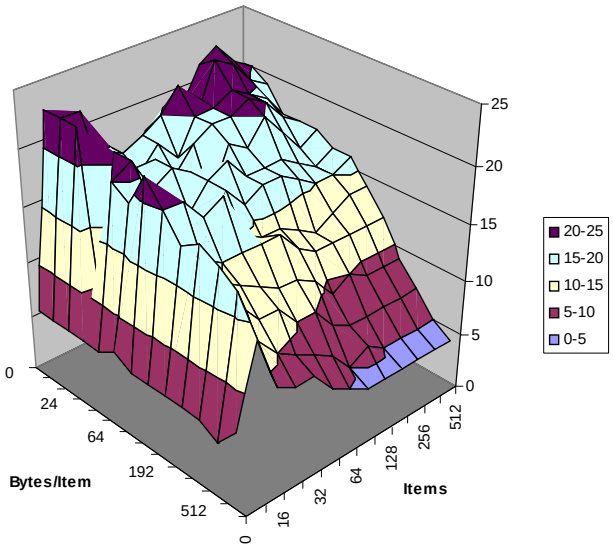

Abbildung 10.8: VK(SDS,C2S,S1)

(siehe Abb. 10.6). Allerdings relativiert sich dieses Phänomen bei sehr großen Listen und sehr großen Datenelementen wieder, sodass SOAP und SDS dann nahezu gleich schnell sind. System 2 hat ein ähnliches Verhalten wie System 1, allerdings bleibt der SDS auch bei großen Datenstrukturen gegenüber SOAP leicht langsamer.

Die größten Variationskoefizienten gegenüber SOAP und RMI besitzt der SDS (siehe Abb. 10.8). Sie nehmen mit steigender Komplexität der Datenstrukturen oder Laufzeit jedoch ab. Als Beispiel für die Verteilung der Laufzeitenhäufigkeit bezüglich einer festen Komplexität ist in Abbildung 10.9 die Verteilung für die Kombination 512 Bytes/512 Items dargestellt.

Die größten Unterschiede zwischen System 1 und System 2 gibt es im Verhältnis der Laufzeiten zwischen dem Datentransfer vom Client zum Server und vom Server zu Client. Während im System 1 die Laufzeiten „Client zum Server“ im Allgemeinen schneller sind als umgekehrt, ist dies in System 2 genau der andere Fall. Hier sind die Laufzeiten „Server zu Client“ besser (Siehe Abbildungen 10.10 und 10.11)[8].

[8]Die Betrachtungswinkel sind aus Gründen der Übersichtlichkeit unterschiedlich.

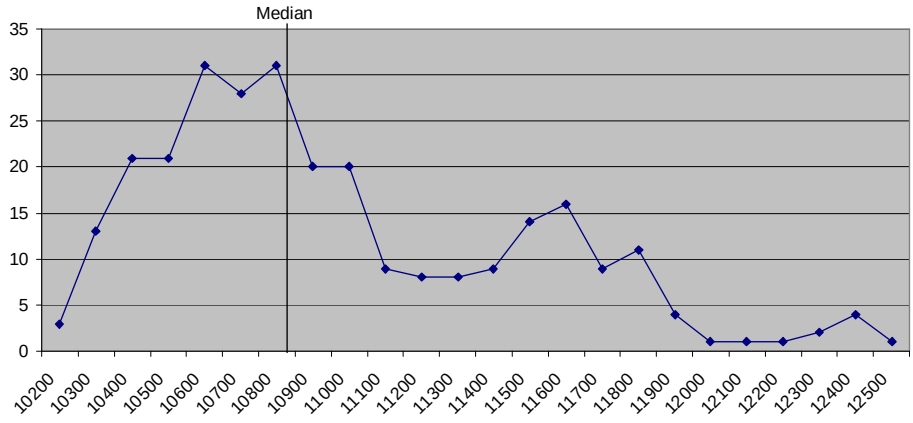

Abbildung 10.9: Häufigkeiten der Laufzeiten (SDS,C2S,S1,512,512)

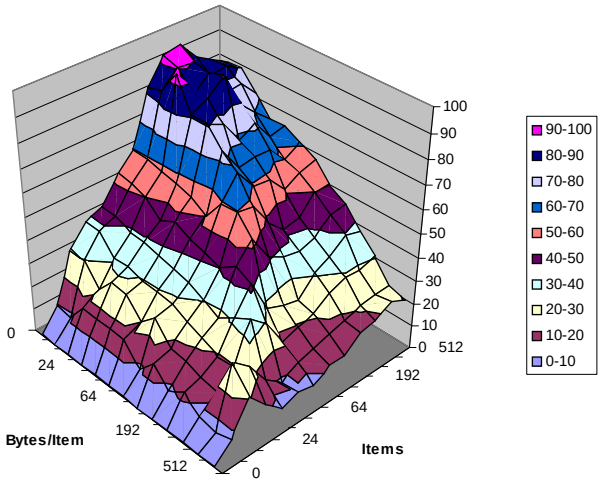

Abbildung 10.10: ME(SDS,S2C,S1)/ME(SDS,C2S,S1)

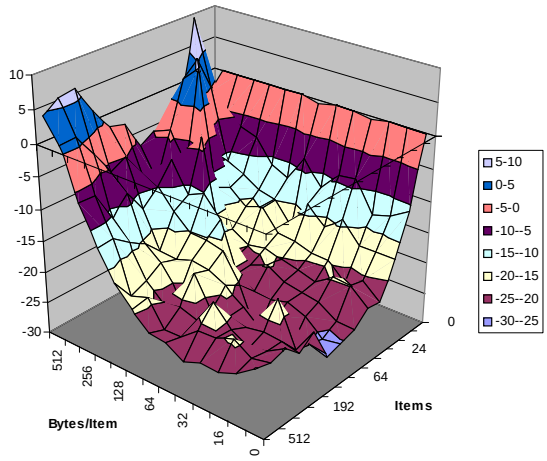

Abbildung 10.11: ME(SDS,S2C,S2)/ME(SDS,C2S,S2)

10.3 Bemerkungen

Bei der Durchführung der Tests wurde auf Praxisbezug geachtet. Obwohl ein Server in einer realen Einsatzumgebgung wesentlich mehr Prozesse gleichzeitig bearbeiten muss als in dem Testszenario, wurden die Messergebnisse teilweise sehr stark durch das System beeinflusst. Der SDS zeigte sich dabei am anfälligsten gegenüber Störungen, was sich in den hohen Variationskoefizienten und der starken Unterschiede zwischen System 1 und System 2 widerspiegelt. Einer der Gründe hierfür liegt in der Tatsache, dass mit dem SDS eine gesamte Server-Plattform gestartet wird und mehrere Threads im Hintergrund laufen, die zur Aufrechterhaltung des Systems benötigt werden.

Da dies bei RMI nicht der Fall ist, ist dies auch einer der Gründe für die wesentlich schnellere Abarbeitung von Anfragen. Der stärkere ist jedoch in dem Nachrichtenformat begründet, das nicht textbasiert ist und deshalb weniger Overhead benötigt.

Dies erklärt auch die ähnlichen Laufzeiten von SDS und SOAP. Beide verpacken die Datenstrukturen mit XML. SOAP verwendet hierbei die Standard-Pakete, der SDS verwendet eigene Klassen, die nur die definierte XML-Sprache beherrschen und im Umfang wesentlich kleiner sind.

Allgemein führen die Störungen, die durch die Betriebssysteme verursacht werden, dazu, dass keine eindeutigen Korrelationen zwischen Datenstrukturkomplexität und Laufzeiten hergestellt werden können. Unklar bleibt, ob ein linearer Zusammenhang zwischen Item-Anzahl und Laufzeit sowie zwischen Bytes/Item und Laufzeit besteht oder ein exponentieller oder anderer Zusammenhang. Die vorliegenden Messwerte liefern hierzu kein eindeutiges Ergebnis.

11 Zusammenfassung und Ausblick

I'm sorry I took your time
I am the poem that doesn't rhyme
Just turn back a page
I'll waste away, I'll waste away
I'll waste away, I'll waste away
I'll waste away, I'll waste away

Goodbye, Elton John - Madman Across The Water
BERNIE TAUPIN

Das Ziel der vorliegenden Arbeit war die Entwicklung von Konzepten und Techniken für Middlewaren. Middleware-Architekturen sind komplexe und vielschichtige Strukturen, die viele Teilaspekte umfassen, sei es die Kommunikation zwischen Client und Middleware, die interne Struktur oder innere Abläufe. Mit dem Smart Data Server wurde eine Middleware-Architektur entwickelt, die viele Konzepte in den unterschiedlichen Teilbereichen umgesetzt hat.

Bevor diese Konzepte motiviert werden konnten, wurden im ersten Teil der Arbeit zunächst in zwei Kapiteln aktuelle Technologien und die Forschung im Middleware-Umfeld dargestellt. Das erste Kapitel hatte den Schwerpunkt in der Betrachtung der entfernten Funktionsaufrufe, Objektzugriffe und Komponententechnologien, im zweiten lag der Fokus auf den inneren Strukturen und Datenflüssen, wichtiger Forschungsplattformen auf diesem Gebiet und angrenzenden Technologien.

Aus den Erkenntnissen dieses Teiles konnten Probleme und Lücken an bestehenden Systemen herausgearbeitet werden und im zweiten Teil der Arbeit als Ziele für eine Middleware-Plattform abgeleitet werden. In den folgenden Kapiteln wurden dann Konzepte entwickelt, die im Smart Data Server umgesetzt wurden.

Einige der definierten Ziele hatten den RPC-Mechanismus im Fokus. Einerseits sollte ein einfacher Mechanismus auf der Basis von entfernten Funktionsaufrufen geschaffen werden, anderseits sollte die Möglichkeit zur Abarbeitung von streamingbasierten Anfragen und Antworten möglich sein.

Hierzu wurde ein spezielles streamingfähiges Nachrichtenformat entwickelt, mit dem streamingbasierte Anfragen und Antworten unter Zuhilfenahme eines neu entwickelten Protokolls zwischen Client und Server ausgetauscht werden können.

In dem folgenden Kapitel wurde beschrieben, welche serverinternen Aktivitäten beim Empfang einer Anfrage involviert sind, insbesondere die Sitzungsbehandlung und die Au-

torisation/Authentifikation. Ein weiteres Ziel war das Auffinden der angesprochenen Module und Funktionen: Es wurde gezeigt, wie durch einfache Mechanismen die Weiterleitung von Anfragen an einen zweiten Smart Data Server möglich ist. Der Aufbau von Netzwerken von Smart Data Server ist somit kein Problem. Mit dem inneren Aufbau des Servers wurde ein weiteres nützliches Merkmal des Smart Data Servers beschrieben: serverinterne, zeitgesteuerte Anfragen.

Als Middleware-Architektur, die auch in der Praxis bestehen muss, wird eine Infrastruktur erwartet, auf die Dienste im Server zugreifen können. Dies wurde in einem weiteren Kapitel erläutert, wobei drei Basisdienste beschrieben wurden: Das Logging-Modul zur Protokollierung von Meldungen und Fehlern, das Datenbank-Modul für den transparenten Zugriff auf heterogene Datenbanklandschaften, sowie das eMail-Modul.

Eine der Kern-Technologien des Smart Data Servers wurde mit der Vorstellung der Workflow-Programme eingeführt. Hier geht es um die Optimierung der serverinternen Abläufe und Datenflüsse, insbesondere im Hinblick auf die Streamingfähigkeit der entfernten Funktionsaufrufe. Neben der Beschreibung des statischen Workflow-Netzes wurde gezeigt, wie mit diesen Strukturen Datenpipelines aufgebaut werden können. Damit ist der Aufbau einer flexiblen und erweiterbaren Server-Struktur mit maximal unabhängigen Server-Komponenten möglich. Das Kapitel schloss mit der Umsetzung dieser Technologie im Smart Data Server ab.

Im nächsten Teil der Arbeit ging es um praktische Aspekte des Smart Data Servers. In Fallstudien aus durchgeführten Kundenprojekten konnte die Praxistauglichkeit der Architektur gezeigt werden. Insbesondere die Anpassungsfähigkeit bei der Umsetzung von Lösungen an neue Randbedingungen konnte beispielhaft dokumentiert werden. In Laufzeittests konnte gezeigt werden, dass der RPC-Mechanismus zumindest hinsichtlich einer anderen aktuellen Technologie vergleichbare Performance besitzt und teilweise sogar bessere Laufzeiten aufweisen kann.

Ausblick: Mit dem Smart Data Server wurde eine Middleware-Architektur entwickelt, in der alle zuvor gesteckten Ziele in Konzepten und Lösungen umgesetzt wurden. Es bleibt dennoch noch Raum für zukünftige Erweiterungen und Optimierungen.

Ein Verbesserungsmöglichkeit stellt die Umsetzung des beschriebenen RPC-Workflow-Programms in der erweiterten Ausbaustufe mit Load Balancing dar. Daneben wird die Einbindung einer erweiterten Datenbank-Bibliothek angestrebt, mit der die herstellerunabhängige Erstellung von Datenbankabfragen, auch über Datenbank-Grenzen hinweg, möglich sein wird.

Die in der Arbeit beschriebenen Konzepte haben Potenzial für zukünftige Weiterentwicklungen. Besonders interessant ist der Ansatz, Workflow-Programme auch im Stub des Clients zu verwenden. Wenn die Workflow-Programme des SDS zum Ansprechen anderer Smart Data Server implementiert sind, um beispielsweise Anfrageweiterleitungen zu ermöglichen, können die beteiligten Workflow-Module auch in einem Workflow-Programm

im Client verwendet werden. Daneben ergeben sich völlig neue Möglichkeiten in der Client-Server-Kommunikation. Beispielsweise könnten Load-Balancing-Mechanismen des Servers in einer vereinfachten Version schon im Client vorhanden sein. Der Server würde auf Anfrage alternative Server zur Anfragebearbeitung nennen.

Die Idee, den Workflow-Programm-Mechanismus des Servers auch im Client einzubauen, ist wegen seiner geringen Größe und seines einfachen Aufbaus ohne Probleme möglich. Das tatsächliche Potenzial dieser Idee ist jedoch erst nach weiteren Forschungsmaßnahmen abschätzbar. Noch einen Schritt weiter geht die Idee der verteilten Workflow-Programme, also der Workflow-Programme, die über die Grenzen eines Smart Data Server hinaus über mehrere SDS verteilt sind. Auch hier sind die Möglichkeiten noch nicht vollständig ausgelotet.

Zur Zeit ist die Anpassung von Workflow-Programmen noch ein Vorgang, der außerhalb der Servers durch Editieren einer Konfigurationsdatei durchgeführt werden muss. Entsprechend der reflektiven Middlewaren ist auch eine Veränderung der Server-Struktur zur Laufzeit denkbar. Ein solcher Mechanismus könnte ohne Probleme so gestaltet werden, dass Veränderungen keine aktuell ablaufenden Workflow-Programme betreffen. Was fehlt, sind Konzepte, auf welche Art, beispielsweise ein Client Einfluss auf die Server-Struktur nehmen kann und darf.

Diese Beispiele zeigen, dass der Bereich der Middlewaren noch lange nicht ausgereizt ist und noch viel Spielraum für zukünftige Forschung besteht.

Zum Abschluss dieser Arbeit noch ein Zitat von Laotse[1]:

Beginnen können ist Stärke, vollenden können ist Kraft.

So sei es!

[1] Chinesischer Philosoph, Begründer des Taoismus.

Teil IV

Anhang

A XML-RCP

```
<value> <i4> 343 </i4> </value>
<value> <struct> ... </struct> </value>
<value> <array> ... </array> </value>
```

Code A.1: XML-RPC Value

```
<struct>
    <member>
        <name> key1 </name>
        <value> <i4> 123 </i4> </value>
    </member>
    <member>
        <name> key2 </name>
        <value> <array> ... </array> </value>
    </member>
</struct>
```

Code A.2: XML-RPC Struktur

```
<array>
    <data>
        <value> <double> 123.34 </double> </value>
        <value> <struct> ... </struct> </value>
    </data>
</array>
```

Code A.3: XML-RPC Liste

```
POST /RPC HTTP/1.0
Content-Type: text/xml
Content-Length: 234

<?xml version="1.0" ?>
<methodCall>
    <methodName> example.getValue </methodName>
    <params>
        <param> <value> <i4> 43 </i4> </value> </param>
        <param> <value> <array> ... </array> </value> </param>
    </params>
</methodCall>
```

Code A.4: XML-RPC Methodenaufruf

```
HTTP/1.0 200 OK
Connection: close
Content-Type: text/xml
Content-Length: 434

<?xml version="1.0" ?>
<methodResponse>
    <params>
        <param>
            <value>
                <String> yes </string>
            </value>
        </param>
        <param> <value> <struct> ... </struct> </value> </param>
    </params>
</methodResponse>
```

Code A.5: XML-RPC Antwort

```
HTTP/1.0 200 OK
Connection: close
Content-Type: text/xml
Content-Length: 434

<?xml version="1.0" ?>
<methodResponse>
    <fault>
        <value>
            <struct>
                <memeber>
                    <name> faultCode </name>
                    <value> <int> 4 </int> </value>
                </member>
                <member>
                    <name> faultString </name>
                    <value>
                        <string> Too many Parameters </string>
                    </value>
                </member>
            </struct>
        </value>
    </fault>
</methodResponse>
```

Code A.6: XML-RPC Fehler

B IPTP

Header-Element	Beispiel	Beschreibung	Anfr.	Antw.
CONNECTION	CLOSE KEEP-ALIVE	Gibt an, ob die Socket-Verbindung nach der Übermittlung der Anfrage und der Rück-Übermittlung der Antwort geschlossen werden soll oder der Server auf eine neue Anfrage wartet.	√	
DATE	2003-06-20 13:18:22 CEST	Datum und Uhrzeit	√	√
USER	ClientUser	Benutzer-Kennung.	√	
SERVER	SDS1	Server-Kennung.		√
SESSION-ID	20030620131822930- acee98a5745b	Session-ID der aktuellen Benutzer-Sitzung. Entspricht der Uhrzeit in Mikrosekunden plus einen zufällig generierter Code. Der Server übermittelt diese Information nach der ersten Anfrage. Der Client kann dann im Folgenden diese Information übermitteln. Die ID stellt kein Sicherheitsfeature dar.	√	√
CONTENT-TYPE	TEXT/XSDML	Nachrichtenformat. Zur Zeit gültiges Nachrichtenformat ist XSDML (siehe Abs. 5.2).	√	√
CONTENT-LINES	60	Anzahl der Textzeilen der Nachricht.	√	√
CONTENT- TRANSFER- ENCODING	BASE64	Kodierung der Nachricht. Wird die Nachricht verschlüsselt, wird dieses Feld zur Angabe der Verschlüsselungstechnologie herangezogen.	√	√

Tabelle B.1: Header-Elemente in IPTP

Code	Fehler-Klasse	Beschreibung
200	OK	Kein Fehler.
401	UNAUTHORIZED	Unautorisierter Zugriff auf das Modul.
500	INTERNAL SERVER ERROR	Bei der Ausführung des Moduls kam es zu einem Fehler, der nicht vom Modul selbst abgefangen werden konnte.
501	NOT IMPLEMENTED	Das angeforderte Modul existiert nicht.

Tabelle B.2: Fehlercodes in IPTP

C XSDML

```
<?xml version="1.0"?>
<sdml>
    <!-- Funktion -->
    <function name="example">

        <!-- Parameter -->
        <reqParaOut>
            <map>
                <mi name="par1"> ... </mi>
                <mi name="par2"> ... </mi>
                <mi name="par3"> ... </mi>
            </map>
        </reqParaOut>

        <!-- Gewünschte Antwort -->
        <resRsltIn>
            <map>
                <mi name="res1"/>
                <mi name="res2"/>
                <mi name="res3"/>
            </map>
        </resRsltIn>

        <!-- Gewünschte Streaming-Daten -->
        <resDataIn>
            <map>
                <mi name="dat1"/>
                <mi name="dat2"/>
                <mi name="dat3"/>
            </map>
        </resDataIn>

        <!-- Streaming-Daten -->
        <reqDataOut>
            <list>
                <le> ... </le>
                <le> ... </le>
                <le> ... </le>
                ...
            </list>
        </reqDataOut>

    </function>
</sdml>
```

Code C.1: XSDML Methodenaufruf

```
<?xml version="1.0"?>
<sdml>
    <!-- Funktion -->
    <result name="example">

        <!-- Streaming-Daten -->
        <resDataOut>
            <list>
                <le>
                    <map>
                        <mi name="dat1"> ... </mi>
                        <mi name="dat2"> ... </mi>
                        <mi name="dat3"> ... </mi>
                    </map>
                </le>
                <le>
                    <map>
                        <mi name="dat1"> ... </mi>
                        <mi name="dat2"> ... </mi>
                        <mi name="dat3"> ... </mi>
                    </map>
                </le>
                ...
            </list>
        </resDataOut>

        <!-- Antwort -->
        <resRsltOut>
           <map>
                <mi name="res1"> ... </mi>
                <mi name="res2"> ... </mi>
                <mi name="res3"> ... </mi>
            </map>
        </resRsltOut>

    <result>
</sdml>
```

Code C.2: XSDML Antwort

D Entfernter Funktionsaufruf

```
String module = "ExampleModule";    // Modul Name
String function  = "getAllInfos";   // Funktion Name

Map reqParaOutMap = new HashMap();  // Parameter
Map resRsltInMap = new HashMap();   // Rückabe-Struktur

// Parameter setzen
reqParaOutMap.put("Person", "Peter"));

// Rückgabe-Struktur setzen
Map infosMap = new HashMap ();
resRsltInMap.put("Infos", infosMap);

// RPC ausführen
remoteProcedureCall (
  module, function,
  reqParaOutMap, resRsltInMap,
  null, null);

// Rückgabe-Daten verwenden
System.out.println(infosMap);
```

Code D.1: Beispiel-Aufruf

Fehler-Klasse	Fehler-Beschreibung
RV_STATUSCODE_OK	Kein Fehler.
RV_STATUSCODE_NI	Die gewünschte Funktion ist nicht implementiert.
RV_STATUSCODE_RP	Ein notwendiger Parameter ist nicht definiert.
RV_STATUSCODE_TI	Der Datentyp des Parameters ist falsch.
RV_STATUSCODE_RD	Notwendige Daten fehlen.
RV_STATUSCODE_ID	Die Struktur der Daten ist falsch (insbesondere Datentyp).
RV_STATUSCODE_DB	Probleme beim Zugriff auf die Datenbank.
RV_STATUSCODE_ML	Probleme mit Mail.
RV_STATUSCODE_UE	Die Funktion wirft eine nicht abgefangene Exception, die jedoch von dem Modul abgefangen wurde.

Tabelle D.1: Fehler-Klassen auf Modul-Ebene

Fehler-Klasse	Fehler-Beschreibung
ERR_NOER	Kein Fehler.
ERR_PROP	SDSBasicConnection wurde mit fehlerhaften Properties initialisiert. Zwei wichtige Properties, mit denen das Verbindungsobjekt initialisiert wird, sind die Angabe der Internet-Adresse des entfernten Servers sowie dessen Port-Nummer.
ERR_INIT	Das Verbindungsobjekt wurde vor dessen Verwendung nicht initialisiert.
ERR_WRFM	Die übertragene Nachricht vom Server hat ein falsches Format.
ERR_CONN	Probleme beim Aufbau der Verbindung zum Server.
ERR_UFRM	Die übertragenen Daten vom Server haben das falsche Format (insbesondere den falschen Typ).
ERR_MESG	Probleme beim Dekodieren der Nachricht.
ERR_MODL	Das Modul meldet einen Fehler.
ERR_SERV	Der Server meldet einen Fehler.
ERR_PARM	Im Allgemeinen wird der Remote Procedure Call auf der Client-Seite auch von spezialisierten Methoden gekapselt. Diese führen vor dem Zusammensetzen der Parameter-Struktur Gültigkeitsüberprüfungen durch und können mit diesem Fehler das Fehlen oder die nicht-Gültigkeit von Parametern anzeigen.
ERR_UNKN	Fehler unbekannter Herkunft.

Tabelle D.2: Fehler-Klassen der SDSConnectionException

```
// Führe Streaming-RPC aus
ClientStreamInOut csio = remoteProcedureCall(
  module,
  function,
  reqParaOutMap,
  resRsltInMap,
  resDataInMap);

// Erzeuge Producer-Thread
Producer p = new Producer (csio);

// Erzeuge Consumer-Thread
Consumer c = new Consumer (csio);

// Start Thread
p.start ();
c.start ();

// Warte bis der RPC abgeschlossen ist
csio.waitUntilRpcFinished();

// Ergebnisauswertung
doSomething (resRsltInMap);
```

Code D.2: Programmbeispiel für Streaming-basierter `remoteProcedureCall`

E Server Infrastruktur

Level	Konst.	Level-Beschreibung
DEBUG	0	Eine Debugging-Information.
INFO	1	Eine weniger bedeutende Information (Standard-Level).
NOTICE	2	Eine interessante Bemerkung.
WARN	3	Etwas ungewöhnliches ist passiert, das Aufmerksamkeit erfordern könnte.
ERROR	4	Ein Fehler ist aufgetreten, das System behandelt diesen Fehler jedoch erwartungsgemäß.
CRIT	5	Ein kritischer Fehler ist aufgetreten, das System behandelt diesen Fehler jedoch erwartungsgemäß.
ALERT	6	Ein schwerer kritischer Fehler ist aufgetreten, der besondere Maßnahme administrative Maßnahmen erfordert.
EMERG	7	Ein dermaßen schwerer kritischer Fehler ist aufgetreten, der nicht behandelt werden kann und zur Beendigung des Servers führt.

Tabelle E.1: Log-Level

F Laufzeittests

Laufzeittests System 1 (C2S)

[ms]	Bytes/Item												
Items	0	8	16	24	32	48	64	96	128	192	256	384	512
0	30	30	30	30	30	30	30	30	30	30	30	30	30
8	30	30	30	30	30	30	30	30	40	40	40	40	40
16	30	30	30	30	30	40	40	40	40	40	40	50	51
24	30	30	31	40	40	40	40	40	40	40	50	60	70
32	30	40	40	40	40	40	40	40	40	50	60	70	80
48	40	40	40	40	40	40	40	41	50	60	70	80	100
64	40	40	40	40	40	40	50	50	60	70	80	91	120
96	40	40	40	50	50	50	50	60	70	80	100	121	160
128	40	50	50	50	50	60	60	70	80	100	120	160	211
192	50	50	60	60	60	70	71	81	100	130	170	220	391
256	50	60	60	70	70	80	90	100	120	151	210	290	581
384	60	70	70	80	90	100	111	140	170	220	291	520	872
512	60	80	90	100	110	120	140	170	210	280	381	741	1112

Tabelle F.1: ME(RMI,C2S,S1)

[%]	Bytes/Item												
Items	0	8	16	24	32	48	64	96	128	192	256	384	512
0	13	13	13	14	12	13	16	13	13	12	12	15	16
8	15	15	15	16	15	16	15	16	15	15	13	12	13
16	17	16	16	15	16	15	14	14	14	11	13	14	11
24	16	16	15	15	15	14	13	13	11	13	13	11	8
32	15	16	15	15	15	13	14	13	12	13	13	10	8
48	15	14	14	12	13	11	15	13	13	12	10	8	7
64	15	13	12	13	14	13	14	12	11	9	9	8	6
96	18	13	15	15	13	13	11	10	9	7	10	9	12
128	13	12	12	13	12	11	9	9	8	8	14	9	13
192	13	12	12	10	10	8	7	8	7	7	11	12	12
256	12	11	10	11	8	8	7	7	7	9	13	14	10
384	12	9	7	7	7	7	6	5	15	11	15	12	10
512	9	8	6	6	5	5	6	17	13	12	13	9	6

Tabelle F.2: VK(RMI,C2S,S1)

[ms]	Bytes/Item												
Items	0	8	16	24	32	48	64	96	128	192	256	384	512
0	340	340	340	340	340	340	340	340	340	340	340	340	340
8	350	350	351	351	351	351	360	360	361	371	380	391	410
16	360	361	361	370	370	371	371	381	391	401	421	450	481
24	371	371	380	381	381	391	391	401	420	440	461	510	570
32	381	390	391	391	400	410	411	430	441	471	510	581	731
48	401	411	420	421	431	441	451	475	500	541	610	791	931
64	421	440	441	451	461	471	491	521	551	630	791	941	1252
96	470	481	500	511	521	541	571	621	680	872	1002	1402	1657
128	511	531	551	570	581	621	651	761	901	1032	1362	1672	2404
192	591	631	661	691	721	791	941	1022	1142	1512	1883	2864	4146
256	681	731	771	822	941	1031	1082	1232	1552	1962	2583	4266	6319
384	901	1041	1116	1162	1212	1322	1622	1803	2123	3185	4466	6860	7971
512	1121	1262	1322	1412	1472	1803	1978	2243	3245	4757	6740	8182	11156

Tabelle F.3: ME(SOAP,C2S,S1)

[%]	Bytes/Item												
Items	0	8	16	24	32	48	64	96	128	192	256	384	512
0	2	2	2	2	2	2	2	2	2	2	2	2	2
8	2	2	2	2	2	2	2	2	2	2	2	2	2
16	2	2	2	2	2	2	2	2	2	2	2	2	2
24	2	2	2	2	2	2	2	2	2	2	2	2	2
32	2	2	2	2	2	2	2	2	2	2	2	6	8
48	2	2	2	2	2	2	2	2	2	2	6	7	6
64	2	2	2	2	2	2	2	2	2	6	7	6	16
96	2	2	2	2	2	2	2	2	6	7	6	15	13
128	2	2	2	2	2	2	5	8	5	5	13	14	14
192	2	2	2	5	7	7	5	5	5	15	14	11	11
256	2	2	5	7	6	5	4	13	14	14	13	10	7
384	6	5	4	4	4	4	10	12	14	11	9	6	6
512	5	4	4	3	11	12	12	12	10	9	6	5	3

Tabelle F.4: VK(SOAP,C2S,S1)

[ms]	Bytes/Item												
Items	0	8	16	24	32	48	64	96	128	192	256	384	512
0	100	100	100	100	100	100	100	100	100	100	100	100	100
8	120	120	120	120	120	120	130	130	140	151	170	211	261
16	130	131	140	140	141	150	150	170	180	210	240	330	431
24	151	151	160	160	170	171	180	200	220	261	311	450	621
32	170	170	180	181	190	200	210	231	260	320	391	601	831
48	210	210	220	230	231	250	261	300	341	431	551	871	1162
64	251	251	260	270	280	300	321	370	421	591	761	1101	1522
96	340	331	341	360	371	410	441	520	631	871	1092	1583	2193
128	411	410	440	451	480	546	591	711	861	1141	1402	2043	2874
192	581	591	621	651	731	791	871	1092	1252	1593	2023	3025	4156
256	802	811	852	892	951	1052	1192	1412	1612	2063	2624	3986	5523
384	1201	1212	1282	1382	1482	1633	1753	2023	2323	3014	3946	5969	8131
512	1612	1647	1772	1902	1953	2123	2293	2624	3054	4116	5398	7951	10885

Tabelle F.5: ME(SDS,C2S,S1)

[%]	Bytes/Item												
Items	0	8	16	24	32	48	64	96	128	192	256	384	512
0	6	6	6	6	6	7	6	6	6	6	6	5	7
8	23	23	23	8	20	19	22	20	18	19	18	17	14
16	19	23	19	21	21	19	18	20	16	15	14	12	9
24	18	19	19	19	18	16	16	18	15	14	12	10	8
32	18	18	17	19	19	16	14	19	15	13	12	9	8
48	16	19	19	17	18	18	18	15	15	12	10	8	6
64	18	20	17	19	15	19	16	14	15	12	10	7	5
96	19	22	20	20	19	17	17	14	13	11	8	5	4
128	19	19	20	20	18	17	17	13	12	9	7	6	4
192	22	23	21	21	20	17	16	12	12	10	7	5	4
256	23	22	20	20	18	16	15	13	12	9	7	5	4
384	21	20	19	18	16	15	15	14	12	9	7	5	4
512	20	19	17	17	17	16	16	14	12	9	7	5	4

Tabelle F.6: VK(SDS,C2S,S1)

Laufzeittests System 1 (S2C)

[ms]	Bytes/Item												
Items	0	8	16	24	32	48	64	96	128	192	256	384	512
0	30	30	30	30	30	30	30	30	30	30	30	30	30
8	30	30	30	30	30	30	30	30	30	40	40	40	40
16	30	30	30	30	30	30	40	40	40	40	40	50	60
24	30	30	30	40	40	40	40	40	40	40	50	60	70
32	30	31	40	40	40	40	40	40	40	50	60	70	80
48	30	40	40	40	40	40	40	50	50	60	70	80	100
64	40	40	40	40	40	40	50	50	60	70	80	100	120
96	40	40	40	50	50	50	60	60	70	80	101	130	170
128	40	50	50	50	50	60	60	70	80	100	130	160	300
192	50	50	60	60	60	70	80	90	100	130	170	230	410
256	50	60	60	70	70	80	90	100	121	160	211	370	661
384	60	70	70	80	90	100	111	140	170	230	301	520	871
512	60	80	81	100	100	120	140	170	211	281	400	741	1061

Tabelle F.7: ME(RMI,S2C,S1)

[%]	Bytes/Item												
Items	0	8	16	24	32	48	64	96	128	192	256	384	512
0	14	17	14	17	13	15	15	14	18	15	14	14	14
8	16	17	13	16	16	16	16	16	16	16	13	15	13
16	15	16	15	15	16	16	15	15	14	12	12	12	12
24	16	16	15	16	15	15	14	15	12	14	13	11	8
32	16	16	15	16	14	14	15	14	14	14	13	9	6
48	15	16	17	12	15	15	14	14	13	11	7	6	5
64	15	13	12	15	15	13	13	12	12	9	6	6	23
96	16	14	13	13	11	13	12	9	8	6	5	21	23
128	15	12	14	12	11	12	9	6	30	6	20	17	19
192	12	11	10	10	7	9	7	8	4	4	3	18	11
256	11	9	7	10	7	7	7	5	4	2	11	15	8
384	11	8	7	7	7	5	5	3	17	17	12	11	4
512	9	8	7	7	7	5	18	19	16	14	11	6	2

Tabelle F.8: VK(RMI,S2C,S1)

[ms]	Bytes/Item												
Items	0	8	16	24	32	48	64	96	128	192	256	384	512
0	340	340	340	341	340	340	341	340	340	340	340	340	340
8	351	351	351	360	360	360	361	370	371	381	391	420	451
16	361	361	370	371	371	380	381	391	401	421	451	520	661
24	371	380	381	381	390	391	401	420	431	480	511	681	782
32	381	391	391	401	401	411	421	441	470	520	651	781	941
48	401	420	421	431	440	451	470	501	541	701	781	981	1232
64	430	441	451	461	471	491	521	570	691	791	941	1222	1472
96	471	500	511	531	551	651	681	761	861	1031	1232	1582	2223
128	521	561	661	681	701	741	791	931	1002	1272	1522	2293	2974
192	701	751	782	811	851	962	1021	1201	1432	1753	2423	3065	4466
256	801	861	951	1001	1041	1112	1271	1522	1742	2524	3025	4377	5759
384	1061	1152	1212	1322	1452	1612	1763	2434	2694	3184	4676	6720	9844
512	1282	1432	1592	1742	1813	2033	2504	2944	3285	4727	6289	9724	11556

Tabelle F.9: ME(SOAP,S2C,S1)

[%]	Bytes/Item												
Items	0	8	16	24	32	48	64	96	128	192	256	384	512
0	2	2	2	2	2	2	3	2	2	2	2	2	2
8	2	2	2	2	2	2	2	2	2	2	2	2	10
16	2	2	2	2	2	2	2	2	2	2	8	10	7
24	2	2	2	2	2	2	2	2	2	10	10	7	6
32	2	2	2	2	2	2	2	2	8	10	8	7	6
48	2	2	2	2	2	2	7	10	9	7	7	6	6
64	2	2	2	2	7	8	10	9	8	7	6	7	6
96	2	8	9	9	9	9	8	7	6	6	6	6	10
128	9	9	9	8	8	7	7	7	6	7	6	12	8
192	8	7	6	6	6	6	5	6	12	12	12	6	7
256	6	6	6	6	6	6	7	6	5	8	7	7	7
384	6	6	6	5	7	7	10	10	8	2	6	7	3
512	6	6	6	6	5	8	9	7	7	7	8	3	2

Tabelle F.10: VK(SOAP,S2C,S1)

[ms]	Bytes/Item												
Items	0	8	16	24	32	48	64	96	128	192	256	384	512
0	100	100	100	100	100	100	100	100	100	100	100	100	100
8	131	130	140	140	140	140	140	150	151	170	181	221	280
16	170	170	170	180	180	180	190	200	210	240	271	431	521
24	210	210	210	211	220	221	231	250	270	311	371	561	701
32	241	241	250	251	260	270	280	301	331	421	530	685	881
48	321	320	330	331	341	360	371	421	495	611	711	942	1262
64	401	400	410	421	431	451	511	591	641	752	892	1242	1632
96	620	610	661	651	671	711	741	821	892	1082	1302	1813	2448
128	791	781	801	831	851	891	941	1052	1181	1402	1693	2384	3204
192	1126	1122	1172	1241	1227	1301	1382	1543	1703	2083	2543	3575	4877
256	1492	1472	1512	1572	1632	1723	1802	2013	2224	2764	3385	4817	6539
384	2188	2213	2283	2343	2414	2558	2714	3035	3355	4206	5057	7250	10005
512	2874	2944	3035	3114	3215	3415	3595	3926	4441	5658	7051	9694	12978

Tabelle F.11: ME(SDS,S2C,S1)

[%]	Bytes/Item												
Items	0	8	16	24	32	48	64	96	128	192	256	384	512
0	28	24	26	6	4	5	24	5	5	5	5	26	26
8	18	5	20	5	5	21	19	16	20	19	15	14	17
16	15	17	18	19	17	7	16	17	12	14	12	17	12
24	16	15	15	16	17	14	14	13	14	16	16	12	8
32	17	15	15	15	15	14	17	17	16	16	12	8	7
48	18	18	19	18	18	17	18	17	16	12	9	7	6
64	18	17	18	18	18	19	17	15	12	9	8	6	5
96	18	18	17	17	16	13	13	12	11	9	7	6	6
128	16	16	16	14	14	14	14	13	10	9	8	6	6
192	14	15	14	14	14	13	12	11	11	9	9	7	5
256	15	15	16	15	13	13	13	12	12	11	9	7	6
384	15	14	16	16	14	16	15	14	12	10	8	6	5
512	18	16	15	17	14	15	14	14	11	10	10	6	4

Tabelle F.12: VK(SDS,S2C,S1)

Laufzeittests System 2 (C2S)

[ms]	Bytes/Item												
Items	0	8	16	24	32	48	64	96	128	192	256	384	512
0	50	50	50	50	50	50	50	50	50	50	50	50	50
8	50	50	50	50	50	50	60	60	60	60	70	70	70
16	51	50	50	60	51	60	60	60	60	70	80	90	91
24	50	50	55	60	60	60	60	65	70	80	90	100	120
32	50	51	60	60	60	60	60	70	80	90	110	120	140
48	51	60	60	60	70	70	70	80	90	101	140	150	171
64	60	60	60	70	70	70	80	90	110	120	161	181	211
96	60	70	70	70	80	80	90	110	130	150	220	250	291
128	60	70	80	80	90	91	110	120	150	180	270	310	370
192	70	80	90	100	100	120	130	151	190	250	371	440	671
256	80	90	100	110	120	130	151	180	230	310	471	561	971
384	90	110	120	140	150	170	200	250	320	440	681	966	1542
512	100	130	140	160	180	210	250	310	405	561	891	1382	1793

Tabelle F.13: ME(RMI,C2S,S2)

[%]	Bytes/Item												
Items	0	8	16	24	32	48	64	96	128	192	256	384	512
0	11	11	11	11	13	11	10	12	13	13	13	7	12
8	9	7	9	9	7	11	9	10	14	8	9	8	8
16	10	12	9	11	11	9	8	6	10	8	6	10	7
24	7	15	10	12	9	7	11	9	11	9	6	11	5
32	12	11	8	9	9	7	9	10	8	10	6	4	3
48	12	9	14	13	9	9	5	10	8	9	4	3	3
64	11	12	9	10	9	11	9	9	10	7	5	3	3
96	9	9	13	12	9	6	9	5	4	4	2	3	9
128	9	12	12	7	11	5	10	6	4	3	2	2	6
192	12	11	7	6	7	4	4	4	2	2	2	10	9
256	11	6	9	8	5	4	4	4	2	1	1	8	6
384	12	7	4	5	4	3	3	2	2	1	1	7	9
512	10	6	5	3	22	3	2	2	2	5	3	4	3

Tabelle F.14: VK(RMI,C2S,S2)

[ms]	Bytes/Item												
Items	0	8	16	24	32	48	64	96	128	192	256	384	512
0	350	350	351	350	350	350	350	350	351	351	345	350	350
8	370	361	370	370	370	370	371	380	390	400	410	430	460
16	380	381	390	390	390	391	401	411	421	440	461	511	561
24	390	401	401	411	411	420	421	441	460	491	531	591	681
32	410	420	421	431	431	441	451	471	491	531	591	691	911
48	440	451	461	470	471	491	511	541	571	641	731	971	1202
64	471	481	495	511	521	541	561	611	651	761	982	1212	1703
96	531	555	581	591	601	650	681	741	821	1131	1272	1983	2253
128	590	621	651	671	691	751	791	931	1142	1342	1798	2333	3375
192	716	761	801	842	882	962	1182	1342	1492	2027	2563	3926	5748
256	832	902	972	1012	1202	1312	1412	1612	2073	2679	3605	5898	8422
384	1111	1312	1442	1502	1583	1762	2183	2414	2975	4912	6169	9559	11106
512	1432	1652	1732	1863	1942	2413	2644	3074	4376	6740	9369	11486	15593

Tabelle F.15: ME(SOAP,C2S,S2)

[%]	Bytes/Item												
Items	0	8	16	24	32	48	64	96	128	192	256	384	512
0	4	3	3	3	3	2	3	3	3	3	3	3	3
8	3	3	3	3	3	3	3	2	4	2	2	3	3
16	3	3	3	2	2	3	3	3	3	3	4	2	3
24	3	3	3	3	3	2	2	3	3	3	3	4	2
32	2	2	3	3	2	3	3	4	2	3	3	7	10
48	3	3	2	3	3	4	4	3	2	2	7	9	6
64	2	3	2	3	3	3	3	3	3	2	7	7	21
96	2	3	3	3	2	4	3	3	9	11	5	16	18
128	3	3	4	3	3	2	6	10	6	4	6	16	16
192	2	3	3	7	8	7	4	6	5	6	13	12	12
256	2	2	8	6	6	6	7	5	6	13	14	12	8
384	7	5	5	5	4	4	16	12	14	12	11	8	4
512	6	3	3	4	12	15	12	11	13	11	7	6	4

Tabelle F.16: VK(SOAP,C2S,S2)

[ms]	Bytes/Item												
Items	0	8	16	24	32	48	64	96	128	192	256	384	512
0	170	170	170	171	170	170	171	170	170	170	170	170	170
8	230	230	231	230	231	240	240	251	260	281	301	360	431
16	290	290	290	291	300	301	320	330	350	400	451	571	701
24	350	341	360	361	361	380	391	411	451	511	581	751	1012
32	401	401	411	421	431	451	461	501	540	641	721	941	1297
48	531	530	541	551	561	581	621	671	731	861	1061	1422	1823
64	656	661	666	681	701	726	766	901	971	1151	1352	1887	2424
96	891	902	957	942	1031	1032	1131	1241	1347	1652	1913	2653	3520
128	1182	1221	1171	1297	1321	1362	1422	1587	1823	2183	2553	3465	4582
192	1737	1723	1772	1832	1853	1943	2058	2383	2594	3154	3750	5067	6669
256	2243	2233	2318	2358	2384	2594	2794	3044	3385	4061	4877	6630	8868
384	3304	3304	3385	3515	3626	3890	4070	4557	4967	5988	7220	9824	12869
512	4376	4426	4606	4767	4867	5122	5443	5953	6519	7911	9444	12793	17114

Tabelle F.17: ME(SDS,C2S,S2)

[%]	Bytes/Item												
Items	0	8	16	24	32	48	64	96	128	192	256	384	512
0	3	20	3	27	3	4	20	3	3	3	4	23	3
8	2	15	15	2	3	14	2	15	3	10	12	2	2
16	13	13	11	2	13	2	13	2	11	9	13	10	7
24	14	3	13	14	11	1	10	2	11	9	8	9	6
32	10	1	7	10	8	12	6	8	10	8	6	7	7
48	9	7	9	7	9	9	8	8	7	7	7	6	4
64	6	6	4	6	7	7	6	7	6	6	6	3	3
96	6	6	6	6	5	5	6	5	5	5	4	3	3
128	6	7	5	5	5	5	5	5	5	1	4	2	2
192	4	4	4	4	4	4	4	3	3	3	3	2	1
256	3	3	3	3	3	3	3	3	3	2	2	1	1
384	3	3	3	3	2	3	2	3	3	1	2	1	2
512	2	3	2	2	2	2	2	2	2	2	1	1	2

Tabelle F.18: VK(SDS,C2S,S2)

Laufzeittests System 2 (S2C)

[ms]	Bytes/Item												
Items	0	8	16	24	32	48	64	96	128	192	256	384	512
0	50	50	50	50	50	50	50	50	50	50	50	50	50
8	50	50	50	50	50	50	50	50	50	50	60	60	70
16	50	50	50	50	50	50	60	60	60	60	70	80	90
24	50	50	50	50	60	60	60	60	70	70	80	90	100
32	50	50	50	60	60	60	60	60	70	80	91	100	120
48	60	60	60	60	60	60	60	70	80	90	110	130	160
64	50	60	60	60	61	65	70	80	90	101	130	160	190
96	60	60	60	70	70	80	80	100	110	130	171	200	255
128	61	70	70	80	80	90	90	110	130	160	200	250	440
192	70	80	81	90	91	100	110	131	161	201	270	340	591
256	80	80	90	100	110	120	135	161	190	241	330	431	971
384	80	100	110	130	130	160	180	210	260	341	460	751	1262
512	91	115	130	150	160	190	211	260	321	421	581	1076	1532

Tabelle F.19: ME(RMI,S2C,S2)

[%]	Bytes/Item												
Items	0	8	16	24	32	48	64	96	128	192	256	384	512
0	15	9	10	15	13	11	11	14	10	14	14	9	8
8	0	9	9	9	12	11	8	9	12	12	10	10	8
16	6	10	14	9	12	13	14	11	10	8	9	10	9
24	12	9	12	9	11	14	11	13	10	11	10	8	4
32	11	8	11	10	11	18	12	9	10	11	11	8	5
48	12	12	11	9	9	13	8	10	10	8	7	5	5
64	10	13	9	10	9	13	11	9	10	7	5	4	2
96	11	12	14	7	12	9	10	11	7	3	4	3	20
128	11	14	10	10	9	9	10	6	5	25	3	2	17
192	12	8	9	5	6	40	4	5	26	2	2	2	11
256	14	8	8	6	4	7	4	5	2	2	13	14	7
384	11	8	6	6	4	4	3	3	3	17	11	1	3
512	9	6	5	3	4	2	3	2	18	11	11	6	1

Tabelle F.20: VK(RMI,S2C,S2)

[ms]	Bytes/Item												
Items	0	8	16	24	32	48	64	96	128	192	256	384	512
0	351	350	355	350	351	345	350	350	350	351	351	360	351
8	371	371	370	371	380	380	381	391	391	411	421	460	511
16	381	390	390	391	400	401	401	421	440	470	501	626	796
24	391	405	410	411	420	431	431	461	481	541	691	831	961
32	411	421	425	431	440	451	470	491	521	591	791	966	1202
48	431	460	461	471	481	500	520	581	751	851	951	1236	1623
64	465	491	500	521	531	551	591	650	841	1001	1176	1583	1918
96	531	581	581	611	671	791	841	941	1042	1402	1612	2098	2879
128	581	751	801	831	861	901	972	1172	1272	1763	2073	3124	4100
192	871	922	981	1011	1052	1231	1312	1562	1983	2303	3325	4146	6329
256	992	1092	1231	1282	1322	1422	1642	1963	2334	3480	4156	6299	7836
384	1352	1582	1642	1712	1838	2173	2354	3255	3650	4307	6569	8272	13835
512	1672	1877	2093	2319	2383	2744	3285	4005	4466	6704	8667	13629	16274

Tabelle F.21: ME(SOAP,S2C,S2)

[%]	Bytes/Item												
Items	0	8	16	24	32	48	64	96	128	192	256	384	512
0	4	3	3	3	3	3	3	4	4	4	3	3	3
8	3	4	3	3	3	2	3	4	3	3	3	3	14
16	2	3	3	2	2	3	2	2	3	3	12	13	9
24	3	3	3	3	3	3	2	3	3	12	12	9	9
32	3	3	3	2	3	2	3	2	3	11	10	8	8
48	2	2	3	3	2	3	3	12	11	9	8	8	7
64	2	3	2	10	11	13	11	11	9	8	7	7	7
96	2	13	11	12	11	10	9	9	7	19	6	7	9
128	2	11	11	10	10	8	9	9	7	5	14	13	3
192	9	7	6	6	9	8	5	7	7	8	7	7	6
256	7	8	6	6	7	7	9	8	13	12	7	6	6
384	7	6	17	6	7	8	12	7	4	3	5	6	2
512	6	7	7	7	7	8	9	8	8	5	9	4	2

Tabelle F.22: VK(SOAP,S2C,S2)

[ms]	Bytes/Item												
Items	0	8	16	24	32	48	64	96	128	192	256	384	512
0	170	170	170	170	170	170	170	170	170	170	170	170	170
8	205	201	200	210	210	211	220	220	231	260	281	340	420
16	240	240	240	250	251	260	261	281	300	350	400	571	771
24	270	271	281	291	291	310	320	350	380	441	531	791	996
32	311	321	330	330	341	360	371	401	441	561	761	971	1242
48	401	401	420	430	441	466	491	531	601	832	972	1367	1733
64	491	491	511	530	540	561	606	781	872	1021	1262	1682	2273
96	661	676	701	721	751	821	951	1051	1142	1442	1723	2423	3455
128	941	922	966	992	1041	1102	1172	1312	1492	1792	2243	3295	4527
192	1352	1352	1392	1432	1457	1522	1712	1873	2143	2664	3430	4756	6814
256	1797	1752	1802	1867	1962	2033	2213	2394	2744	3490	4327	6479	9458
384	2624	2604	2729	2794	2874	2999	3134	3685	3975	5263	6750	9939	13670
512	3505	3455	3540	3676	3695	3975	4246	4791	5437	6940	8833	12928	17886

Tabelle F.23: ME(SDS,S2C,S2)

[%]	Bytes/Item												
Items	0	8	16	24	32	48	64	96	128	192	256	384	512
0	4	28	23	3	4	3	3	3	3	22	23	3	22
8	21	18	3	3	2	20	20	2	19	17	15	10	15
16	3	2	3	17	17	18	2	15	3	11	11	15	12
24	2	12	12	16	13	15	13	13	12	11	13	12	9
32	2	14	14	2	12	12	11	11	10	15	12	9	6
48	13	12	11	10	8	9	7	10	11	10	9	5	5
64	9	11	12	9	8	8	10	13	10	7	7	5	5
96	6	7	13	9	9	11	10	8	7	6	5	6	5
128	11	10	10	9	10	8	7	6	6	5	5	6	4
192	7	6	6	6	5	6	5	5	5	5	6	5	4
256	5	4	6	5	5	5	4	5	4	6	5	4	4
384	3	3	3	5	4	5	3	5	6	4	4	4	3
512	3	3	3	3	4	4	5	4	5	3	4	2	4

Tabelle F.24: VK(SDS,S2C,S2)

Glossar

Abstract Windows Toolkit *Siehe* **AWT**

Access Control List *Siehe* **ACL**

ACL *Access Control List*. Liste, die den Zugriff auf Ressourcen reglementiert.

ActiveX Komponenten-Technologie von Microsoft.

ActiveX Data Objects *Siehe* **ADO**

Active Server Pages *Siehe* **ASP**

ADO *ActiveX Data Objects*. Klassenbibliothek für den Datenbank-Zugriff in der Windows-Plattform.

API *Applicaton Programm Interface*. Die Möglichkeit, eine Klassenbibliothek nutzen zu können, wird durch ihre Programmschnittstelle, die API, beschrieben.

Applet *Java*-Programme, die innerhalb eines *Web-Browsers* laufen, u.a. um die dargestellten Seiten dynamischer oder interaktiver zu machen. Aus Sicherheitsgründen laufen Applets innerhalb einer *Sandbox* im Browser, um den Zugriff auf System-Ressourcen zu reglementieren.

Application Programm Interface *Siehe* **API**

ASP *Active Server Pages*. Technologie von Microsoft für serverseitige Scripte.

Assembly Zusammenfassung mehrerer *PE*-Module, Ressourcen und einem *Manifest*. Ein Assembly repräsentiert eine *.NET*-Anwendung.

AWT *Abstract Windows Toolkit*. Klassenbibliothek von Java für *GUI*-Programmierung.

Base64 Verfahren zur Transformation von Binärdateien in Text.

Basic Object Adapter *Siehe* **BOA**

Bean Komponenten in Java, auch *JavaBean* oder *Enterprise Java Bean*.

Bean Managed Persistence *Siehe* **BMP**

BMP *Bean Managed Persistence*. Verfahren, bei der ein *EJB* sich selbst um seine *Persistenz* kümmern muss. Gegenteil von *CMP*.

BOA *Basic Object Adapter*. Objekt Adapter bei *CORBA*.

Broadcast-Adresse Eine Broadcast-Adresse ist eine spezielle *IP-Adresse*, mit der mehrere Rechner in einem Netz gleichzeitig angesprochen werden können.

C# Objektorientierte Programmiersprache von Microsoft. Teil der *.NET* Initiative

Cascading Style Sheeds *Siehe* **CSS**

ccTLD *Country Code TLD*. Länderspezifische *TDLs*, z.B. „de“: Deutschland, „uk“: Großbritannien.

CGI *Common Gateway Interface*. Definiert den Weg, wie Formulardaten einer Web-Seite über den Web-Server zu einer Anwendung geführt werden, damit diese auf der Grundlage dieser Daten eine dynamisch generierte Antwort-Seite liefern kann.

CIL *Common Intermediate Language*. *Microsoft Intermediate Language*. Programme, die für das *.NET-Framworks* entwickelt werden, werden in plattformunabhängigen Bytecode übersetzt, der innerhalb der *CLR* lauffähig ist. Auch als *Microsoft Intermediate Language (MSIL)* oder *Intermediate Language (IL)* bezeichnet.

Class ID *Siehe* **CLSID**

Client Ein Client nimmt *Dienste* in Anspruch, die von einem *Server* zur Verfügung gestellt werden.

CLR *Common Language Runtime*. Laufzeitumgebung für *.NET*-Anwendungen

CLS *Common Language Specification*. Spezifikation, die garantiert, dass CLS-konforme Programmiersprachen einen Code produzieren, der innerhalb der *CLR* lauffähig ist.

CLSID *Class ID*. Eindeutige ID einer Klasse. Sie wird bei *DCOM* verwendet.

CMP *Container Managed Persistence*. Verfahren, bei dem der *Bean Container* für die *Persistenz* eines *EJB* sorgt.

COM *Component Object Model*. Komponenten-Technologie von Microsoft.

Common Gateway Interface *Siehe* **CGI**

Common Intermediate Language *Siehe* **CIL**

Common Language Runtime *Siehe* **CLR**

Common Language Specification *Siehe* **CLS**

Common Object Request Broker Architecture *Siehe* **CORBA**

Component Object Model *Siehe* **COM**

Container Managed Persistence *Siehe* **CMP**

CORBA *Common Object Request Broker Architecture*. Architektur für den Zugriff auf entfernte Objekte. Spezifiziert durch die *OMG*.

Common Type System *Siehe* **CTS**

CSS *Cascading Style Sheeds*. Technologie zur Trennung von Inhalt und Layout von *HTML*-Seiten. Die CSS definieren das Layout.

CTS *Common Type System*. Typensystem in *.NET*, das die Inter-Operabilität von .NET-Programmiersprachen garantiert.

Datagram Datenpakete, die im Internet über *IP* übertragen werden.

DCE *Distributed Computer Environment. Middleware*-Architektur der *OSF*.

DCOM *Distributed Component Object Model*. Erweiterung der *COM*-Technologie um den Aspekt der Verteilung.

DDE *Dynamic Data Exchange*. Technologie von Microsoft zum Austausch von Daten zwischen Anwendungen.

Dienst *Server* stellen im *Internet* Dienste *(Service)* zur Verfügung, z.B, Mail-Service, Web-Service, o.ä.

DII *Dynamic Invocation Interface*. Bei *CORBA* werden dynamische Aufrufe auf der Client-Seite durch das DII und auf der Server-Seite durch das *DSI* abgearbeitet.

Distributed Component Object Model *Siehe* **DCOM**

Distributed Computer Environment *Siehe* **DCE**

DNS *Domain Name System*. *Service*, der aus *Internet-Namen* die dazugehörige *IP-Adresse* liefert.

Document Root Wurzelverzeichnis in einem Web-Server, unter dem die Web-Seiten abgelegt werden.

Document Type Definition *Siehe* **DTD**

DOD U.S. *Departement of Defense*

DOD-Modell Modell der *DOD* zur Beschreibung der Kommunikation von Rechnern im *Internet*.

DOM *Document Object Modell*. Dieses Modell definiert, wie die Objekte in einer HTML-Seite angesprochen werden können und welche Eigenschaften sie besitzen.

Domain Name System *Siehe* **DNS**

DSI *Dynamic Skeleton Interface*. Bei *CORBA* werden dynamische Aufrufe auf der Client-Seite durch das DII und auf der Server-Seite durch das *DSI* abgearbeitet.

DTD *Document Type Definition*. Möglichkeit, die Gültigkeit von *XML*-Dokumenten festzulegen.

Dynamic Data Exchange *Siehe* **DDE**

Dynamic Invocation Interface *Siehe* **DII**

Dynamic Skeleton Interface *Siehe* **DSI**

ECMA *European Computer Manufacturers Association*

ECMAScript Specification der *ECMA*, auf dessen Basis *JavaScript* und *JScript* aufbauen.

EJB *Enterprise Java Beans*. Komponenten-Technologie von Sun auf Basis von Java für Geschäftsprozesse.

EJB Container Kontainer, der ein *EJB* verwaltet (Lebenszyklus, etc.) und die Schnittstelle für das Bean nach außen bereitstellt.

Enterprise Java Beans *Siehe* **EJB**

Exception Ausnahmesituation (z.B. Fehler) während eines Programmlaufes, die z.B. in *Java* dadurch ausgedrückt wird, dass ein von der Klasse `Throwable` abgeleitetes *Objekt* erzeugt wird und der Programmfluss innerhalb eines try-catch-Blocks an der Catch-Anweisung fortgesetzt wird, die für die definierte Exception-Klasse zuständig ist.

eXtensible Markup Language *Siehe* **XML**

File Transfer Protocol *Siehe* **FTP**

Firewall Mechanismus zum Schutz eines Netzes vor unbefugtem Zugriff. In der Firewall sind Regeln definiert, welche Zugriffe in das Netz und aus dem Netz erlaubt sind.

Garbage Collection Speicher, der während eines Programmlaufes nicht mehr referenziert wird, kann mit der Garbage Collection wieder verfügbar gemacht werden.

General Inter-ORB Protocol *Siehe* **GIOP**

GIOP *General Inter-ORB Protocol*. Die Interoperabilität zwischen zwei *ORBs* wird durch dieses Protokoll garantiert.

Globally Unique Identifier *Siehe* **GUID**

Graphical User Interface *Siehe* **GUI**

gTLD *Generic TLD*. Themenbezogene *TLDs*, z.B. „org“: non-Profit-Oranisationen, „com“: Commercial.

GUI *Graphical User Interface*. Grafische Benutzer-Schnittstelle bei Anwendungen.

GUID *Globally Unique Identifier*. Eindeutige ID zum Identifizieren von Klassen/Schnittstellen. Sie werden bei *DCOM* als *IID* oder *CLSID* verwendet.

FTP *File Transfer Protocol*. Protokoll zum Transfer von Dateien im *Internet*.

HTML *HyperText Markup Language*. Seitenbeschreibungssprache für Internet-Seiten.

HTTP *HyperText Transfer Protocol*. Protokoll, u.a. zum Übertragen von Dokumenten im *Internet* zwischen *Web-Server* und *Web-Browser*.

Hyperlink Verweis innerhalb eines *HTML*-Dokumentes auf eine Ressource im Internet, zu dem der Browser nach „Anklicken“ des Links wechselt. Hyperlinks werden durch *URL's* beschrieben und machen das Navigieren im Internet erst möglich.

Hypertext-Dokument Dokument, das neben Text und Grafik auch weitere Multimedia-Elemente beinhalten kann. Im *Internet* werden diese Dokumente zur Zeit in *HTML* beschrieben.

HyperText Markup Language *Siehe* **HTML**

HyperText Transfer Protocol *Siehe* **HTTP**

ICMP *Internet Control Message Protocol*. Protokoll, das Bestandteil von *IP* ist und zur Versendung von Kontrollinformationen dient.

IDL *Interface Description Language*. Programmiersprachen-unhabhängige Beschreibungssprache von *Interfaces*.

IID *Interface ID*. Eindeutige ID einer Schnittstelle. Sie wird bei *DCOM* verwendet.

IIOP *Internet Inter-ORB Protocol*. Protokoll, das bei *CORBA* oder *RMI* Verwendung findet, um den Zugriff auf entfernte Objekte zu ermöglichen.

IL *Siehe* **CIL**

Intermediate Language *Siehe* **CIL**

IMAP *Internet Message Access Protocol*. Protokoll zum Abruf von eMail aus dem Postfach.

Interface Schnittstelle zwischen Programmaufruf und Programmcode.

Interface Description Language *Siehe* **IDL**

Interface ID *Siehe* **IID**

Intermediate Language *Siehe* **CIL**

Internet Weltweiter Verbund von Rechnern

Internet-Adresse Spezifiziert eine Ressource im *Internet* als *URL*.

Internet Control Message Protocol *Siehe* **ICMP**

Internet-Domain Jeder Rechner im *Internet* mit einem *Internet-Name* ist in einer eindeutigen Domäne (Domain) positioniert.

Internet Inter-ORB Protocol *Siehe* **IIOP**

Internet Message Access Protocol *Siehe* **IMAP**

Internet-Name Eindeutiger Name eines Rechners im *Internet*, der sich aus dem Rechner-Namen, *Sub-Domänen* und der *Top-Level-Domäne* zusammensetzt.

Internet Protocol *Siehe* **IP**

Intranet Ein auf *Internet*-Technologie aufbauendes lokales Netz, das keinen direkten Zugang zum Internet hat.

Introspection Technologie, die mittels *Reflection* die Eigenschaften von *JavaBeans* ermittelt.

IP *Internet Protocol* Protokoll zur Übertragung von Datenpaketen (*Datagrammen*) im *Internet*.

IPv6 *Internet Protocol Version 6*. In dieser Version von *IP* besteht die *IP-Adresse* aus 6 Byte.

IP-Adresse Eine IP-Adresse besteht aus 4 Byte, die durch Punkte getrennt sind, z.B. „153.96.230.40“. Sie spezifiziert eindeutig einen Rechner im *Internet*. In der Version *IPv6* besteht die Adresse aus 6 Byte.

ISAPI *Information Server API*. Möglichkeit, die Fähigkeiten des Internet Information Servers über eine definierte Schnittstelle zu erweitern.

ISO *International Standards Organisation*

ISO-OSI-Modell *Open Systems Interconnect Reference Modell* der ISO-Gesellschaft. Dieses Modell beschreibt die Kommunikation zwischen Rechnern in offenen Netzen.

J2SE *Java 2 Plattform Standard Edition*. Standard *Java2*-API

J2EE *Java 2 Plattform Enterprise Edition*. Erweiterung *Java2* zum Einsatz in Unternehmen durch *Enterprise Java Beans*.

JAR *Java Archiv*. Zusammenfassung von *Java*-Klassen und Ressourcen, u.U. auch in komprimierter Form.

Java Objektorientierte Programmiersprache von Sun Microsystems.

Java2 *Java* ab Version 1.2

Java Archiv *Siehe* **JAR**

Java API for XML Processing *Siehe* **JAXP**

Java Bean Komponente in Java, hauptsächlich als visuelle Komponente mit *GUI*-Funktionalität.

Java Database Connectivity *Siehe* **JDBC**

JavaMail EMail-Unterstützung für *Java*.

Java Message Service *Siehe* **JMS**

Java Naming and Directory Interface *Siehe* **JNDI**

Java Plug-in *Plug-in* für Browser, um *Applets* ohne eigene *JVM* des Browsers ausführen zu können.

JavaScript Scriptsprache, mit der die Darstellung einer Web-Seite im Browser dynamischer werden kann, ohne neue Verbindungen zum Web-Server aufbauen zu müssen. Entspricht dem *ECMAScript*-Standard.

Java Runtime Environment *Siehe* **JRE**

Java Server Pages *Siehe* **JSP**

Java Transaction API *Siehe* **JTA**

Java Virtual Machine *Siehe* **JVM**

JAXP *Java API for XML Processing*. API für die Bearbeitung/Erstellung von XML-Dokumenten.

JDBC *Java Database Connectivity*. Datanbankunterstützung in *Java*.

Jini Netzwerk-Technologie von Sun auf der Basis von *Java*.

JIT *Just-in-Time-Compiler*. Der Plattform-unabhängige *Bytecode* wird erst zur Laufzeit in die Plattform-abhängige Maschinensprache übersetzt. Bei *Java* geschieht dies nur bei Bedarf und häufig ausgeführten Programmteilen. In der *CLR* von *CLR* geschieht dies immer vollständig vor der Ausführung.

JMS *Java Message Service*. Nachrichtendienst in *Java*.

JNDI *Java Naming and Directory Interface*. Namen- und Verzeichnisdienst von Sun auf der Basis von *Java*.

JRE *Java Runtime Environment*. Laufzeitumgebung für *Java*-Programme.

JRMP *Java Remote Object Protocol*. Protokoll, das bei *RMI* den Zugriff auf ein entferntes Objekt ermöglicht.

JScript Scriptsprache entsprechend *JavaScript* mit einem leicht anderen *DOM*. Sie entspricht dem *ECMAScript*-Standard.

JSP *Java Server Pages*. Möglichkeit, dynamisch generierte Web-Seiten zu erstellen, indem der *HTML*-Code um *Java*-Code erweitert wird. Vor der Übertragung zum Browser werden diese Seiten zunächst in *Servlets* überführt, die dann die Seite produzieren.

JTA *Java Transaction API*. API für Transaktionen in *Java*.

Just-in-Time-Compiler *Siehe* **JIT**

JVM *Java Virtual Machine* Laufzeitumgebung für Java Programme.

Komponente Eine Komponente ist ein funktional in sich abgeschlossener Softwarebaustein, der über wohldefinierte Schnittstellen mit seiner Umgebung in Beziehung tritt.

Local Procedure Call *Siehe* **LPC**

LPC *Local Procedure Call*. Funktionsaufruf innerhalb der lokalen Maschine. Gegenteil von *RPC*.

Manifest Beschreibung der inneren Struktur, z.B. bei *Assemblies* oder bei *JAR*-Dateien.

Markup-Sprache Sprache zur Beschreibung von Sprachen.

Microsoft Foundation Classes *Siehe* **MFC**

Microsoft Intermediate Language *Siehe* **CIL**

Middleware Bindeglied zwischen Frond-End (Anwendungen, Browser, etc.) und Back-End (Datenbanken, Datawarehouse, etc.).

MIME-Type *Multipurpose Internet Mail Extension*. Legt den Typ des im *Internet* übertragenen Dokumentes fest.

MFC *Microsoft Foundation Classes*. Klassenbibliothek von Microsoft für die Windows-Plattform.

MSIL *Siehe* **CIL**

Multipurpose Internet Mail Extension *Siehe* **MIME-Type**

.NET Technologien von Microsoft für die Window-Plattformen. Kern-Technologie ist das *.NET-Framework*.

.NET-Framework Technologie von Microsoft zur Entwicklung von Anwendungen für die Windows-Plattform. Teile des .NET-Frameworks sind u.a. *CLR, CIL, CTS, CTS* sowie die .NET-Klassen

Netz-Adresse Eine *IP-Adresse* unterteilt sich in eine Netz-Adresse und die Nummer des Rechners in diesem Netz.

Netz-Klasse Der Nummern-Raum der *IP-Adressen* wird in Netz-Klassen unterteilt, die jeweils eine unterschiedliche Anzahl von Rechnern aufnehmen können. Man unterscheidet Class-A-, Class-B-, und Class-C-Netze.

NSAPI *Netscape Server API*. Möglichkeit, die Fähigkeiten des Netscape Servers über eine definierte Schnittstelle zu erweitern.

ONC *Open Network Computing*. *RPC*-Technologie von Sun.

Object Im Gegensatz zu prozeduralen Programmiersprachen, in denen Funktionen mit Parametern aufgerufen werden, existieren in objektorientierten Programmiersprachen Objekte, die Eigenschaften und Methoden besitzen. Objekte können Methoden und Eigenschaften von anderen Objekten erben, so dass eine Objekt-Hierarchie aufgebaut werden kann.

Object Linking and Embedding *Siehe* **OLE**

Object Management Architectur *Siehe* **OMA**

Object Management Group *Siehe* **OMG**

Object Request Broker *Siehe* **ORB**

OLE *Object Linking and Embedding*. Technologie von Micrcosoft zum Einbetten von Daten unterschiedlicher Quellen in eine Anwendung und der Möglichkeit, diese Daten durch die Ursprungsanwendung innerhalb der neuen Anwendung bearbeiten/anzeigen zu lassen.

OMA *Object Management Architectur*. Kern der *CORBA*-Spezifikation. Definiert dort *Komponente* und *Dienst*.

OMG *Object Management Group*. Non-Profit Organisation mit dem Ziel der Schaffung von Standards im Kontext verteilter Objekte.

Open Software Foundation *Siehe* **OSF**

ORB *Object Request Broker*. Stellt die Interaktionskomponente für verteilte Objekte bei *CORBA* zur Verfügung.

OSF *Open Software Foundation*. Entwickler der *DCE*-Architektur.

PE *Portable Excecutable*. Ausführbare Programm-Einheit der Windows-Plattform.

Persistenz Persistenz bezeichnet die Möglichkeit, verwendete Daten dauerhaft, d.h. über die Verwendung innerhalb eines Programms hinaus, zu speichern.

PHP Script-Sprache, mit deren Hilfe dynamische Web-Seiten erzeugt werden können.

Plug-in Definierte Möglichkeit, die Fähigkeiten eines Programms, z.B. eines Browsers, zu erweitern.

POA *Portable Object Adapter*. Objekt Adapter bei *CORBA*. Nachfolger von *BOA*.

POP3 *Post Office Protocol, Version 3*. Protokoll zum Abruf von eMail aus dem Postfach.

Port Bei der Internet-Kommunikation über *TCP/IP* und *UDP/IP* wird neben der *IP-Adresse* auch eine Port-Nummer benötigt, die den *Dienst* des angesprochenen Rechners definiert.

Portable Excecutable *Siehe* **PE**

Portable Object Adapter *Siehe* **POA**

Post Office Protocol *Siehe* **POP3**

Proxy Stellvertreter/Vermittler zwischen zwei Parteien. Beispiel: *Proxy-Server* zum Reduzieren von Web-Zugriffen oder als Proxy (Stub) bei *DCOM*.

Reflection Technologie, um die Eigenschaften eines Objektes zu erfragen und ggf. Methoden dieses Objektes auszuführen, ohne die Klassen-Definition zur Kompilierzeit besessen zu haben.

Remote Methode Invocation *Siehe* **RMI**

Remote Object Entferntes Objekt. Objekt, das nicht im gleichen Adressraum existiert und unter Umständen auf einem anderen Rechner liegt.

Remote Procedure Call *Siehe* **RPC**

RMI *Remote Methode Invocation*. Möglichkeit von *Java*, auf entfernte Objekte *(Remote Object)* zugreifen zu können.

RMI-IIOP *RMI* über *IIOP*

Root-DNS Wurzel-DNS-Server, der die Liste aller TLD-DNS-Server führt.

Routing Routing beschreibt den Vorgang der Auswahl des Weges, den Datenpakete zwischen Rechnern in unterschiedlichen Teilnetzen gehen können.

RPC *Remote Procedure Call*. Aufruf einer Funktion in einem anderen Adressraum, insbesondere einem anderen Rechner unter Zuhilfenahme von *Stub* und *Skeleton*. Gegenteil von *LPC*.

Sandbox Laufzeitumgebung von Applets innerhalb eines *Web-Browsers*

Serialisation Möglichkeit in Java, ein Objekt mit seinem inneren Zustand und all seinen Eigenschaften zu speichern oder über das Netz zu übertragen.

Server Ein Server stellt *Dienste* zur Verfügung, die von einem *Client* abgerufen werden können.

Server Side Include *Siehe* **SSI**

Servlet Web-Server-seitiges *Java*-Programm.

Servlet-Engine Stellt die Schnittstelle zwischen Web-Server und *Servlet* her und ist für den Lebenszyklus eines Servlets verantwortlich. Bekannter Vertreter ist die *Tomcat*-Servlet-Engine.

Service *Siehe* **Dienst**

SGML *Standard Generalized Markup Language*. *Markup-Sprache* zur Beschreibung von Dokumenten.

Simple Mail Transfer Protocol *Siehe* **SMTP**

Skeleton Entsprechendes Pendant zu *Stub* auf der Server-Seite, der für die eigentliche Ausführung des Methodenaufrufes verantwortlich ist.

SMTP *Simple Mail Transfer Protocol*. Protokoll zum Versenden von eMail im *Internet*

SOAP *Simple Object Access Protocol*. *RPC* über ein Internet-Protokoll (z.B. *HTTP* oder *SMTP*) mittels *XML*-kodierter Nachrichten. Wird u.a. bei den *Web Services* verwendet.

Socket Software-Schnittstelle zur Internet-Kommunikation

SSI *Server Side Include*. Erweiterung einer *HTML*-Seite um Elemente, die vor der Übertragung zum Browser erst von dem Web-Server unter Zuhilfenahme externer Programme dynamisch generiert werden.

Standard Generalized Markup Language *Siehe* **SGML**

Stub Stellvertreter-Code, um den Zugriff auf ein *entferntes Objekt* gegenüber dem aufrufenden Programm zu verbergen.

Sub-Domäne Bestandteil des Internet-Namens. Eine Sub-Domäne liegt entweder unterhalb der *TLD* oder einer anderen Sub-Domäne.

Subnet-Maske Die Subnet-Maske gibt an, welcher Teil der *IP-Adresse* die *Netz-Adresse* darstellt und welcher Teil davon die Rechner-Nummer ist.

Sub-Netz Unterteilung eines Class-A-, Class-B- oder Class-C-Netzes unter Zuhilfenahme der *Subnet-Maske* in Unter-Netze.

TCP Protokoll, das auf *IP* aufbauend einen verbindungsorientierten Datenstrom bietet.

TCP/IP *Transmission Control Protocol over Internet Protocol*. Standardprotokoll für Internet-Kommunikation.

Thread Möglichkeit in Java, nebenläufige Prozesse innerhalb derselben *Virtual Maschine* ablaufen zu lassen.

TLD *Top-Level-Domain*. Alle Rechner und *Sub-Domänen* sind unterhalb von TLD's angesiedelt. TDLs können *ccTLD* oder *gTLD* sein.

Tomcat *Servlet-Engine* des Apache-Jakarta-Projektes

Top-Level-Domain *Siehe* **TLD**

Transmission Control Protocol *Siehe* **TCP**

UDDI *Universal Description, Discovery and Integration*. Service zum Auffinden von *Web Services*. Informationen zu den Web Services werden in *White Pages* Telefonbuch, *Yellow Pages* (Branchen-Verzeichnis) und *Green Pages* (technisches Verzeichnis) abgelegt.

UDP *User Datagram Protocol*. Nicht verbindungsorientiertes Protokoll, aufbauend auf *IP*.

UDP/IP *User Datagram Protocol over Internet Protocol*

Uniform Resource Locator *Siehe* **URL**

Universal Description, Discovery and Integration *Siehe* **UDDI**

Universal Resource Identifier *Siehe* **URI**

URI *Universal Resource Identifier*. Allgemeine Spezifikation einer Ressource im *Internet*.

URL *Uniform Resource Locator*. Verweis auf eine Ressource im *Internet*. Spezialform von *URI*, im Allgemeinen als *Internet-Adresse* bezeichnet. Abhängig vom Protokoll, mit dem die Ressource angesprochen werden soll, können *Port-Nummer*, Pfad, Dokumenten-Name und weitere Informationen Bestandteil der URL sein.

User Datagram Protocol *Siehe* **UDP**

VB *Visual Basic*. Programmiersprache.

Virtual Machine *Siehe* **VM**

Visual Basic *Siehe* **VB**

VM *Virtual Machine*. Laufzeitumgebung, z.B. als *JVM* für ein *Java*-Programm. Wird für jedes Betriebssystem benötigt, auf dem ein Java-Programm laufen soll.

W3C *World Wide Web Consortium*

Web *Siehe* **WWW**

Web-Browser Programm, das Dokumente, die von *Web-Servern* übertragen werden, visualisiert, i.A. *HyperText-Dokumente*. Durch die Verfolgung der *Links* innerhalb der Dokumente entsteht das so genannte „Surfen“ im *Internet*.

Web-Server *Server* im *Internet*, der den *HTTP*-Service zur Verfügung stellt, um Dokumente (z.B. *HyperText-Dokumente*) u.a. zu *Web-Browsern* zu übertragen, die diese dann visualisieren.

Web Service Das W3C*W3C* versteht Folgendes unter Web Service[1]:

> Ein Web Service ist eine Software, die durch eine URL identifiziert ist (RFC2396) und deren öffentliche Schnittstellen und Bindungen durch XML definiert werden. Ihre Definitionen können durch andere Software-Systeme ermittelt werden. Diese Systeme können mit dem Web Service in der vorgeschriebenen Art interagieren, wobei über Internet-Protokolle versandte XML-basierende Nachrichten verwendet werden.

Web Service Description Language *Siehe* **WSDL**

Web-Site Logische Zusammenfassung von Dokumenten im Internet zu einer Web-Site. Im Allgemeinen wird die Web-Site mit einer Initialen *URL* identifiziert, über die alle Dokumente der Site durch Verfolgung von *Links* innerhalb der *HyperText-Dokumente* erreicht werden können.

World Wide Web *Siehe* **WWW**

WSDL *Web Service Description Language*. Beschreibung von *Web Services* in einer speziellen *XML*-Sprache.

WWW *World-Wide Web*. Das *W3C* definiert das WWW folgendermaßen[2]: Das *World Wide Web (Web)* ist ein Netzwerk von Informationsressourcen. Das Web ist auf drei Mechanismen angewiesen, um diese Ressourcen leicht für die größtmögliche Zielgruppe verfügbar zu machen:

1. Ein einheitliches Namensschema, um Ressourcen im Web aufzufinden, z.B. *URIs*
2. Ein Protokoll für den Zugriff auf benannte Ressourcen Web, z.B. *HTTP*
3. Hypertexte zum einfachen Navigieren unter den Ressourcen, z.B. *HTML*

XDR *eXternal Data Representation*. Rechner-unabhängiges Datenformat. Wird bei SUNs *RPC* verwendet, um Parameter und Ergebnisse zu kodieren.

XML *eXtensible Markup Language*. Eine *Markup-Sprache*. Zur Zeit das Dokumenten-Austauschformat im Internet.

XML Namespace Definiert den Namensraum von XML-Dokumenten.

XML Schema Möglichkeit, die Gültigkeit von *XML*-Dokumenten festzulegen. Besser geeignet als *DTD*, da unter anderem selber in XML beschrieben.

XML-RPC Einfaches *RPC* über *HTTP* mittels *XML*-kodierter Nachrichten.

[1] Übersetzung aus Brown und Haas, 2002
[2] Übersetzung aus Raggett u. a., 1999

Literaturverzeichnis

[Adam u. a. 1997] ADAM ; HOFER ; KALLER ; KAUL ; LANDWEHR: Corba 2.0 / Steinbeis-Transferzentrum Softwaretechnik Esslingen. URL `http://www.stz-softwaretechnik.de/~dk/corbascr.pdf`. – Zugriffsdatum: 19.03.2003, 1997. – Forschungsbericht

[Andersen u. a. 2000] ANDERSEN, A. ; BLAIR, G.S. ; ELIASSEN, F.: *OOOP: A refective component-based middleware*. NIK 2000. November 2000. – URL `http://citeseer.nj.nec.com/andersen00oopp.html`

[Andersen 2002] ANDERSEN, Anders: *OOOP - A Reflective Middleware Platform including Quality of Service Management*, University of Tromsø, Norwegen, PhD Dissertation, Februar 2002

[Apache Software Foundation 2003] APACHE SOFTWARE FOUNDATION: *Apache XML-RPC*, Februar 2003. – URL `http://ws.apache.org/xmlrpc/`. – Zugriffsdatum: 19.03.2003

[Beer u. a. 2003] BEER, Wolfgang ; BIRNGRUBER, Dietrich ; MÖSSENBÖCK, Hanspeter ; WÖSS, Albrecht: *Die .NET- Technologie. Grundlagen und Anwendungsprogrammierung*. dpunkt Verlag, 2003. – ISBN 3-89864-174-0

[Bellwood u. a. 2002] BELLWOOD, Tom ; CLÉMENT, Luc ; EHNEBUSKE, David ; HATELY, Andrew ; HATELY, Andrew ; HUSBAND, Yin L. ; JANUSZEWSKI, Karsten ; LEE, Sam ; MCKEE, Barbara ; MUNTER, Joel ; RIEGEN, Claus von: UDDI Version 3.0 / UDDI. URL `http://uddi.org/pubs/uddi-v3.00-published-20020719.pdf`. – Zugriffsdatum: 19.03.2003, Juli 2002. – Forschungsbericht

[Bengel 2002] BENGEL, Günther: *Verteilte Systeme*. 2. Friedrich Vieweg & Sohn, 2002. – ISBN 3-528-15738-0

[Bhatti u. a. 1998] BHATTI, Nina T. ; HILTUNEN, Matti A. ; SCHLICHTING, Richard D. ; CHIU, Wanda: Coyote: A system for Constructing Fine-Grain Configurable Communication Services. In: *ACM Transactions on Computer Systems* 16 (1998), Nr. 4, S. 321–366. – URL `http://citeseer.nj.nec.com/article/bhatti98coyote.html`

[Biron und Malhotra 2001] BIRON, Paul V. ; MALHOTRA, Ashok: XML Schema Part 2: Datatypes / World Wide Web Consortium (W3C). URL `http://www.w3.org/TR/xmlschema-2/`. – Zugriffsdatum: 30.04.2003, Mai 2001 (REC-xmlschema-2-20010502). – W3C Recommendation

[Birrel und Nelson 1984] BIRREL, Andrew D. ; NELSON, Bruce J.: Implementing Remote Procedure Calls. In: *ACM Transactions on Computer Systems* 2 (1984), Februar, Nr. 1, S. 39–59

[Blair u. a. 1998] BLAIR, Gordon S. ; COULSON, G. ; ROBIN, P. ; PAPATHOMAS, M.: An architecture for next generation middleware. In: *Proceedings of the IFIP International Conference on Distributed Systems Platforms and Open Distributed Processing*. London : Springer-Verlag, 1998. – URL `http://citeseer.nj.nec.com/blair98architecture.html`

[Bray u. a. 1999] BRAY, Tim ; HOLLANDER, Dave ; LAYMAN, Andrew: Namespaces in XML / World Wide Web Consortium (W3C). URL `http://www.w3.org/TR/REC-xml-names`. – Zugriffsdatum: 30.04.2003, Januar 1999 (REC-xml-names-19900114). – W3C Recommendation

[Bray u. a. 2000] BRAY, Tim ; PAOLI, Jean ; SPERBERG-MCQUEEN, C. M. ; MALER, Eva: Extensible Markup Language (XML) 1.0 (Second Edition) / World Wide Web Consortium (W3C). URL `http://www.w3.org/TR/REC-xml`. – Zugriffsdatum: 30.04.2003, Oktober 2000 (REC-xml-20001006). – W3C Recommendation

[Brown und Haas 2002] BROWN, Allen ; HAAS, Hugo: Web Services Glossary / World Wide Web Consortium (W3C). URL `http://www.w3.org/TR/ws-gloss/`. – Zugriffsdatum: 30.04.2003, November 2002 (WD-ws-gloss-20021114). – W3C Working Draft

[Byous 1998] BYOUS, Jon: *Java Technology: An Early History*, 1998. – URL `http://java.sun.com/features/1998/05/birthday.html`. – Zugriffsdatum: 19.03.2003

[Campione und Walrath 1998] CAMPIONE, Mary ; WALRATH, Kathy: *The Java Tutorial - Object-Oriented Programming for the Internet*. 2. Addison-Wesley, 1998. – ISBN 0-201-31007-4

[Champion u. a. 2002] CHAMPION, Michael ; FERRIS, Chris ; NEWCOMER, Eric ; ORCHARD, David: Web Services Architecture / World Wide Web Consortium (W3C). URL `http://www.w3.org/TR/ws-arch/`. – Zugriffsdatum: 30.04.2003, November 2002 (WD-ws-arch-20021114). – W3C Working Draft

[Cinnini u. a. 2003] CINNINI, Roberto ; GUDGIN, Martin ; MOREAU, Jean-Jaques ; WEERAWARANA, Sanjiva: Web Services Description Language (WSDL) Version 1.2 / World Wide Web Consortium (W3C). URL `http://www.w3.org/TR/wsdl12/`. – Zugriffsdatum: 30.04.2003, März 2003 (WD-wsdl12-20030303). – W3C Working Draft

[Clarke u. a. 2001] CLARKE, M. ; BLAIR, C.S. ; COULSON, G. ; PALAVANTZAS, N.: An Efficient Component Model for the Construction of Adaptive Middleware. In: GUERRAOUI, R. (Hrsg.): *Middleware 2001 : IFIP/ACM International Conference on Distributed Systems Platforms Heidelberg, Germany, November 12-16, 2001* Bd. LNCS 2218, Spriger-Verlag, November 2001. – ISSN 0302-9743

[Coulson und Baichoo 1999] COULSON, G. ; BAICHOO, S.: A Distributed Object Platform for Multimedia Applications. In: *Proceedings of the IEEE Multimedia Systems, Florence, Italy*, URL `http://citeseer.nj.nec.com/coulson99distributed.html`, Juni 1999, S. 122–126. – ISBN 0-7695-0253-9

[Coulson u. a. 2002a] COULSON, G. ; BAICHOO, S. ; MOONIAN, O.: A retrospective on the design of the GOPI middleware platform. In: *Multimedia Systems* 8 (2002), Dezember, Nr. 5, S. 340–352. – ISSN 0942-4962

[Coulson u. a. 2002b] COULSON, G. ; BLAIR, G.S. ; CLARKE, M. ; PARLAVANTZAS, N.: The design of a configurable and reconfigurable middleware platform. In: *Distributed Computing* 15 (2002), April, Nr. 2, S. 109–126. – ISSN 0178-2770

[Coulson 2002] COULSON, Geoff: What is Reflective Middleware? In: *IEEE Distribute Systems Online* (2002), Oktober. – URL `http://dsonline.computer.org/middleware/RMarticle1.htm`. – Zugriffsdatum: 30.07.2003. – ISSN 1541-4922

[Deitel u. a. 2003] DEITEL, Harvey M. ; DEITEL, Paul J. ; DUWALDT, B: *Web Services*. Prentice Hall International, 2003. – ISBN 0-13-046135-0

[Denninger und Peters 2000] DENNINGER, Stefan ; PETERS, Ingo: *Enterprise JavaBeans*. Addison-Wesley Verlag, 2000. – ISBN 3-8273-1534-5

[Eddon und Eddon 1998] EDDON, Guy ; EDDON, Henry: *Inside Distributed COM*. Microsoft Press International, 1998. – ISBN 1-57231-849-X

[Eliassen u. a. 2000] ELIASSEN, F. ; PLAGEMANN, T. ; RAFAELSEN, H.O.: *MULTE-ORB: Adaptive QoS Aware Binding*. April 2000. – URL `http://citeseer.nj.nec.com/393530.html`

[Eliassen u. a. 1999] ELIASSEN, Frank ; ANDERSEN, Anders ; BLAIR, Gordon S. ; COSTA, Fabio ; COULSON, Geoff ; GOEBEL, Vera ; HANSEN Øivind ; KRISTENSEN, Tom ; PLAGEMANN, Thomas ; RAFAELSEN, Hans O. ; SAIKOSKI, Katia B. ; YU, Weihai:

Next Generation Middleware: Requirements, Architecture, and Prototypes. In: *Proceedings of 7th Workshop on Future Trends of Distributed Computing Systems (FTDCS'99), Cape Town, South Africa*. Cape Town, South-Africa : IEEE, Dezember 1999, S. 60–68. – ISBN ISBN 0-7695-0468-X

[Eliassen u. a. 2002] ELIASSEN, Frank ; PLAGEMANN, Thomas ; HAFSKJOLD, Brita ; KRISTENSEN, Tom ; RAFAELSEN, Hans O. ; MACDONALD, Robert H.: W QoS management in the MULTE-ORB. In: *IEEE Distribute Systems Online* (2002), September. – URL `http://dsonline.computer.org/middleware/articles/dsonline-MULTE-ORB.html`. – Zugriffsdatum: 18.08.2003. – ISSN 1541-4922

[Fallside 2001] FALLSIDE, David C.: XML Schema Part 0: Primer / World Wide Web Consortium (W3C). URL `http://www.w3.org/TR/xmlschema-0/`. – Zugriffsdatum: 30.04.2003, Mai 2001 (REC-xmlschema-0-20010502). – W3C Recommendation

[Farley 1998] FARLEY, Jim: *Java Distributed Computing*. O'Reilly & Associates, Januar 1998. – ISBN 1-56592-206-9

[Fitzpatrick u. a. 1998] FITZPATRICK, T. ; BLAIR, G. ; COULSON, G. ; DAVIES, N. ; ROBIN, P.: *Supporting Adaptive Multimedia Applications Through Open Bindings*. Proceedings of the 4. International Conference on Configurable Distributed Systems (ICCDS'98), Annapolis, Maryland, USA. Mai 1998. – URL `http://citeseer.nj.nec.com/fitzpatrick98supporting.html`

[Flanagan 2003] FLANAGAN, David: *Java in a Nutshell - Deutsche Ausgabe für Java 1.4*. 4. O'Reilly, 2003. – ISBN 3-89721-332-X

[Gottlieb und Peterson 2002] GOTTLIEB, Yitzchak ; PETERSON, Larry: A Comparative Study of Extensible Routers. In: *2002 IEEE Open Architectures and Network Programming Proceedings*. New York, USA, Juni 2002, S. 51–62

[Gottschalk u. a. 2002] GOTTSCHALK, Karl ; GRAHAM, Stephen ; KREGER, Heather ; SNELL, James: Introduction to Web Services Architecture. In: *IBM Systems Journal* 41 (2002), Nr. 2, S. 170–170

[Gudgin u. a. 2002a] GUDGIN, Martin ; HADLEY, Marc ; MENDELSOHN, Noah ; MOREAU, Jean-Jacques ; NIELSEN, Henrik F.: SOAP Version 1.2 Part 1: Messaging Framework / World Wide Web Consortium (W3C). URL `http://www.w3.org/TR/soap12-part1/`. – Zugriffsdatum: 30.04.2003, Juni 2002 (WD-soap12-part1-20020626). – W3C Last Call Working Draft

[Gudgin u. a. 2002b] GUDGIN, Martin ; HADLEY, Marc ; MENDELSOHN, Noah ; MOREAU, Jean-Jacques ; NIELSEN, Henrik F.: SOAP Version 1.2 Part 2: Adjuncts / World

Wide Web Consortium (W3C). URL `http://www.w3.org/TR/soap12-part2/`. – Zugriffsdatum: 30.04.2003, Juni 2002 (WD-soap12-part2-20020626). – W3C Last Call Working Draft

[Haas und Orchard 2002] HAAS, Hugo ; ORCHARD, David: Web Services Architecture Usage Scenarios / World Wide Web Consortium (W3C). URL `http://www.w3.org/TR/ws-arch-scenarios/`. – Zugriffsdatum: 30.04.2003, Juli 2002 (WD-ws-arch-scenarios-20020730). – W3C Working Draft

[Haffner 2001] HAFFNER, Ernst-Georg: *Request-Prediction and Hyperlink-Proposals - Methodologies and Mathematics behind Web-Applications*, Fachbereich IV, Informatik, Universität Trier, Dissertation, 2001

[Haffner u. a. 2000a] HAFFNER, Ernst-Georg ; ROTH, Uwe ; ENGEL, Thomas ; MEINEL, Christoph: Advanced Techniques for Analyzing Web Logs. In: GRAHAM, Peter (Hrsg.) ; MAHESWARAN, Muthucumaru (Hrsg.): *Proceedings of the 1st International Conference on Internet Computing, IC'2000, 26-29 June 2000, Las Vegas, Nevada, USA*, CSREA Press, Juni 2000, S. 71–78. – ISBN 1-892512-65-3

[Haffner u. a. 2000b] HAFFNER, Ernst-Georg ; ROTH, Uwe ; ENGEL, Thomas ; MEINEL, Christoph: Optimizing Requests for the Smart Data Server. In: HAMZA, M.H. (Hrsg.): *Proceeding of the Eighteenth IASTED International Conference Applied Informatics, AI2002, February 14-17, 2000, Innsbruck, Austria*, IASTED/ACTA Press, Februar 2000. – ISBN 0-88986-280-X

[Haffner u. a. 2001] HAFFNER, Ernst-Georg ; ROTH, Uwe ; HEUER, Andreas ; MEINEL, Christoph: Managing Distributed Personal Firewalls with Smart Data Servers. In: FOWLER, Wendy (Hrsg.) ; HASEBROOK, Joachim (Hrsg.): *Proceedings of WebNet 2001 - World Conference on the WWW and Internet, October 23-27, 2001, Orlando, Florida, USA*, AACE, Oktober 2001, S. 466–471. – ISBN 1-880094-46-0

[Øyvind Hanssen und Eliassen 1999] HANSSEN Øyvind ; ELIASSEN, Frank: A Framework for Policy Binding. In: *Proceedings of the International Symposium on Distributed Objects and Applications, Edinburgh, UK*, IEEE Computer Society, September 1999, S. 2–11. – URL `http://www.computer.org/proceedings/doa/0182/01820002abs.htm`. – Zugriffsdatum: 13.08.2003

[Harold 1999] HAROLD, Elliotte R.: *Java I/O*. O'Reilly, 1999. – ISBN 1-56592-485-1

[Hayton 1999] HAYTON, Richard: ANSA FlexiNet Architecture / Cytrix Systems. Cambridge, Februar 1999. – Architecture Report

[Hayton u. a. 1999] HAYTON, R.J. ; BURSELL, M.H. ; DONALDSON, D.I. ; HARWOOD, W. ; HERBERT, A.J.: Mobile Java Objects. In: *Distributed Systems Engeneering* (1999), März, Nr. 6, S. 51–60

[Heuer 2002] HEUER, Andreas: *Web-Präsenz-Management im Unternehmen*, Fachbereich IV, Informatik, Universität Trier, Dissertation, 2002

[Hoffman u. a. 2001] HOFFMAN, Kevin ; GABRIEL, Jeff ; GOSNELL, Denise: *Professional .NET Framework*. Wrox Press, 2001. – ISBN 1-86100-556-3

[Hofmann u. a. 2001] HOFMANN, Johann ; JOBST, Fritz ; SCHABENBERGER, Roland: *Programmieren mit COM und CORBA*. Hanser Fachbuch, 2001. – ISBN 3-446-21479-8

[Huang u. a. 2002] HUANG, Wanjun ; ROTH, Uwe ; MEINEL, Christoph: Improvement to the Smart Data Server with SOAP. In: MASTORAKIS, Nikos E. (Hrsg.) ; KLUEV, Vitaliy V. (Hrsg.) ; KORUGA, Djuro (Hrsg.): *2nd WSEAS International. Conference on Multimedia, Internet and Video Technologies, ICOMIV 2002, Sep 25-29, 2002, Skiathos, Greece* WSEAS (Veranst.), WSEAS Press, September 2002, S. 1141–1145. – ISBN 960-8052-688

[Huang u. a. 2003] HUANG, Wanjun ; ROTH, Uwe ; MEINEL, Christoph: A Flexible Middleware Platform With Piped Workflow. In: *Proceedings of On The Move to Meaningful Internet Systems 2003, OTM'03 Workshops, Sicily, Italy* Bd. LNCS 2889, Springer, November 2003, S. 950–959

[Hutchinson und Peterson 1991] HUTCHINSON, N. C. ; PETERSON, L. L.: The x-Kernel: An Architecture for Implementing Network Protocols. In: *IEEE Transactions on Software Engineering* 17 (1991), Nr. 1, S. 64–76. – URL `http://citeseer.nj.nec.com/hutchinson91xkernel.html`

[ISO/IEC 10746-1 1998] Open Distributed Processing - Reference Modell - Part 1: Overview / International Organization for Standardization. URL `http://www.iso.ch/iso/en/CatalogueDetailPage.CatalogueDetail?CSNUMBER=20696`. – Zugriffsdatum: 12.08.2003, 1998 (ISO/IEC 10746-1:1998). – Forschungsbericht

[ISO/IEC 10746-2 1996] Open Distributed Processing - Reference Modell - Part 2: Foundations / International Organization for Standardization. URL `http://www.iso.ch/iso/en/CatalogueDetailPage.CatalogueDetail?CSNUMBER=18836`. – Zugriffsdatum: 12.08.2003, 1996 (ISO/IEC 10746-2:1996). – Forschungsbericht

[ISO/IEC 10746-3 1996] Open Distributed Processing - Reference Modell - Part 3: Architecture / International Organization for Standardization. URL `http://www.iso.ch/iso/en/CatalogueDetailPage.CatalogueDetail?CSNUMBER=20697`. – Zugriffsdatum: 12.08.2003, 1996 (ISO/IEC 10746-3:1996). – Forschungsbericht

[ISO/IEC 10746-4 1998] Open Distributed Processing - Reference Modell - Part 4: Architectural Semantics Amedment / International Organization for Standardization. URL `http://www.iso.ch/iso/en/CatalogueDetailPage.`

CatalogueDetail?CSNUMBER=20698. – Zugriffsdatum: 12.08.2003, 1998 (ISO/IEC 10746-4:1998). – Forschungsbericht

[ISO/IEC 13235-1 1998] Open Distributed Processing - Trading function / International Organization for Standardization. URL http://www.iso.ch/iso/en/CatalogueDetailPage.CatalogueDetail?CSNUMBER=20698. – Zugriffsdatum: 12.08.2003, 1998 (ISO/IEC 13235-1:1998). – Forschungsbericht

[Kohler u. a. 2000] KOHLER, Eddie ; MORRIS, Robert ; CHEN, Benjie ; JANNOTTI, John ; KAASHOEK, M. F.: The Click Modular Router. In: *ACM Transactions on Computer Systems* 18 (2000), August, Nr. 3, S. 263–297

[Kon u. a. 2000a] KON, Fabio ; CAMPBELL, Roy H. ; MICKUNAS, M. D. ; NAHRSTEDT, Klara ; BALLESTEROS, Francisco J.: 2K: A Distributed Operating System for Dynamic Heterogeneous Environments. In: *Proceedings of the 9th IEEE International Symposium on High Performance Distributed Computing (HPDC'9)*. Pittsburgh, USA, August 2000, S. 201–208

[Kon u. a. 1999] KON, Fabio ; CARVALHO, Dulcineia ; CAMPBELL, Roy: Automatic Configuration in the 2K Operating System. In: *Proceedings of the ECOOP'99 Workshop on Object Orientation and Operating Systems*. Lissabon, Juni 1999, S. 10–14

[Kon u. a. 2002] KON, Fabio ; COSTA, Fábio ; CAMPBELL, Roy ; BLAIR, Gordon: The Case for Reflective Middleware. In: *Communications of the ACM* 45 (2002), Juni, Nr. 6, S. 33–38

[Kon u. a. 2000b] KON, Fabio ; GILL, Binny ; ANAND, Manish ; CAMPBELL, Roy H. ; MICKUNAS, M. D.: Secure Dynamic Reconfiguration of Scalable CORBA Systems with Mobile Agents. In: *ASA/MA*, URL http://citeseer.nj.nec.com/kon00secure.html, 2000, S. 86–98

[Kon u. a. 2000c] KON, Fabio ; ROMÁN, Manuel ; LIU, Ping ; MAO, Jina ; YAMANE, Tomonori ; MAGALHÃES, Luiz C. ; CAMPBELL, Roy H.: Monitoring, Security, and Dynamic Configuration with the dynamicTAO Reflective ORB. In: *Proceedings of the IFIP/ACM International Conference on Distributed Systems Platforms and Open Distributed Processing (Middleware'2000)*. New York : Springer-Verlag, April 2000 (LNCS 1795), S. 121–143. – URL http://citeseer.nj.nec.com/kon00monitoring.html

[Koster u. a. 2001] KOSTER, R. ; BLACK, A. P. ; HUANG, J. ; WALPOLE, J. ; PU, C.: Thread Transparency in Information Flow Middleware. In: *Proceedings of Middleware'01 (IFIP/ACM International Conference on Distributed Systems Platforms and Open Distributed Processing)*, Springer Verlag, November 2001

[Koster 2002] KOSTER, Rainer: *A Middleware Platform for Information Flows*, Universität Kaiserslautern, Dissertation, Juli 2002

[Koster u. a. 2003] KOSTER, Rainer ; BLACK, Andrew P. ; HUANG, Jie ; WALPOLE, Jonathan ; PU, Calton: Thread transparency in information flow middleware. In: *Software: Practice and Experience, John Wiley & Sons* 33 (2003), Februar, Nr. 4, S. 321–349

[Kramp und Coulson 1999] KRAMP, T. ; COULSON, G.: The Design of a Flexible Communications Framework for Next-Generation Middleware / Department of Computer Science, University of Kaiserslautern, Germany. Oktober 1999 (12/99). – Forschungsbericht. SFB 501 Report

[Kramp und Koster 1999] KRAMP, T. ; KOSTER, R.: Flexible Event-Based Threading for QoS-supporting Middleware. In: *Proceedings of the Second International Working Conference on Distributed Applications and Interoperable Systems (DAIS)* IFIP (Veranst.), Juli 1999

[Kristensen und Plagemann 2000] KRISTENSEN, Tom ; PLAGEMANN, Thomas: Enabling Flexible QoS Support in the Object Request Broker COOL. In: *International Workshop on Distributed Real-Time Systems (IWDRS 2000), Taipei, Taiwan*, URL `http://citeseer.nj.nec.com/kristensen00enabling.html`, April 2000, S. B45–B54

[Kuhns u. a. 1999] KUHNS, Fred ; SCHMIDT, Douglas C. ; LEVINE, David L.: The Design and Performance of a Real-Time I/O Subsystem. In: *IEEE Real Time Technology and Applications Symposium*, URL `http://citeseer.nj.nec.com/article/kuhns99design.html`, 1999, S. 154–163

[Laifer 1998] LAIFER, Roland: *Komponenten und Konzepte des Distributed Computing Environment (DCE)*. Rechenzentrum der Uni Karlsruhe. August 1998. – URL `http://www.uni-karlsruhe.de/~rz54/Dce/Adm/dce_konzepte980811.ps.gz`. – Zugriffsdatum: 22.08.2003

[Laurent 2001] LAURENT, Simon S.: *Programming Web Services with XML-RPC*. O'Reilly & Associates, 2001. – ISBN 0-596-00119-3

[Lerner u. a. 2000] LERNER, Michah ; VANECEK, George ; VIDOVIC, Nino ; VRSALOVIC, Dado: *Middleware Networks*. Kluwer Academic Publisher, 2000. – ISBN 0-7923-7840-7

[Lindblad u. a. 1994] LINDBLAD, Christopher ; WETHERALL, David ; TENNENHOUSE, David L.: The VuSystem: A Programming System for Visual Processing of Digital Video. In: *ACM Multimedia*, URL `http://citeseer.nj.nec.com/lindblad94vusystem.html`, 1994, S. 307–314

[Liu u. a. 1999] LIU, Xiaoming ; KREITZ, Christoph ; RENESSE, Robbert van ; HICKEY, Jason ; HAYDEN, Mark ; BIRMAN, Kenneth P. ; CONSTABLE, Robert L.: Building reliable, high-performance communication systems from components. In: *ACM Symposium on Operating Systems Principles (SOSP '99)*, URL `http://citeseer.nj.nec.com/liu99building.html`, 1999, S. 80–92

[Magee u. a. 1994a] MAGEE, Jeff ; DULAY, Naranker ; KRAMER, Jeff: A Constructive Development Environment for Parallel and Distributed Programs. In: *Proceedings of the International Workshop on Configurable Distributed Systems*. Pittsburgh, USA, März 1994. – URL `http://citeseer.nj.nec.com/magee94constructive.html`

[Magee u. a. 1994b] MAGEE, Jeff ; DULAY, Naranker ; KRAMER, Jeff: REGIS: A Constructive Development Environment for Distributed Programs. In: *IOP/IEE/BCS Distributed Systems Engineering* 1 (1994), Nr. 5, S. 304–315. – URL `http://citeseer.nj.nec.com/magee94regis.html`

[McCanne u. a. 1997] MCCANNE, S. ; BREWER, E. ; KATZ, R. ; ROWE, L. ; AMIR, E. ; CHAWATHE, Y. ; COOPERSMITH, A. ; MAYER-PATEL, K. ; RAMAN, S. ; SCHUETT, A. ; SIMPSON, D. ; SWAN, A. ; TUNG, T. L. ; WU, D. ; SMITH, B: Toward a Common Infrastucture for Multimedia-Networking Middleware. In: *Proceedings of 7th. Intl. Workshop on Network and Operating Systems Support for Digital Audio and Video (NOSSDAV'97)*. St. Louis, Missouri, USA, Mai 1997, S. 39–49. – URL `http://citeseer.nj.nec.com/mccanne97toward.html`

[McLaughlin 2001] MCLAUGHLIN, Brett: *Java & XML*. 2. O'Reilly & Associates, August 2001. – ISBN 0-596-00197-5

[Microsoft Corporation 2002] MICROSOFT CORPORATION: *.NET Development*, 2002. – URL `http://msdn.microsoft.com/library/default.asp?url=/nhp/default.asp?contentid=28000519`. – Zugriffsdatum: 19.03.2003

[Mitev 2003] MITEV, Martin: *Konzeption und Implementierung eines DatabaseExplorers: Entwicklung einer Java-Klassenbibliothek für den herstellerunabhängigen Zugriff auf SQL-Datenbanken*, Unversität Trier, Diplomarbeit, April 2003

[Mitra 2002] MITRA, Nilo: SOAP Version 1.2 Part 1: Primer / World Wide Web Consortium (W3C). URL `http://www.w3.org/TR/soap12-part0/`. – Zugriffsdatum: 30.04.2003, Juni 2002 (WD-soap12-part0-20020626). – W3C Last Call Working Draft

[Monson-Haefel 2000] MONSON-HAEFEL, Richard: *Enterprise JavaBeans*. O'Reilly &Associates, März 2000. – ISBN 1-56592-869-5

[Moreau und Schlimmer 2003] MOREAU, Jean-Jaques ; SCHLIMMER, Jeffrey: Web Services Description Language (WSDL) Version 1.2: Bindings / World Wide Web Consortium (W3C). URL http://www.w3.org/TR/wsdl12-bindings/. – Zugriffsdatum: 30.04.2003, Januar 2003 (WD-wsdl12-bindings-20030124). – W3C Working Draft

[Mosberger 1997] MOSBERGER, D.: *Scout: A Path-based Operating System*, Department of Computer Science, University of Arizona, USA, PhD Dissertation, Juli 1997

[Mosberger und Peterson 1996] MOSBERGER, David ; PETERSON, Larry L.: Making Paths Explicit in the Scout Operating System. In: *Operating Systems Design and Implementation*, URL http://citeseer.nj.nec.com/mosberger96making.html, 1996, S. 153–167

[Mungee u. a. 1999] MUNGEE, S. ; SURENDRAN, N. ; SCHMIDT, D.C. ; KRISHNAMURTHY, Y.: *The Design and Performance of a CORBA Audio/Video Streaming Service*. Januar 1999. – URL http://citeseer.nj.nec.com/mungee99design.html

[OMG 2000] OMG: *Audio/Video Streams, Version 1.0*, Januar 2000. – URL http://www.omg.org/docs/formal/00-01-03.pdf. – Zugriffsdatum: 13.08.2003

[OMG 2002] OMG: *Common Object Request Broker Architecture: Core Specification (Version 3.0)*, Dezember 2002. – URL ftp://ftp.omg.org/pub/docs/formal/02-12-06.pdf. – Zugriffsdatum: 19.03.2003

[Orfali und Harkey 1997] ORFALI, Robert ; HARKEY, Dan: *Client/Server Programming with JAVA and CORBA*. Wiley Computer Publisher, 1997. – ISBN 0-471-16351-1

[Parlavantzas u. a. 2000] PARLAVANTZAS, N. ; COULSON, G. ; CLARKE, M. ; BLAIR, G. ; CAZZOLA, W. (Hrsg.) ; CHIBA, S. (Hrsg.) ; LEDOUX, T. (Hrsg.): *Towards a Reflective Component Based Middleware Architecture*. Juni 2000. – URL http://citeseer.nj.nec.com/331827.html

[Raggett u. a. 1999] RAGGETT, Dave ; LE HORS, Arnaud ; JACOBS, Ian: HTML 4.01 Specification / World Wide Web Consortium (W3C). URL http://www.w3.org/TR/html401/. – Zugriffsdatum: 30.04.2003, Dezember 1999 (REC-html401-19991224). – W3C Recommendation

[Raymond 1998] RAYMOND, Kerry: *Reference Modell of Open Distributed Processing (RM-ODP): Introduction*. Mai 1998. – URL http://www.dstc.edu.au/Research/Projects/ODP/papers/icodp95.ps. – Zugriffsdatum: 12.08.2003

[Renesse u. a. 1996] RENESSE, Robbert V. ; BIRMAN, Kenneth P. ; GLADE, Bradford B. ; GUO, Katie ; HAYDEN, Mark ; HICKEY, Takako ; MALKI, Dalia ; VAYSBURD, Alex ; VOGELS, Werner: Horus: A Flexible Group Communications Sys-

tem. In: *Communications of the ACM* 39 (1996), April, Nr. 4, S. 76–83. – URL `http://citeseer.nj.nec.com/renesse96horus.html`

[Richter 2001] RICHTER, Jeffrey: *Microsoft .NET Framework Programmierung*. Microsoft Press Deutschland, 2001. – ISBN 3-86063-650-2

[Román u. a. 1999] ROMÁN, Manuel ; KON, Fabio ; CAMPBELL, Roy H.: Design and Implementation of Runtime Reflection in Communication Middleware: the *dynamicTAO* Case. In: *Proceedings of the ICDCS'99 Workshop on Middleware*. Austin, Texas, USA, Juni 1999, S. 122–127

[Rosenberry 1992] ROSENBERRY, Ward: *Understanding DCE*. O'Reilly & Associates, 1992. – ISBN 1-56592-005-8

[Roth 2000a] ROTH, Jörg: *Entwicklungs- und Laufzeituntertützung für synchrone Groupware*. dissertation.de, 2000. – ISBN 3-89825-148-9

[Roth 2000b] ROTH, Uwe: Der Smart Data Server (SDS) - Eine modulare verteilte Middle-Tier-Architektur. In: *Informatiktage 2000, Fachwissenschaftlicher Informatik-Kongreß, 27. und 28. Oktober 2000 im Neuen Kloster Bad Schussenried, Deutschland* GI (Veranst.), GI, Oktober 2000, S. 165–167. – ISBN 3-920560-17-7

[Roth 2000c] ROTH, Uwe: Smart Data Server (SDS) für kommunale Selbstverwaltung. In: MEINEL, Christoph (Hrsg.) ; ENGEL, Thomas (Hrsg.): *Tätigkeitsbericht 2000, Institut für Telematik, e.V.* 2000, S. 31

[Roth 2001] ROTH, Uwe: Web-basiertes Persoalmanagement mit dem Smart Data Server. In: MEINEL, Christoph (Hrsg.) ; ENGEL, Thomas (Hrsg.): *Tätigkeitsbericht 2001, Institut für Telematik, e.V.* 2001, S. 24–25

[Roth u. a. 2000] ROTH, Uwe ; ENGEL, Thomas ; MEINEL, Christoph: Improving the Quality of Information-Flow with the Smart Data Server. In: GRAHAM, Peter (Hrsg.) ; MAHESWARAN, Muthucumaru (Hrsg.): *Proceedings of the 1st International Conference on Internet Computing, IC'2000, 26-29 June 2000, Las Vegas, Nevada, USA*, CSREA Press, Juni 2000, S. 353–357. – ISBN 1-892512-65-3

[Roth u. a. 1998] ROTH, Uwe ; HAFFNER, Ernst-Georg ; ENGEL, Thomas ; MEINEL, Christoph: Smart Data Server (SDS) - Modularer Aufbau und verteilte Funktionalität / Institut für Telematik e.V., Trier. 1998 (Preprint 98-14). – Preprint. ISSN 1433-0106

[Roth u. a. 1999a] ROTH, Uwe ; HAFFNER, Ernst-Georg ; ENGEL, Thomas ; MEINEL, Christoph: An Approach to Distributed Functionality - The Smart Data Server. In: BRA, Paul de (Hrsg.) ; LEGGETT, John (Hrsg.): *Proceedings of WebNet 99 - World Conference on the WWW and Internet, October 24-30, 1999, Honolulu, Hawaii, USA* AACE (Veranst.), AACE, Oktober 1999, S. 931–936. – ISBN 1-880094-36-3

[Roth u. a. 1999b] ROTH, Uwe ; HAFFNER, Ernst-Georg ; ENGEL, Thomas ; MEINEL, Christoph: The Smart Data Server: A New Kind of Middle-Tier. In: FURTH, B. (Hrsg.): *Proceedings of the IASTED International Conference - Internet, Multimedia Systems and Applications, IMSA'99, October 18-21, 1999, Nassau, The Bahamas* IASTED (Veranst.), IASTED/ACTA Press, Oktober 1999, S. 361–365. – ISBN 0-88986-269-9

[Roth u. a. 2001a] ROTH, Uwe ; HAFFNER, Ernst-Georg ; HEUER, Andreas ; ENGEL, Thomas ; MEINEL, Christoph: *Konzeption und Realisierung eines intelligenten Informationsmanagers auf Basis von Internet-Technologie*. Abschlussbericht für die Stiftung Rheinland-Pfalz für Innovationen. Juni 2001

[Roth u. a. 2002] ROTH, Uwe ; HAFFNER, Ernst-Georg ; MEINEL, Christoph: The Internal Workflow of the Smart-Data-Server. In: ISAÍAS, Pedro (Hrsg.): *Proceedings of the IADIS International Conference WWW/Internet, 13-15 November, 2002, Lisbon, Portugal*, IADIS Press, November 2002, S. 572–576. – ISBN 972-9027-53-6

[Roth u. a. 2001b] ROTH, Uwe ; LOUIZI, Kais ; HAFFNER, Ernst G. ; MEINEL, Christoph: How Much Middle Tier Do You Need? In: FOWLER, Wendy (Hrsg.) ; HASEBROOK, Joachim (Hrsg.): *Proceedings of WebNet 2001 - World Conference on the WWW and Internet, October 23-27, 2001, Orlando, Florida, USA*, AACE, Oktober 2001, S. 1052–1056. – ISBN 1-880094-46-0

[Roth und Meinel 2002] ROTH, Uwe ; MEINEL, Christoph: Fallbeispiel: Einsatz des SDS in der öffentlichen Verwaltung / Institut für Telematik e.V., Trier. Februar 2002 (Preprint 01-08). – Preprint. ISSN 1433-0106

[Rowe u. a. 2002a] ROWE, Lawrence A. ; OOI, Wei T. ; PLETCHER, Peter: INDIVA: Distributed Streaming Media and Equipment Control Middleware / Berkeley Multimedia Research Center, University of California. Berkley, April 2002. – TR 2002-163

[Rowe u. a. 2002b] ROWE, Lawrence A. ; OOI, Wei T. ; PLETCHER, Peter: Indiva: Middleware for Managing Distributed Media Environment / Berkeley Multimedia Research Center, University of California. Berkley, September 2002. – TR 2002-166

[Sadiq und Kumar 2002] SADIQ, Waqar ; KUMAR, Sandeep: Web Service Description Usage Scenarios / World Wide Web Consortium (W3C). URL `http://www.w3.org/TR/ws-desc-usecases/`. – Zugriffsdatum: 30.04.2003, Juni 2002 (WD-ws-desc-usecases-20020604). – W3C Working Draft

[Scribner und Stiver 2001] SCRIBNER, Kennard ; STIVER, Mark C.: *SOAP Developer's Guide*. Markt+Technik Verlag, 2001. – ISBN 3-8272-5944-4

[Sessions 1997] SESSIONS, Roger: *COM and DCOM*. Wiley & Sons, 1997. – ISBN 0-471-19381-X

[Stal 2001] STAL, Michael: C# und .NET-Tutorial: Teil1. In: *IX* 2001 (2001), Dezember, Nr. 12, S. 122. – ISSN 0935-9680

[Steflik und Sridharan 2000] STEFLIK, Dick ; SRIDHARAN, Prashant: *Advanced Java Networking*. 2. Prentice Hall, April 2000. – ISBN 0-13-084466-7

[Stiller u. a. 1997] STILLER, Burkhard ; BAUER, Daniel ; CARONNI, Germano ; CLASS, Christina ; CONRAD, Christian ; PLATTNER, Bernhard ; VOGT, Martin ; WALDVOGEL, Marcel: Project Da CaPo++, Volume I: Architectural and Detailed Design / Computer Engineering and Networks Lab (TIK), Swiss Federal Institute of Technology (ETH) Zurich. Cambridge, Juli 1997. – TIK Report Nr. 28

[Sun Microsystems 1987] SUN MICROSYSTEMS: XDR: External Data Representation standard / Internet Engineering Task Force. Juni 1987 (1014). – RFC. Online Referenz: `http://www.ietf.org/rfc/rfc1014.txt`

[Sun Microsystems 1988] SUN MICROSYSTEMS: RPC: Remote Procedure Call Protocol specification / Internet Engineering Task Force. April 1988 (1050). – RFC. Online Referenz: `http://www.ietf.org/rfc/rfc1050.txt`

[Sun Microsystems 1997] SUN MICROSYSTEMS: *Java Core Reflection - API and Specification*, Januar 1997. – URL `ftp://ftp.java.sun.com/docs/jdk1.1/java-reflection.pdf`. – Zugriffsdatum: 19.03.2003

[Sun Microsystems 2001a] SUN MICROSYSTEMS: *Enterprise JavaBeans Specification, Version 2.0*, August 2001. – URL `ftp://ftp.java.sun.com/pub/ejb/947q9tbb/ejb-2_0-fr2-spec.pdf`. – Zugriffsdatum: 19.03.2003

[Sun Microsystems 2001b] SUN MICROSYSTEMS: *Java Object Serilization Specificaton Version 1.4.4*, 2001. – URL `ftp://ftp.java.sun.com/docs/j2se1.4/serial-spec.pdf`. – Zugriffsdatum: 19.03.2003

[Sun Microsystems 2002a] SUN MICROSYSTEMS: *Java Remote Method Invocation (Revision 1.8)*, 2002. – URL `ftp://ftp.java.sun.com/docs/j2se1.4/rmi-spec-1.4.pdf`. – Zugriffsdatum: 19.03.2003

[Sun Microsystems 2002b] SUN MICROSYSTEMS: *Java2 Platform, Enterprise Edition Specification v1.4*, November 2002. – URL `http://java.sun.com/j2ee/j2ee-1_4-pfd2-spec.pdf`. – Zugriffsdatum: 19.03.2003

[Thompson und Mendelsohn 2001] THOMPSON, Henry S. ; MENDELSOHN, David Beech Murray Maloney N.: XML Schema Part 1: Structures / World Wide Web Consortium (W3C). URL `http://www.w3.org/TR/xmlschema-1/`. – Zugriffsdatum: 30.04.2003, März 2001 (PR-xmlschema-1-20010330). – W3C Recommendation

[UDDI-Tech 2000] UDDI: *UDDI Technical White Paper*, September 2000. – URL `http://www.uddi.org/pubs/Iru_UDDI_Technical_White_Paper.pdf`. – Zugriffsdatum: 19.03.2003

[Vogt u. a. 1993] VOGT, M. ; PLAGEMANN, T. ; PLATTNER, B. ; WALTER, T.: A Runtime Environment for Da CaPo. In: *Proceedings of INET'93, Int. Networking Conf. Internet Society*. San Francisco, USA, August 1993. – URL `http://citeseer.nj.nec.com/vogt93runtime.html`

[Winer 1999] WINER, Dave: *XML-RPC Specification*, Juni 1999. – URL `http://www.xmlrpc.com/spec`. – Zugriffsdatum: 19.03.2003

[Xerox Corporation 1981] XEROX CORPORATION: *Courier: The Remote Procedure Call Protocoll*. XSIS 038112, Dezember 1981

Index

Tabellarischer Lebenslauf

Name:	Uwe Michael Roth
Geburtsdatum:	30.05.1968
Geburtsort:	Irsch, jetzt Trier
08/1974 - 07/1976	Grundschule Budel, Niederlande
08/1976 - 07/1978	Grundschule Trier-Irsch
08/1978 - 06/1987	Max-Planck-Gymnasium Trier Abschluss: Abitur
07/1987 - 09/1988	Grundwehrdienst
10/1988 - 06/1995	Studium an der Universität Kaiserslautern
09/1990	Vordiplom Informatik
12/1994	Diplom Informatik Thema der Diplomarbeit: Strategie für einen autonomen, mobilen Roboter zur Erkundung von Gebäudeumgebungen mittels 2 1/2-dimensionaler Hindernisinformation
06/1995 - 12/1997	Mitarbeiter von OCR-Datensysteme, Mannheim
01/1998 - 03/2002	Wissenschaftlicher Mitarbeiter am Institut für Telematik e.V., Trier
04/2002 - 12/2003	Wissenschaftlicher Mitarbeiter der Universität Trier
seit 04/2004	Assistant-Chercheur der Universität Luxemburg