Ein Beitrag zur Evaluierung von Komponententechnologien im Umfeld von Software zur medizinischen Bildbearbeitung.

Diplomarbeit zur Erlangung des akademischen Grades Diplominformatiker,
vorgelegt der Fakultät für Informatik und Automatisierung der
Technischen Universität Ilmenau

von: Jens Kleinschmidt
Betreuer: Dipl.-Ing. Christian Heller
verantwortlicher Hochschullehrer: Prof. Dr.-Ing. habil. Ilka Philippow
Inventarisierungsnummer: 2003-04-09/035/IN92/2232
eingereicht am: 09. Juli 2003

Danksagung

An dieser Stelle möchte ich mich recht herzlich bei meinem Betreuer Dipl.-Ing. Christian Heller bedanken. Nicht zuletzt das für mich sehr spannende Thema hat dazu beigetragen diese Arbeit zu schreiben. Vielen Dank für deine Unterstützung.

Mein Dank geht auch an die Firma Intershop Software Entwicklungs- GmbH insbesondere an den Director Consulting Services Dr. Evgeni Ivanov sowie an den Senior System Engineer Dipl.-Inf. Steffen Schneider für die Hilfe und das Verständnis sowie die Bereitstellung meiner Testumgebung.

Den größten Dank schulde ich jedoch meiner Familie. Vielen Dank für Eure Unterstützung jeglicher Art.

Inhaltsverzeichnis

1 Einführung

Die Fortschritte in der Medizin werden mit zunehmender Technik immer größer. Allein die bildgebenden Verfahren in der Medizin werden ständig verbessert bzw. es werden neue Verfahren entwickelt. Dank dieser Verfahren werden die Diagnosen und somit die Heilungschancen verbessert. Doch mit diesen Verfahren fallen eine Menge von Daten an, die man weiter verarbeiten muss.

Für Bilddaten in der Medizin gibt es ein Austauschformat, welches sich als Standard durchgesetzt hat das Digital Imaging and COmmunications in Medicine (DICOM). Dieser Standard beschreibt den Aufbau eines Bildes inklusive einiger Metadaten sowie die Kommunikation mit den Geräten, die diese Bilder erzeugen. Der Transport der Daten erfolgt mittels des proprietärem Transmission Control Protocol/Internet Protocol (TCP/IP).
Bei einem Austausch von Daten über Rechnergrenzen hinweg kann das Protokoll auch benutzt werden, was aber nicht zu empfehlen ist, da dieses Protokoll keine Möglichkeiten der Verschlüsselung der Daten oder Authentifizierung bietet. So sollten andere Wege der Kommunikation in Betracht gezogen werden.

Eine Untersuchung von Komponententechnologien wie CORBA und SOAP soll zeigen, welche der Technologien sich am besten eignet, um Bilder bzw. im weiteren Sinne Binärdaten zwischen Applikationen auszutauschen. Dazu ist es notwendig zu untersuchen, wie Informationen über Plattform-, Netzwerk- und System-Grenzen hinweg ausgetauscht werden können.

1.1 Motivation

Die eingesetzte Software in Kliniken und Praxen genügt bei Weitem nicht mehr den Anforderungen, die an eine effiziente, wenig kostenintensive Software gestellt wird.
Durch die Änderungen im Gesundheitswesen ändern sich ständig technische Prozesse, die laufend Updates der Software nötig machen. Das Ganze macht Software im Bereich der Medizin sehr kostenintensiv und aufwendig. Trotz des anscheinend lukrativen Marktes gibt es wenig gute Software, die alle Gebiete der Verwaltung und Behandlung von Patienten umfasst.

Deshalb ist es das vorrangige Ziel von Res Medicinae diese Bereiche abzudecken, um von Praxen, Kliniken, Laboren, Hardwareherstellern und pharmazeutischen Unternehmen eingesetzt werden zu können.
Da das Projekt Res Medicinae unter der GNU General Public Licence (GPL) steht, wird diese Software auch denen zur Verfügung stehen, deren Software Budget nicht ausreicht, um sich die teure Software leisten zu können [1].

1.2 Problemstellung

Ein Teilgebiet medizinischer Software ist die Verwaltung von Bildern aller Art, die im Laufe der Zeit bei Patienten anfallen. Zu diesen bildgebenden Verfahren in der Medizin gehören Röntgen, Computertomographie (CT), Ultraschall, Magnetresonanztomographie (MRT) und viele andere.
Die Situation dabei in deutschen Kliniken und Arztpraxen ist leider so, dass bei einem Wechsel des Arztes zwar die Daten der Krankenversicherung auf einer Chipkarte gespeichert werden, aber die Vita des Patienten muss vom nachfolgenden Arzt angefordert werden, damit unnötige Untersuchungen vermieden werden können, was nicht nur Kosten einspart, sondern unter Umständen, wie bei beim Röntgen, auch die Strahlenbelastung des Patienten reduziert.
Leider gibt es für diesen Austausch der Krankenakte keine Standards in diesem Bereich. Ganz abgesehen von den rechtlichen Gegebenheiten, die ein solcher Austausch von geschützten Daten unterliegt. Dennoch wäre ein solches System zum Beispiel innerhalb einer Klinik einsetzbar.
Diese Arbeit soll nun untersuchen welche Technologien beim Transport von Bildern bzw. binären Daten zum Einsatz kommen könnten. Dabei soll besonders auf zwei Technologien zurückgegriffen werden: Common Object Request Broker Architecture (CORBA) und Simple Object Access Protocol (SOAP).
Ein Vergleich dieser Technologien und deren Alternativen soll zeigen, welche Technologie sich für diesen Anwendungsfall am besten eignet.
Dabei sollen Kriterien wie Implementierungsaufwand, Einarbeitungsaufwand, Wartbarkeit, Portierbarkeit, Handhabung usw. untersucht werden. Eine prototypische Umsetzung soll dabei den Einsatz in der Praxis beweisen.

1.3 Zielsetzung

Das Ziel dieser Diplomarbeit ist es dem Entwickler eine Unterstützung zur Entscheidung über den Einsatz einer Komponententechnologie wie CORBA oder SOAP zu geben.
Dabei soll insbesondere Augenmerk auf das Umfeld der medizinischen Bildverarbeitung Rücksicht genommen werden. Aber auch für andere Bereiche außerhalb der Bildverarbeitung soll diese Arbeit Entscheidungshilfe sein. Zu diesem Zweck werden die Komponententechnologien CORBA und SOAP ausführlich beleuchtet.
Um bestimmte Eigenschaften wie Verarbeitungsgeschwindigkeit, Handhabung, Integration in die Systemumgebung sowie Implementierungsaufwand usw. vergleichen zu können sollen minimale Testsysteme erstellt werden.
Diese Testsysteme werden nach erfolgreich durchgeführten Tests durch Prototypen abgelöst, die die Implementationen von CORBA und SOAP in Umfeld der medizinischen Bildverarbeitung zeigen. Dazu wird das Werkzeug ImageJ [2] so erweitert, dass man Bilder via CORBA bzw. SOAP von einem entfernten Rechner übertragen kann, um es dann in ImageJ weiter bearbeiten zu können.
Die Untersuchungen kommen dann freier Software, wie Res Medicinae, zu Gute.
Res Medicinae (lateinisch: „Sache der Medizin“) [3] ist ein Projekt zur Entwicklung eines freien medizinischen Informationssystems. Das Projekt [4] ist Open Source und untersteht der GNU General Public Licence (GPL).
Diese Lizenz beinhaltet das freie und kostenlose Nutzen der Software sowie die Modifizierung der Software für eigene Zwecke. Eine Einschränkung der Nutzung gibt es nur, falls die Software kommerziell vertrieben werden sollte. Eine kommerzielle Nutzung der Software ist jedoch frei. D.h. der Einsatz in einer Arztpraxis oder eine Klinik ist kostenlos. Auch können Anpassungen an die eigenen Geschäftprozesse im Quellkode gemacht werden.
Das Ziel der Entwicklung von Res Medicinae ist es eine Anwendung für das Gesundheitswesen zu schaffen, welches nicht kommerziell, frei, stabil, offen, sicher und plattformunabhängig ist.
Das Projekt soll eine Verwaltung medizinischer Informationen in Praxen für Mediziner sowie für Kliniken sein. Durch den Einsatz kostenloser Software lassen sich so enorme Kosten einsparen. Um weiter Kosten zu sparen, wie Doppeluntersuchungen oder auch nur das Porto für den Befund des Facharztes soll dem Austausch von Daten große Beachtung zukommen. Zu diesem Zweck soll diese Arbeit ihren Beitrag leisten.

2 CORBA

Dieses Kapitel gibt einen Einblick in CORBA. CORBA steht für Common Object Request Broker Architecture und wurde von einem Konsortium, der Object Management Group (OMG) entwickelt. Die OMG ist ein Konsortium aus namhaften Firmen, wie Microsoft, IBM, Hewlett-Packard, Sun Microsystems, Canon, Philips und vielen Anderen.

CORBA bietet eine Plattform für verteilte Anwendungen und komponentenbasierte Softwareentwicklung. Über ORB's (Object Request Broker) sind Applikationen in der Lage, Objekte auf beliebigen Rechnern und Plattformen zu benutzen, ohne das dazwischenliegende Netzwerk berücksichtigen zu müssen.

Was macht den Einsatz von CORBA oder anderer verteilter Technologien notwendig?
Zur Zeit monolithischer Systeme, den Großrechnern mit Terminals, gab es natürlich noch keine Notwendigkeit zum Einsatz verteilter Software. Aber im Laufe der Zeit, als verteilte System sich mehr und mehr durchsetzten, wurde nach Lösungen gesucht auch verteilt zu rechnen, um die Power von mehreren Rechnern für eine Aufgabe zu nutzen [5].

Welche Vorteile erhofft man sich durch den Einsatz von CORBA?

Erhoffte Vorteile:

- Erhöhte Stabilität und Ausfallsicherheit durch Strukturierung des Gesamtsystems und Trennung von Systemteilen
- Transparenz bezüglich des Ortes eines Objektes - der Client weis nicht, wo sich die Objekt Implementation befindet.
- Transparenz bezüglich der Objektimplementierung - der Client kennt weder die Programmiersprache, in der das Objekt implementiert ist, noch das Betriebssystem und die Hardware, auf der sie läuft.
- Größere Flexibilität durch Verteilung von Objekten
- Wiederverwendbarkeit
- Bessere Wartbarkeit
- Herstellerunabhängigkeit

Da kein System vollkommen ist, sollen die zu erwartenden Nachteile beim Einsatz von CORBA auch Erwähnung finden.

Zu erwartende Nachteile:

- Mehraufwand an Kommunikation zwischen den einzelnen Objekten auch über Plattformen hinweg
- Mehraufwand zur Gewährleistung von Sicherheit, Portabilität, Erweiterbarkeit, Komplexität des Systems steigt

Als Konsequenz der Nachteile lässt sich ziehen, dass sich durch Einführung eines Mehrschichtensystem (Multi-Tier) die Aufwände für Sicherheit, Erweiterbarkeit, Portabilität usw. minimieren lassen.

2.1 CORBA als Komponententechnologie

Als Komponenten versteht man Einheiten von Software, die kontextunabhängig sind sowohl in konzeptioneller als auch technischer Hinsicht [6].
Das bedeutet, dass Komponenten von der Problemstellung so weit abstrahieren, dass sie leicht zur Lösung ähnlicher Probleme verwendet werden können. Dabei sind sie auch vom technischen Umfeld wie Programmiersprache oder Plattform unabhängig.
Aus diesen Eigenschaften leitet sich die Wiederverwendung von Lösungen mittels Komponententechnologien ab.

Zu diesen Komponententechnologien gehören die Middleware CORBA und SOAP. Unter Middleware versteht man Softwaretechnologien, dessen Einsatz dem Entwickler maßgebliche Arbeit bei der Entwicklung von Software abnehmen soll. Zu ihnen gehören unter anderem CORBA und SOAP.

Middleware ist also eine Software für Entwickler zur Entwicklung verteilter Anwendungen. Sie übernimmt größtenteils die Aufgaben zur Verteilung der Komponenten, die Kommunikation zwischen Client und Server.

Welche Anforderungen werden an eine offene Architektur wie CORBA gestellt?
Diese Anforderungen gelten natürlich im gleichen Maße für CORBA und für SOAP. Zu diesen zählen Erweiterbarkeit, Verteilung, Skalierbarkeit, Stabilität, Sicherheit und Portabilität.

2.2 Alternativen zu CORBA

CORBA ist sicher nicht die einzige Möglichkeit verteilte Anwendungen zu entwickeln. Aber mit Sicherheit ist sie eine der effektivsten und am leichtesten zu implementierende Möglichkeit. Alternativen zu CORBA findet man jeweils in Abhängigkeit der eingesetzten Plattform sowie der Programmierumgebung, der Art der zu erstellenden Anwendung sowie einer Vielzahl anderer Parameter.

2.2.1 Socket

Bei Kommunikation unter Rechnern oder auch Prozessen auf einem einzelnen Rechner wird sehr häufig auf die Socket-Programmierung zurückgegriffen. Das bekannteste Beispiel ist wohl die Implementierung des Netzwerkprotokoll TCP/IP unter Unix (BSD-Sockets) und seit 1992 auf der Windows-Plattform (WinSock).
Die Anwendungsprogrammierschnittstelle - Application Program Interface (API) von Sockets ist hardwarenah, was zur Folge hat, das bei komplexeren Projekten ein nicht unerheblicher Teil des Programmieraufwandes darauf verwand werden muss sich um die Kommunikation zu kümmern.
Deshalb wird meistens nicht auf die Socket-Programmierung zurückgegriffen, wenn man größere Projekte bearbeitet, obwohl man mit ihr sehr wohl sehr effiziente und kompakte Anwendungen entwickeln kann.

2.2.2 Remote Procedure Calls (RPC)

Einen höhergestellten Dienst als Socket-Programmierung bieten Remote Procedure Calls. Sie bieten eine funktionsorientierte Schnittstelle zur Kommunikation auf Socketebene. Es existiert eine ganze Anzahl von nicht miteinander kompatiblen Implementierungen. Trotzdem gibt es einen Standard, der auf allen Systemen eingesetzt werden kann.

2.2.3 Distributed Computing Environment (DCE)

Unter DCE versteht man eine Reihe von Standards rund um das verteilte Rechnen. Den Grundstein hierfür legte die Open Software Foundation (OSF). Obwohl dieser Standard schon relativ alt ist, fand er jedoch keine größere Verbreitung.

2.2.4 Distributed Component Object Model (DCOM)

DCOM ist ein von Microsoft entwickeltes Objektmodell für verteilte Systeme. Und besonders gut in die Betriebssystemreihe „Windows“ integriert. DCOM hat aber defacto keine Bedeutung für andere nicht Windows-Systeme. Deshalb kommt DCOM nicht für verteilte Systeme in Frage, die kein Windows Betriebsystemen besitzen.

2.2.5 Remote Method Invocation (RMI)

Remote Method Invocation (RMI) ist CORBA sehr ähnlich und unterstützt auch verschiedene Plattformen und Systeme. RMI wurde das erste Mal mit dem Java™ Development Kit (JDK) 1.1 veröffentlicht und wurde als die natürliche Erweiterung der Remote Procedure Calls (RPC) gehandelt. Das Ziel der Entwicklung von RMI war es Aufrufe zwischen Java Objekten verschiedener virtueller Maschinen über verschiedene physische Maschinen hinweg zu gewährleisten. Daraus folgt, dass RMI nur auf Systeme mit einer Java-Umgebung beschränkt ist.

CORBA unterstützt RPC in gleicher Weise wie RMI, welches es lokalen Objekten erlaubt Methoden entfernter Objekte aufzurufen. Der große Unterschied zu CORBA ist aber, dass CORBA RPC zwischen Objekten, implementiert in irgendeiner Sprache, möglich macht.
Natürlich ist es auch möglich für RMI entfernte Services aufzurufen, die nicht in Java geschrieben wurden. Dazu ist es nur nötig einen Wrapper zu schreiben, der den nicht Java Kode des Servers einhüllt. Das Wrapper Objekt verbindet sich extern zum Java-Klienten über RMI und intern, zum nicht Java Kode, zum Beispiel über das Java Native Interface (JNI).
Doch das ist genau das, was CORBA schon ermöglicht.

	Common Object Request Broker Architecture CORBA	Distributed Component Object Model DCOM	Remote Method Invocation RMI
Standard	Offen (Konsortium mit 800 Mitgliedern)	Microsoft (Open Software)	Sun Microsystems (Java)
Plattformen	Windows, Unix (ca. 30)	Windows, Unix	Jede mit Java Virtual Machine (JVM)
Sprachen	C, C++, Cobol, Java, Smalltalk, Objective-C	C, C++, Cobol, Fortran, Java, Perl, VB	Java, Smalltalk
Netzwerke	TCP-IIOP, proprietär	RPC, IPX, HTTP	JavaSocket/TCP, IIOP

Abbildung 1: Vergleich CORBA/DCOM/RMI

2.3 Geschichte

2.3.1 Object Management Group (OMG)

Die OMG wurde im April 1989 mit insgesamt elf Mitgliedern gegründet. Zu diesen zählt unter Anderen Sun Microsystems, 3Com, Canon und Hewlett-Packard.
Heute hat die OMG mehr als 800 Mitglieder mit Sitz in Needham (USA) und Marketingbüros in Deutschland, England, Italien, Indien und Brasilien.

2.3.2 Die erste Version CORBA 1.0

Im Dezember 1990 wurde die Version CORBA 1.0 eingeführt. Anfang des folgenden Jahres dann die Version 1.1. Diese Version beinhaltete die Interface Definition Language (IDL) sowie die API für Anwendungen zur Kommunikation mit dem Object Request Broker (ORB).

2.3.3 Die aktuelle Version CORBA 2.x

CORBA 1.x stellte einen wichtigen Schritt hin zur Objektkompatibilität dar. Ein Manko dieser Version war jedoch, dass es keine Spezifikation zur Kommunikation unter den einzelnen ORB's gab.
Was zur Folge hatte, dass Produkte unterschiedlicher Hersteller nicht miteinander über die ORB's kommunizieren konnten. Im Dezember 1994 wurde dann als erster Schritt für die Version 2.0 ein Standardprotokoll definiert. Dieses Protokoll heißt Internet Inter-ORB Protocol (IIOP). Wie der Name schon ausdrückt, handelt es sich bei diesem Protokoll um ein Protokoll auf der Grundlage des Internet Protokolls TCP/IP.

2.3.4 Ein Ausblick auf CORBA 3.0

CORBA 3 wird eine bessere Integration in Java und an das Internet enthalten. So wird die Firewall für CORBA kein größeres Hindernis mehr darstellen. Ein neuer Service, der Quality of Service Control (QoS) wird für effizientere Dienste sorgen. So wird es möglich sein, CORBA in einer Real-Time Umgebung einzusetzen. Ein dritter großer Punkt in CORBA 3 wird die Integration von Scriptsprachen sein. So wird es ein Mapping zu Python und eine eigene CORBA spezifische Sprache geben [7].

CORBA 1.x	CORBA 2.x	CORBA 3.0
Basic ORB	IIOP	IIOP Firewall
C Binding	C++ Bindings	Java Bindings
Naming	Security	Mobile Agents
Persistence Life Cycle	Transactions	Automatic Persistence
	Time Licensing	CORBA/DCOM Bridge
		Messaging
		Multiple Interfaces

Abbildung 2: Vergleich der CORBA-Versionen

2.4 Die Object Management Architecture

Die Object Management Group (OMG), Inc. hat es sich zur Aufgabe gemacht, ein architekturbezogenes Gerüst für objektorientierte Anwendung und deren Schnittstellenspezifikation zu erstellen. Diese wurden in der Object Management Architecture (OMA) zusammengefasst. Ein Teil dieser OMA ist CORBA.

Die OMA besteht aus folgenden Komponenten:

Das Application Interface stellt Anwendungsobjekte für den Entwickler zur Verfügung. Durch Ableitung von neuen Klassen oder von existierenden Klassen durch Generalisierung oder Spezialisierung können neue Objekte erstellt werden.

Das Domain Interface beinhaltet schnittstellenspezifische Anwendungen aus folgenden Domänen [8] (Auswahl):

- Finanzwelt – Finance DTF (früher CORBAfinancials) [9]
- Telekommunikation – Telecommunications DTF (früher CORBAtel) [10]
- Gesundheitswesen – Healthcare DTF (früher CORBAmed) [11]
- Transportwesen – Transportation DTF (CORBAtransport) [12]
- Business System Integration (BSI) – Business Enterprise Integration DTF [13]

Die Spezifikationen wurden von den jeweiligen Domain Task Force's (DTF's) festegelegt.

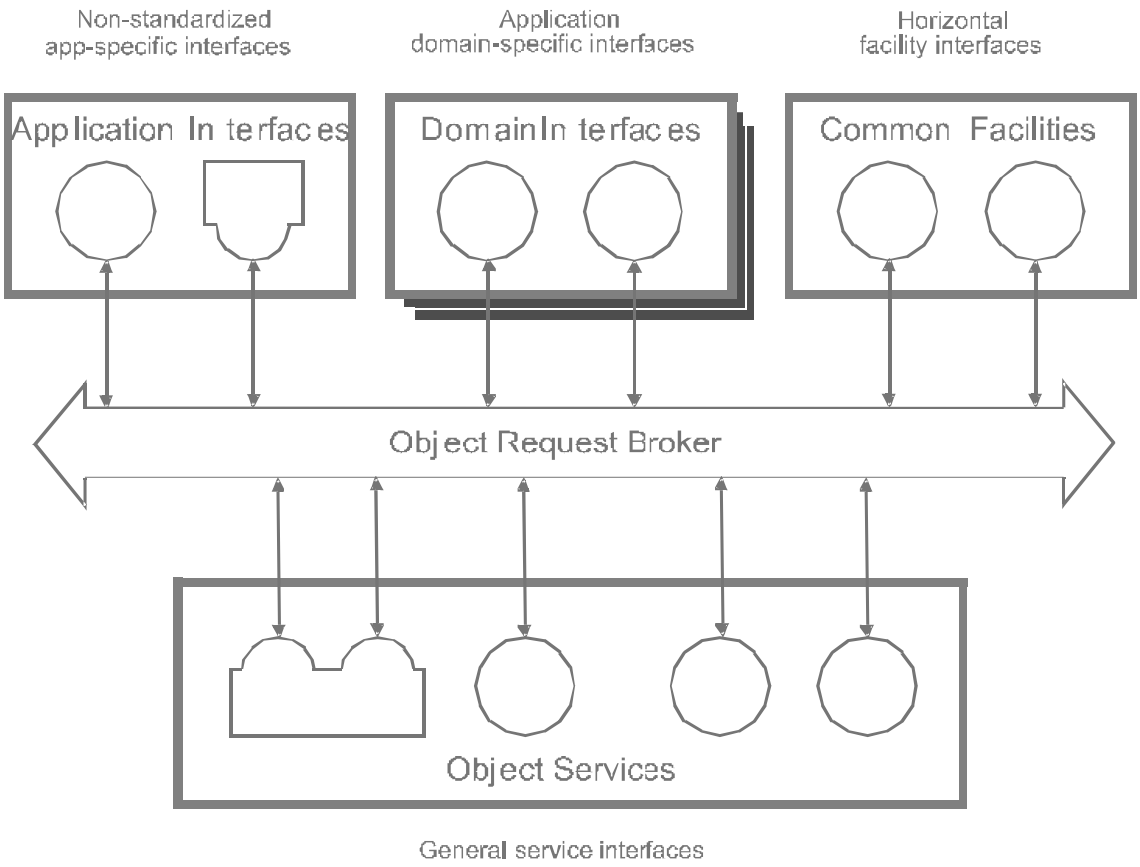

Abbildung 3: "A Discussion of the Object Management Architecture", OMG, Inc.

Die Common Facilities stellen horizontale (anwendungsübergreifende) und vertikale (anwendungsorientierte) Dienste zu Verfügung:

Horizontal Facilities:

- Dokumentenverwaltung
- Informationsverwaltung,
- Taskverwaltung
- Systemverwaltung
- Mobile Agenten

Vertical Facilities:

- Telekommunikation
- Gesundheitswesen
- Finanz- und Geschäftsanwendungen
- Öl- und Gasindustrie

Die Object Services haben die Aufgabe den Lebenszykluss von Objekten zu managen und stellt Funktionen zum Erstellen von Objekten sowie regeln sie die Kontrolle des Zugriffs auf diese. Die Spezifikation der Object Services ist in den CORBAservices (Common Object Services Specifications) festgehalten und unterscheidet folgende fünf Services:

Der Lifecycle Service dient zur Objekterzeugung und -freigabe, dem Kopieren, Bewegen, Löschen von Objekten.

Der Persistence Service dient als einfache Schnittstelle zur dauerhaften Speicherung von Objekten in OODBMS, RDBMS sowie in einfache Dateien.

Der Naming Service ist ein Verzeichnis von Objekten bzw. eine Abbildung von (hierarchischen) Namen auf Objekte und dient zur Integration anderer Dienste (z.B. DCE, X.500, Sun NIS).

Der Event Service dient der Meldung von Ereignissen über eine asynchrone Kommunikation. Welches einem einfache Message Queueing entspricht.

Der Concurrency Control Service enthält Thread-Mechanismen zur Vermeidung gleichzeitiger Zugriffe.

Der Object Request Broker trennt die Implementation von der Schnittstelle. Er bietet die Infrastruktur zur Kommunikation der Objekte untereinander und gewährleistet darüber hinaus die Portabilität und Interoperabilität der Objekte über Netzwerke und heterogene Systeme.
Die Spezifikation des ORB's wurde in CORBA, der Common Object Request Broker Architecture and Specification, festgehalten.
Diese Spezifikation unterstützt laut CORBA sowohl statische als auch dynamische Methoden-Aufrufe. Vielen Middleware Produkte haben jedoch nur das statische Binden implementiert. Der ORB ist ein selbstbeschreibendes System mit Anbindung an alle Hochsprachen, wie C++, Pascal sowie Java.
Eine weitere Eigenschaft von CORBA ist die Unterstützung von Metadaten zur Laufzeit, die jeden Server beschreiben. CORBA ist lokal und entfernt transparent, d.h. ORB´s können auf einem Rechner oder im Netzwerk, durch Internet Inter-ORB Protocol (IIOP) verbunden, laufen.

CORBA beinhaltet deshalb auch eine eingebaute Sicherheit und Transaktionen über Plattformen hinweg. Des Weiteren unterstützt CORBA polymorphe Botschaften, was dazu führt, dass gleiche Funktionsaufrufe zu verschiedenen Effekten in Abhängigkeit des Objektes, welches sie erhält, führt.

2.4.1 Die Common Object Request Broker Architecture

Die folgende Abbildung 4 zeigt die Common Object Request Broker Architecture, deren Bestandteile nachfolgend näher erläutert werden.

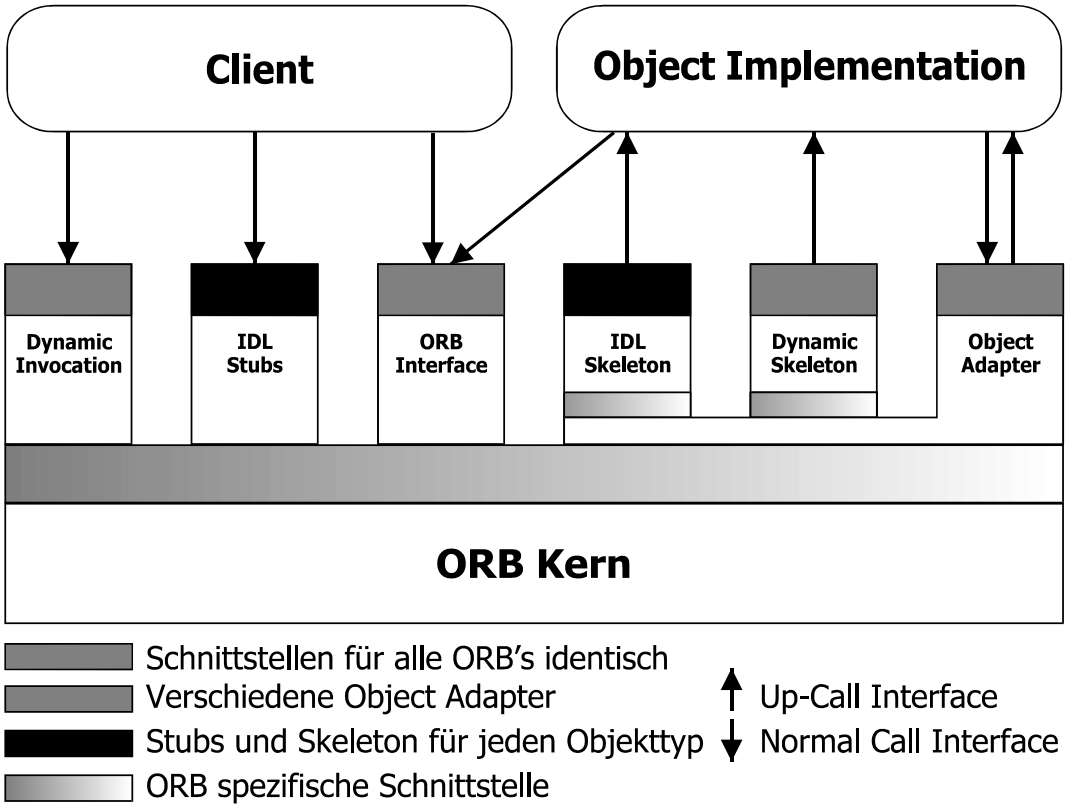

Abbildung 4: Common Object Request Broker Architecture (CORBA)

2.4.2 Object Request Broker (ORB)

Der ORB ist die Kommunikations- und Aktivierungsinfrastruktur für Anwendungen verteilter Objekte. Er liefert Anfragen (Requests) an möglicherweise entfernte Objekte und gibt die Antwort (Response) an den Client zurück.

Es besteht eine Transparenz der Client/Objekt Kommunikation bezüglich:
Des Ortes eines Objektes: Der Client weiß nicht, wo sich das Zielobjekt (Objektimplementierung) befindet.

Der Objektimplementierung: Der Client kennt weder die Programmiersprache, in der das Objekt implementiert ist, noch das Betriebssystem und die Hardware, auf der es läuft.
Des Ausführungszustandes eines Objektes: Der ORB startet (aktiviert), falls nötig, die Objekte, ehe er Requests an sie übergibt. Daher braucht der Client bei seinen Anfragen darauf keine Rücksicht zu nehmen.
Der Kommunikationsmechanismen: Der Client kennt die vom ORB zur Kommunikation benutzten Mechanismen (z.B.: TCP/IP, Lokale Methodenaufrufe) nicht. Deshalb sind Requests aus Sicht des Clients wie lokale Prozeduraufrufe.

2.4.3 Client

Der Client ist der Dienstnutzer der Object Implementation (Server). Er löst die Requests aus.

2.4.4 Object Implementation

Die Object Implementation ist eine Entität einer Programmiersprache, dass ein oder mehrere CORBA-Objekte implementiert. Sie ist der Diensterbringer für den Client.
Object Implementations (Servants) verkörpern CORBA-Objekte oder andersherum CORBA-Objekte sind Instanzen von Servants.

2.4.5 Stubs

IDL-Compiler erzeugen aus IDL-Definitionen auf der Clientseite Stubs (englisch: „Stummel“). Alle Informationen müssen zur Komplierungszeit vorhanden sein. Stubs sind statisch und dienen als Client-Proxy zur Weiterleitung von Requests an den ORB.
Sie übertragen Eingabewerte (Marshaling) an den ORB in einem schnittstellenspezifischen Format.

2.4.6 Skeletons

IDL-Compiler erzeugen aus IDL-Definitionen auf der Serverseite Skeletons (englisch: „Skelette“). Alle Informationen müssen zur Komplierungszeit vorhanden sein. Skeletons sind statisch und dienen als Server Proxy zur Weiterleitung von Requests an die Objektimplementierung.
Sie übertragen Eingabewerte (Unmarshaling) in der jeweiligen Programmiersprache in einem schnittstellenspezifischem Format und vis-versa mit den Ausgabewerten.

2.4.7 Interface Definition Language (IDL)

Die Interface Definition Language (IDL) ist eine programmier- und sprachenunabhängige Schnittstellenbeschreibungssprache. Sie beschreibt die an den Schnittstellen der verteilten Objekte sichtbaren Eigenschaften und legt die Methoden und Attribute der im CORBA-System verteilten Objekte fest.
IDL ist eine rein deklarative Sprache, d.h. sie beinhaltet keine Implementierungen von Funktionalitäten. Dadurch erhält man eine strikte Trennung von Schnittstellendefinition und -implementierung eines Objektes. Dieses Vorgehen ermöglicht die Integration möglichst vieler auch nicht objektorientierter Programmiersprachen. Die Trennung von Definition und Implementierung ist auch die Voraussetzung für die Weiterverwendung von bereits bestehendem Quellkode.

Die Abbildung 5 zeigt das Mapping der CORBA IDL Schnittstellenbeschreibung auf die entsprechenden Java Datentypen.

CORBA IDL	Java
void	void
boolean	boolean
wchar	char
octet	byte
short	short
long	int
long long	long
float	float
double	double
long long	long

Abbildung 5: Mapping CORBA IDL/Java – Datentypen

CORBA IDL	Java
Module	Package
Interface	Interface
Method	Method

Abbildung 6: Mapping CORBA IDL/Java – Begriffe

2.4.8 ORB-Interface

Das ORB-Interface stellt grundlegende Operationen, wie das Konvertieren von Objektreferenzen zu Strings und umgekehrt zur Verfügung. Eine weitere Operation ist die Erstellung von Parameterlisten für Methodenaufrufe über Dynamic Invocation.

2.4.9 Dynamic Invocation/Dynamic Skeleton

Dynamic Invocation bzw. Dynamic Skeleton sind unabhängig von der IDL und benötigen keine Informationen über Objektschnittstellen zur Kompilierung.

2.4.10 Interface Repository

Das Interface Repository ist ein hierarchisch aufgebautes Archiv bestehend aus IDL-Definitionen, die zur Laufzeit abgefragt werden. Informationen werden als IDL-Definitionen repräsentiert, so dass ein Zugriff auch für entfernte Objekte möglich ist.

2.4.11 Implementation Repository

Das Implementation Repository enthält Informationen, die der Lokalisierung und Aktivierung von Objektimplementierungen durch den ORB dienen. Sowie zusätzliche Informationen zu den Objektimplementierungen wie Debugger-Informationen, administrative Kontrollinformationen und Informationen zur Ressourcenallokation.

2.4.12 Object Adapter

Der Object Adapter demultiplext Requests. D.h. er leitet eine Anfrage zum passenden Servant weiter. Des Weiteren leitet er die auszuführenden Operationen nach Auffinden des Zielservants weiter. Dazu werden Objektreferenzen registrierter CORBA-Objekte zur Identifizieren und Adressierung benötigt, die der Object Adapter erzeugt.
Folgende Arten von spezialisierten Object Adaptern gibt es in CORBA: Basic Object Adapter (BOA), Portable Object Adapter (POA), Library Object Adapter (LOA), Data Base Object Adapter (DBOA).

2.4.13 Inter-ORB Protocol (IIOP)

Das Inter-ORB Protocol (IOP) dient der Kommunikation mit anderen ORB's und ist eine spezialisierte Version des General Inter-ORB Protocol (GIOP). Die Interoperabilität wird durch Brücken realisiert, die dafür sorgen, dass eine Protokollumwandlung zwischen den Systemen, wobei jeder ORB sein eigenes spezielles Protokoll verwenden darf, statt findet.
Das Internet Inter-ORB Protocol (IIOP) wiederum ist ein spezialisierteres Protokoll des Inter-ORB Protocol's und verwendet die im Internet üblichen Protokollfamilie TCP/IP.
Für ORB-Interoperabilität gilt ein standardisiertes Referenzformat für Objekte die Interoperable Object Reference (IOR). Eine IIOP-IOR enthält z.B. den Namen des Hosts und eine TCP/IP-Portadresse. Die IOR speichert die notwendigen Informationen, um Objekte zu finden und mit ihnen auch über mehrere Protokolle hinweg kommunizieren zu können [14].

2.5 Beispiel in Java

Zwei Arten von Methoden zur Entwicklung verteilter Anwendungen mit CORBA:

Statische Methode:

1. Definieren der Server Interfaces mittels Interface Definition Language (IDL)
2. IDL mittels Language Precompiler kompilieren
3. Servant implementieren
4. Servant kompilieren
5. Instanzzieren der Objekte des Servers
6. Client implementieren, kompilieren, starten

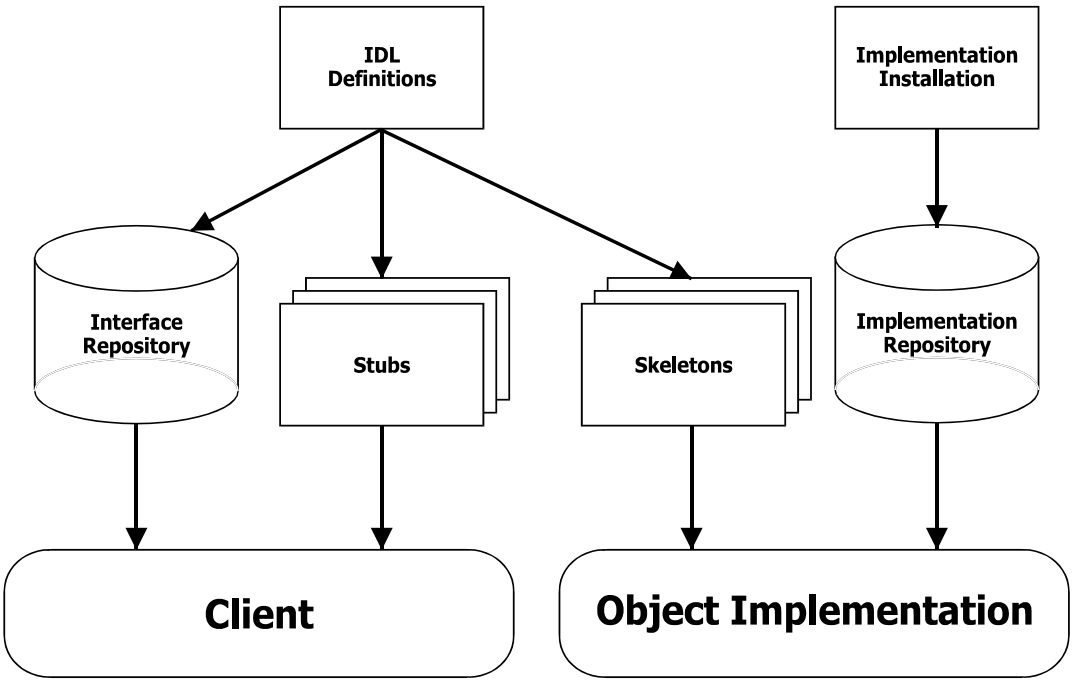

Abbildung 7: Methoden zur Entwicklung verteilter Anwendungen

Dynamische Methode:

Mittels Repository

```
module Image
{
  struct SImage
  {
    sequence <octet> imageData;
  };

  interface IImage
  {
    SImage getImage(in string fileName);
  };
};
```

Abbildung 8: Image.idl

```
…
public class ImageServer {

  public static void main(String[] args) {
    try {
      // Create and initialize the ORB.
      ORB orb = ORB.init(args, null);

      // Get reference to rootpoa & activate the POAManager.
      POA rootpoa =
        POAHelper.narrow(orb.resolve_initial_references("RootPOA"));
      rootpoa.the_POAManager().activate();

      // Create servant and register it with the ORB.
      ImageServerImpl imageServerImpl = new ImageServerImpl("Image");
      imageServerImpl.setORB(orb);

      // Get object reference from the servant.
      org.omg.CORBA.Object ref =
        rootpoa.servant_to_reference(imageServerImpl);
      IImage href = IImageHelper.narrow(ref);

      // Get the root naming context.
      // NameService invokes the name service.
      org.omg.CORBA.Object objRef =
        orb.resolve_initial_references("NameService");
      // Use NamingContextExt which is part of the Interoperable
      // Naming Service (INS) specification.
      NamingContextExt ncRef = NamingContextExtHelper.narrow(objRef);

      // Bind the Object Reference in Naming.
      String name = "Image";
      NameComponent path[] = ncRef.to_name(name);
      ncRef.rebind(path, href);

      // Wait for invocations from clients.
      orb.run();
    }
    catch (Exception exception) {
      System.err.println("ERROR: " + exception);
```

```
      exception.printStackTrace(System.out);
    }
  }
}
```

Abbildung 9: ImageServer.java

```
...
public class Client {
  static IImage serverImpl;

  public static void main(String[] args) {
    try {
      // Create and initialize the ORB.
      ORB orb = ORB.init(args, null);

      // Get the root naming context.
      org.omg.CORBA.Object objRef =
        orb.resolve_initial_references("NameService");

      // Use NamingContextExt instead of NamingContext. This is
      // part of the Interoperable naming Service.
      NamingContextExt ncRef = NamingContextExtHelper.narrow(objRef);

      // Resolve the Object Reference in Naming.
      String name = "Image";
      serverImpl = IImageHelper.narrow(ncRef.resolve_str(name));
...
      // Get the image.
      byte[] imageByteArray = serverImpl.getImage(fileName).imageData;
...
    }
    catch (Exception exception) {
      System.err.println("ERROR : " + exception);
      exception.printStackTrace();
    }
  }
}
```

Abbildung 10: Client.java

Projekte bei denen CORBA eingesetzt wird:

- Hong Kong Telecom - Interactive Multimedia Services (IMS)
- Home Shopping, Video-On-Demand, Home Banking und andere Internet Services
- Ericsson – Cellular Management Operations System (CMOS)
- Project Managing, Software zur Planung, Operation und Aufrechterhaltung eines Mobilfunknetzes
- Lufthansa - Traffic And Network Ground Operation (TANGO)
- Information Management, Informationen über alle Flüge und Flugzeuge usw.

2.6 Zusammenfassung

Zusammenfassend lässt sich über CORBA sagen, dass sich diese Komponententechnologie sehr gut zum Erstellen von objektorientierten, verteilten Anwendungen eignet. CORBA stellt eine große Anzahl spezifischer Dienste zur Unterstützung von verteilten Objekten zu Verfügung.
Durch die Sprachenunabhängigkeit von CORBA ist es möglich die unterschiedlichsten Programmiersprachen zu benutzten. So gibt es Implementierungen für C++ und Java, um nur die bedeutendsten Programmiersprachen zu nennen.
Die Interoperabilität von CORBA gewährleistet das Zusammenspiel unterschiedlichster Architekturen, Plattformen und Betriebsysteme.
So gibt es für alle gängigen Betriebsysteme z.B.: Unix, Linux, Windows, Solaris und MacOS Implementierungen, ebenso wie für Intel oder Mac-Rechner oder RISC bzw. CISC Architekturen.
Alles in Allem bietet CORBA eine bessere Stabilität, eine bessere Wartbarkeit, sowie Wiederverwendbarkeit seiner Anwendungen als herkömmliche verteilte Anwendungen (siehe Kapitel 2.2 Alternativen zu CORBA).
Negativ wirkt sich nur eine nicht vorhandene Integration von Firewalls in der Version 2.x aus, die erst in CORBA 3.0 verabschiedet werden soll.

3 SOAP

Das Simple Object Access Protocol (SOAP) wurde hauptsächlich entwickelt um einen einfachen und leicht zu implementierenden Mechanismus zu haben, der strukturierte Daten in eine dezentralisierten, verteilten Umgebung austauschen kann.
Es ist im Wesentlichen ein Modell zum Kodieren von Daten im standardisierten XML-Format für eine Vielzahl von Einsatzmöglichkeiten von Messaging oder RPC.

Natürlich treffen die gleichen Ziele für verteilte Anwendungen auch für SOAP, nicht nur für CORBA, zu: größere Flexibilität, Wiederverwendbarkeit, bessere Wartbarkeit, Plattformunabhänhigkeit, Systemunabhängigkeit, Herstellerunabhängigkeit, erhöhte Stabilität sowie Integration von Geschäftsprozessen. Insbesondere Letzteres soll mit SOAP einen Aufschwung erleben.
So soll SOAP, ein Hauptbestandteil des Architekturkonzeptes Enterprise Service BUS (ESB), die Integration von Systemen verschiedener Hersteller vorantreiben.

3.1 SOAP als Komponententechnologie

Wie CORBA ist SOAP auch eine Komponententechnologie, die es erlaubt Informationen zwischen dezentralisierten, verteilten Umgebungen auszutauschen. Durch seine Basistechnologien wie HTTP und XML ist SOAP leicht auf jeder Plattform, unabhängig von der Programmiersprache, einzusetzen. So kann SOAP auch ohne eine Middleware wie axis eingesetzt werden, wenn einem auf der Zielplattform ein HTTP-Server mit einer Servlet-Engine oder eine Scriptsprache zur Verfügung steht, die XML verarbeiten kann.

3.2 Alternativen zu SOAP

Im Grunde genommen lässt sich SOAP durch alles ersetzen, was auf der Service-Oriented Architecture (SOA) aufbaut. SOA ist ein einfaches Architekturkonzept bei dem der Benutzer eines Dienstes sich durch eine Anfrage an den Diensterbringer wendet, der dann seinerseits diese Anfrage beantwortet.

Prinzipiell fallen unter SOA also alle RPC Dienste, wie CORBA, DCOM, RMI, wie sie schon ausführlicher im Kapitel 2.2 Alternativen zu CORBA beschrieben wurden.

Trotzdem soll kurz auf andere Alternativen eingegangen werden, die auch wie SOAP webbasiert sind.

3.2.1 Active Server Pages (ASP)

Active Server Pages (ASP) ist eine Technik, um dynamische und interaktive Webseiten zu gestalten. ASP benutzt dafür serverseitige Scripte um Browserunabhängige Seiten zu gestalten.
Das war es aber auch schon was die Unabhängigkeit von ASP betrifft. Denn ASP ist von Microsoft und läuft nur auf den Microsoft Betriebssystemen Windows NT4/95/98/ME/2000/XP Pro und Home, weil es den Internet Information Server (IIS) voraussetzt. Als Scriptsprachen können VBScript und JScript verwendet werden.

3.2.2 Java Servlet

Java Servlet ist eine Technologie zur Erweiterung der Funktionalität von Web-Servern. Ein Servlet kann als Anwendung die auf dem Server ausgeführt wird betrachtet werden. Java Servlets sind server- und plattformunabhängig beschränken sich aber bei der Implementierung, wie der Name schon vermuten lässt, auf die Programmiersprache Java. Die bekannteste Implementierung von Servlets ist Jakarta Tomcat. Tomcat ist ein Projekt der Apache Group, von der auch der Apache HTTP-Server stammt [15].

3.2.3 JavaServer Pages (JSP)

JavaServer Pages (JSP) sind eine Erweiterung von Java Servlet, die es serverseitigen, 100%-igen Java Anwendungen erlauben soll die Funktionalitäten von Web-Servern ohne großen Overhead zu erweitern. Sie sind in der Technologie vergleichbar mit ASP [16].

3.2.4 eXtensible Markup Language-Remote Procedure Calls (XML-RPC)

eXtensible Markup Language-Remote Procedure Calls (XML-RPC) ist eine Spezifikation, die es Anwendungen erlauben soll über verschiedene Betriebssysteme und Plattformen hinweg Prozeduren aufzurufen. Der Transportweg für diese entfernten Aufrufe ist HTTP. Das Transportmittel ist XML. Und genau das trifft auch auf SOAP zu. Denn XML-RPC ist aus einem frühen Stadium von SOAP entstanden.
XML-RPC ist also wie SOAP Hersteller-, Plattform-, Betriebsystemunabhängig und unabhängig von seiner Implementierung – egal in welcher Programmiersprache [17].

3.2.5 Digital Imaging and COmmunications in Medicine (DICOM)

Eine Ausnahme unter den vorgestellten Alternativen, da nicht webbasiert, ist Digital Imaging and COmmunications in Medicine (DICOM). DICOM ist ein Austauschformat für medizinische Geräte, welches vom American College of Radiology (ACR) und dem National Electrical Manufacturers Association (NEMA) entwickelt wurde.
Es definiert ein Datei-Format mit Metadaten und den eigentlichen Bilddaten in unkomprimierter oder komprimierter Form sowie ein Transportprotokoll, das auf TCP aufsetzt. Wie SOAP auch baut DICOM auf der Service-Oriented Architecture (SOA) auf. Einem einfachen Prinzip, dass einen Service beschreibt als Aufruf einer entfernten Funktion eines Dienstnutzers bei einem Diensterbringer.
Das Transportprotokoll DICOM Message Service Element (DIMSE) definiert Services wie den Notification Service und Operation Services in zwei Modi synchron und asynchron.
Da der Standard nur für den medizinischen Bereich entwickelt wurde und somit auch nur dort große Verbreitung gefunden hat, gibt es nur wenig freie Implementationen. Die geringe Verbreitung macht solche Produkte ansonsten sehr kostenintensiv. Deshalb wird sich das Format auch nicht auf breiter Front als Transportprotokoll zwischen Anwendungen durchsetzen können obwohl es die Spezifikation ausdrücklich vorsieht. Vielmehr wird es medizinische Geräte via LAN mit anderen Rechnern verbinden [18], [19].

3.3 Geschichte

3.3.1 HyperText Transfer Protocol (HTTP)

Das HyperText Transfer Protocol (HTTP) war in seiner frühsten Version, dokumentiert als Version HTTP/0.9, ein sehr einfaches Protokoll. Mit diesem Protokoll war es möglich sich zu einem Server zu verbinden und über ein einfaches Kommando "GET" Seiten vom Server zu beziehen. Für diese frühe Version des HTTP's gab es noch keine Header und nur ein Kommando. Die Antwort des Servers musste unbedingt ein HTML-Dokument sein.
Alle heutigen Server sollten in der Lage sein HTTP/0.9 Anfragen zu verstehen und beantworten zu können. Da die Version 0.9 aber nur eine erste Spezifikation war und nicht sehr ausgereift, wurde bald eine neue Spezifikation mit neuen Anfragemethoden und einem Header herausgegeben.
Die neue Version HTTP/1.0 wurde Anfang 1996 mit der Request For Comments (RFC) 1945 [20] veröffentlicht.
Während der Dokumentation von 1.0 wurde schon an der nächsten Version, der Version 1.1, die heute noch die Aktuelle ist, gearbeitet. Im Juni 1999 wurde dann die Version HTTP/1.1 unter RFC 2616: Hypertext Transfer Protocol -- HTTP/1.1 [21] und RFC 2617: HTTP Authentication: Basic and Digest Access Authentication [22] sowie die Errata [23] zu RFC 2616 und RFC 2617 veröffentlicht [24].

3.3.2 eXtensible Markup Language (XML)

Bei den Dokumenten, die bei SOAP zum Beispiel über HTTP ausgetauscht werden, handelt es sich um ein XML-Derivat. Die eXtensible Markup Language (XML) dient also als Austauschformat zwischen Client und Server.
XML ist ein Textformat, welches aus der Standard Generalized Markup Language (SGML) ISO 8879 von 1986 hervorgegangen ist. Ursprünglich diente es dem Veröffentlichen und dem Austausch großer Dokumente [25].
Da SGML Probleme hatte sich durchzusetzen, wurde ein Subset definiert und 1998 als neuer Standard des World Wide Web Committee (W3C) veröffentlicht.

Die Unterschiede zwischen SGML und XML liegen hauptsächlich in der Mächtigkeit. SGML bietet eine größere Anzahl Möglichkeiten der Anpassung und ist somit flexibler als XML. Der Vorteil von XML der sich daraus ergibt ist die einfachere Integration in eigene Projekte. Durch seinen geringeren Umfang ist XML leichter und schneller zu integrieren.
Von da an entwickelte sich XML zu dem Austauschformat schlechthin, wenn es über Systemgrenzen hinweg Daten auszutauschen gilt. So kamen zu XML ständig neue Fromate, die auf XML beruhen hinzu, wie Scalable Vector Graphics (SVG), Synchronized Multimedia Integration Language (SMIL), Mathematical Markup Language (MathML) und letztlich auch SOAP sowie eine sehr große Anzahl von XML-Formaten, die keiner eigenen Standarisierung unterliegen [26].

3.3.3 Simple Object Access Protocol (SOAP)

Die Entwicklung von SOAP begann 1998. Zu diesem Zeitpunkt war SOAP noch keine Sprache mit einem definierten Schema bzw. einem Typen System für XML, da erst zum gleichen Zeitpunkt die Empfehlung für XML 1.0 herausgegeben wurde.
Die früheren Versionen von SOAP inklusive die eXtended Markup Language-Remote Procedure Call (XML-RPC), welche ein Subset der Spezifikation von SOAP von 1998 ist, legten ihren Fokus auf die Definition eines Typen Systems.
Das originale Typen System von SOAP und XML-RPC besitzt eine handvoll primitiver Typen und Kompositionen, die durch den Namen referenziert werden, auch bekannt als „structs“, sowie Kompositionen referenzierbar durch eine Position z.B. „arrays“.

Ab 1999, nach einem Treffen von Microsoft, der treibenden Kraft hinter SOAP, IBM und anderen interessierten Firmen, bildete sich ein größeres Konsortium, dass mit der Version SOAP/1.1 eine Spezifikation heraus brachte, die dem World Wide Web Consortium (W3C) vorgelegt wurde.
In den Anfangszeiten von SOAP musste seitens Microsoft große Lobbyarbeit betrieben werden, um SOAP als standardisiertes, nachrichtenbasiertes XML-Protokoll zu etablieren.
„Das Problem war die Leute dazu zu bringen zu Denken, dass es gut war, auch wenn es von Microsoft kam.“ Sagte Andrew Layman, ein XML Architekt von Microsoft [27].
Im Sommer 2000 dann gewann SOAP eine größere Akzeptanz. Nach einigen Diskussionen und Protokoll Vorschlägen wurden die Ideen von Microsoft und IBM verflochten.

IBM trug ihre Network Accessible Service Specification Language (NASSL) bei und Microsoft offerierten ihre Service Description Language (SDL) und ihre SOAP Contract Language (SCL). Im Herbst 2000 wurden diese Spezifikationen zu einer Web Services Description Language (WSDL) zusammengeführt [28].

Heute wird die Entwicklung von SOAP von der Web Services Activity [29] vorangetrieben. Ihre Aktivitäten spalten sich in vier Gruppen auf: die Web Services Architecture Working Group [30], die XML Protocol Working Group [31], die Web Services Description Working Group [32] und die Web Services Choreography Working Group [33].
Dabei beschäftigt sich nur die XML Protocol Working Group des World Wide Web Consortium (W3C) mit SOAP. Alle anderen Gruppen bemühen sich um Standards rund um Web Services (WS). Web Services ist ein Standard, der die Kommunikation von Applikation zu Applikation auf Basis von SOAP beschreibt.
Aus diesen Aktivitäten gingen Standards wie WSDL, zur Beschreibung von Services, und UDDI, zum Finden von angebotenen Services, hervor.

3.3.4 Ausblick auf SOAP 1.2

Mittlerweile liegt die Version 1.2 von SOAP als Entwurf vor und wird wohl demnächst verabschiedet werden. Sie wird zum Beispiel ein Rollenkonzept einführen, welches die Sicherheit von WS deutlich erhöhen kann.

3.4 Architektur

SOAP ist kein Standard, der von Grund auf neu entwickelt wurde. Vielmehr hat man hier auf bewehrte Basistechnologien zurückgegriffen. So besteht SOAP aus einem Dokument, welches dem Austausch der Daten dient. Beim Transport der Dokumente vertraut man bei SOAP auf die Protokolle HTTP bzw. die verschlüsselte Variante HTTPS, SMTP sowie FTP. Grundsätzlich wären auch noch andere Protokolle möglich doch meistens greift man beim Transport auf HTTP zurück.

3.4.1 Protokolle

3.4.1.1 Transmission Control Protocol (TCP)

Das über 20 Jahre alte Protokoll welches von SOAP als Transportschicht verwendet werden kann ist das Transmission Control Protocol (TCP). Es wurde im September 1981 in der RFC 793 [34] beschrieben und als Standard für das Internet verabschiedet. Weshalb es auch den Beinamen Internet Protocol (IP) trägt.
Da das TCP kein nachrichtenorientiertes Protokoll ist werden die SOAP Nachrichten bei TCP im TCP-Stream übertragen. Auf TCP bauen verbindungsorientierte Anwendungsprotokolle wie HTTP(S) auf.

3.4.1.2 HyperText Transfer Protocol (Secure) (HTTP(S))

Das am weitesten verbreitete Protokoll für SOAP ist das Protokoll HTTP bzw. seine verschlüsselte Variante HTTPS. HTTP besitzt den Vorteil, dass genügen Anwendungen gibt, die HTTP verstehen. So kann man von jedem Browser eine SOAP Anfrage starten.
Das ist natürlich nicht sehr sinnvoll, den als Antwort erhält man nur ein SOAP-Dokument, welches sich aber auf dem Browser ganz gut darstellen lässt, weil es ja ein XML konformes Dokument ist und die meisten Browser mit XML ganz gut klar kommen. Natürlich lässt sich auch für die Antwort des SOAP-Servers eine eXtensible Stylesheet Language Transformation (XSLT) mit dem zu erwartenden Dokument ablegen, sodass eine strukturiertere Darstellung der Antwort als HTML-Dokument möglich ist.
Somit eignet sich das Transport Protokoll HTTP sehr gut zum Testen von Web Services (WS).
Zudem bietet HTTPS durch die Verschlüsselung wenigstens einen sicheren Übertragungsweg, der Heute von vielen Web Servern unterstützt wird. Denn Leider gibt es im SOAP Standard bisher noch keine Möglichkeit sich gegenüber dem Web Service zu authentifizieren. Es sei denn man nutzt die Authentifizierung von HTTP oder übergibt im Anfrage-Dokument ein Login und Passwort mit, dass der Web Service auswertet.

Eine weitere Möglichkeit wäre einen Proxy dazwischen zu schalten der nur Zugriffe über ein Login und Passwort zulässt.
Wünschenswert wäre allerdings ein Standard, der dies von sich aus schon vorsieht, damit nicht jeder Entwickler sich erneut um dieses Thema Gedanken machen muss.
Einen Schritt zu mehr Sicherheit bietet der Vorschlag zur neuen Spezifikation SOAP/1.2. Dieser sieht die Einführung von Rollen vor. So können nur SOAP-Anfragen bearbeitet werden, die eine bestimmte Rolle besitzen. Eine Rolle lässt sich dann durch der Wert eines Uniform Resource Identifiers (URI) zuweisen [35], [36].

3.4.1.3 User Datagram Protocol (UDP)

Im Gegensatz zu TCP ist das User Datagram Protocol (UDP) ein nachrichtenorientiertes Protokoll, dessen Standard in der RFC 768 [37] gut ein Jahr früher, im August 1980, verabschiedet wurde.
Das Protokoll ist Grundlage vieler Dienste im Internet, so wie die beiden nachfolgenden, verbindungslosen Protokolle SMTP und FTP.
Daraus lässt sich schon der Grundlegende Unterschied zu TCP erkennen. Bei den UDP basierten Protokollen kann nicht garantiert werden, dass die Nachricht vollständig beim Empfänger ankommt.

3.4.1.4 Simple Mail Transfer Protocol (SMTP)

Aus dem heutigen Alltag nicht mehr wegzudenken ist das Simple Mail Transfer Protocol (SMTP). Es wurde im August 1980 als RFC 821 [38] verabschiedet und ist wohl neben dem Protokoll HTTP das meistgenutzte Protokoll.

3.4.1.5 File Transfer Protocol (FTP)

Das File Transfer Protocol (FTP) ist ein vergleichsweise altes Protokoll. Es wurde 1985 von der Network Working Group als RFC 959 [39] veröffentlicht.

Der Nachteil dieses Protokolls ist, dass es keine Verschlüsselung besitzt. Auch Logins und Passwörter werden im Klartext über das Transmission Control Protocol (TCP) übertragen.
Zwar lässt sich mittlerweile diesem Manko auch Abhilfe schaffen aber die sichere Variante von FTP, Secure File Transfer Protocol (SFTP), ist nicht sehr weit verbreitet.
Bei SFTP wird über eine Secure SHell (SSH) Verbindung getunnelt. Dies setzt einen SSH-Server voraus.
Anhand der nächsten Abbildung 11 kann man die Einordnung der Transportprotokolle in das OSI Referenzmodell ersehen.

Anwendungsschicht	Applikation	
Sitzungsschicht	SOAP	
Darstellungsschicht	HTTP(S)	SMTP, (S)FTP
Transportschicht	TCP	UDP
Netzwerkschicht	IP	
Mediumzugriffsschicht	MAC	
Bitübertragungsschicht	PHY	

Abbildung 11: Open System Interconnection (OSI) Referenzmodell

3.4.2 Dokumente

Die Grundlage der Kommunikation zwischen Client und Server bilden die Dokumente, die Mittels verschiedener Protokolle ausgetauscht werden können. Diese Dokumente sind XML-konforme Dokumente, dessen Schema sich in der SOAP Spezifikation wieder finden. Ein genauerer Blick auf das SOAP Schema findet sich in den nächsten Kapiteln.

Warum hat man aber auf XML zurückgegriffen und nicht auf zum Beispiel Electronic Document Interchange (EDI) oder anderen universellen Formaten?

Vorteile von XML [40], [41]:

- Unterstützung in einer Vielzahl von Applikationen.
- Weit verbreitet im Internet.
- Kompatibel zu SGML (Superset von XML).
- ASCII Format und somit für den Menschen lesbar.

- Einfaches Format, leicht zu implementieren.
- Offener Standard des W3C.
- Lizenzfrei, plattformunabhängig und systemunabhängig.
- Selbstbeschreibend und erweiterbar.
- Möglichkeit von Kontextinformationen.
- Trennung von Inhalt und Präsentation.
- Unterstützung von mehreren Sprachen durch Unicode.

Nachteile von XML [42]:

- ASCII-Format, dadurch großer Overhead an Daten (Umgehung durch Kompression).
- Relativ langsame Verarbeitung durch das Parsen des Dokuments.
- Kein Standard auf Anwendungsebene.
- Dokumentenorientierte Metasprache, aber keine Modellierungssprache, weshalb es nicht möglich ist m:n Beziehungen darzustellen.
- Nicht objektorientiert und somit keine Vererbungsmechanismen.
- Schwaches Typsystem, weil es nur Text in Form von CDATA kennt und keine numerischen Typen und komplexe Typen besitzt.
- Um aus XML eine Modellierungssprache zu machen, wurde das Format in der Spezifikation zu SOAP festgelegt.

3.4.3 SOAP

Anfragen in SOAP gestalten sich bei HTTP als Transportprotokoll genau so wie Web-Anfragen. Nur das Dokumentenformat ist nicht HTML, sondern ein XML spezifisches Format – SOAP.
Die SOAP-Dokumentenbeschreibung ist ein XML Dokument und besteht aus drei Teilen. Dem verbindlichen Envelope, dem optionalen Header und dem verbindlichen Body.

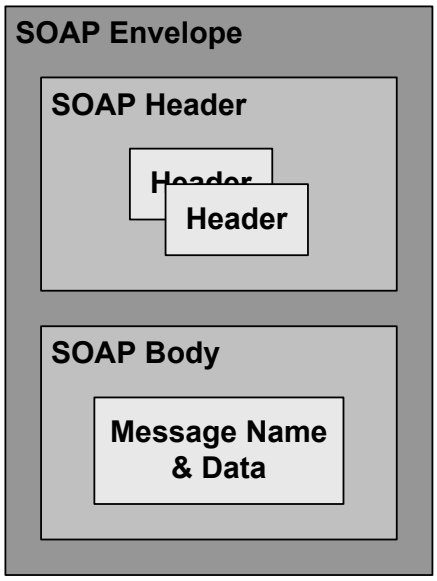

Abbildung 12: Darstellung der Bestandteile einer SOAP Nachricht

3.4.3.1 SOAP Envelope

Der Envelope (englisch: „Hülle“, „Umschlag“) ist das Top-Element des XML-Dokumentes, welches die Nachricht repräsentiert. Er ist somit eine Art einfacher Wrapper für die eigentliche Nachricht [43]. Er kann außerdem auch das Encoding der Nachricht spezifizieren. Die Spezifikation von SOAP definiert bereits ein Encoding [44]. Auch bekannt als RPC Encoding.
Dieses Encoding enthält einfache primitive Datentypen und Felder, Kompositionen dieser Datentypen. Auch andere Encodings können benutzt werden inklusive Nutzerspezifische [45].

```
<soapenv:Envelope
  xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/"
  xmlns:xsd="http://www.w3.org/2001/XMLSchema"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
…
</soapenv:Envelope>
```

Abbildung 13: SOAP Nachricht - Envelope

3.4.3.2 SOAP Header

Der Header einer SOAP Nachricht kann optionale Informationen enthalten z.B. Informationen zur Authentifizierung und Autorisierung, Session- und Transaktionsmanagment.
Der SOAP-Header ist sehr flexibel zu benutzen und besitzt nur ein paar vordefinierte Attribute, die aber nur selten verwendet werden [46].

```
<soapenv:Envelope
  xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/"
  xmlns:xsd="http://www.w3.org/2001/XMLSchema"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
  <soapenv:Header>
…
  </soapenv:Header>
…
</soapenv:Envelope>
```

Abbildung 14: SOAP Nachricht - Header

3.4.3.3 SOAP Body

Der Body ist ein Container für die verbindlichen Informationen für den Empfänger der Nachricht. SOAP definiert nur ein Element im Body, das Fault-Element (englisch: „Fehler“). Dieser enthält den Fehler-Kode und eine Beschreibung des Fehlers vom Server bei misslingen der Anfrage.

Ein SOAP-Body unterstützt zwei Formate und eine Mischform aus diesen zwei Formaten: RPC und Document (MSG, Message vorrangiger Sprachgebrauch von Apache axis).

Das Format RPC, welches auch als RPC/encoded bezeichnet wird, stützt sich auf die gleichen Prinzipien wie CORBA und RMI. Die Aufrufparameter und das Resultat werden im Body kodiert übertragen. Eine Beschreibung der Ein- und Ausgabeparameter einer Methode erfolgt immer im Body der Nachricht als Attribut eines XML-Tags.

Im Unterschied dazu wird beim Format Document, oder auch Document/literal, ein XML-Dokument übertragen. Beim Service wird keine Methode aufgerufen wie beim RPC sondern ein XML-Dokument wird verarbeitet. Dazu müssen an jedem Ende die Daten erst in die jeweiligen Datentypen der Zielprogrammiersprache konvertiert werden [47].

Die dritte Form, auch als RPC/literal bezeichnet, ist eine Mischform aus Beiden. Im Prinzip ist es ein RPC Mechanismus, bei dem die Aufrufparameter der Methode ein Dokument ist. Der Nachteil von RPC/literal gegenüber Document/literal ist, dass das Schema der RPC/literal Nachricht nicht dazu verwendet werden kann die eigentliche Nachricht zu validieren. Denn die Nachricht wird ja von der RPC-Methode umhüllt. Würde man diese entfernen, wäre das dann eine andere Form, nämlich Document/literal [48].

```
<soapenv:Envelope
  xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/"
  xmlns:xsd="http://www.w3.org/2001/XMLSchema"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
  <soapenv:Body>
    <ns1:getImage
      soapenv:encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
      xmlns:ns1="urn:ImageServiceRPC">
      <filename xsi:type="xsd:string">C:\16 B Testdatei</filename>
    </ns1:getImage>
  </soapenv:Body>
</soapenv:Envelope>
```

Abbildung 15: SOAP Nachricht – Body (RPC)

Eine komplette Anfrage inklusive HTTP/1.0-Header sieht wie folgt aus:

```
POST /axis/services/urn:ImageServiceRPC HTTP/1.0
Content-Type: text/xml; charset=utf-8
Accept: application/soap+xml, application/dime, multipart/related, text/*
User-Agent: Axis/1.1beta
Host: 127.0.0.1
Cache-Control: no-cache
Pragma: no-cache
SOAPAction: ""
Content-Length: 464

<?xml version="1.0" encoding="UTF-8"?>
   <soapenv:Envelope
xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/"
xmlns:xsd="http://www.w3.org/2001/XMLSchema"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
      <soapenv:Body>
         <ns1:getImage
soapenv:encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
xmlns:ns1="urn:ImageServiceRPC">
            <filename xsi:type="xsd:string">C:\16 B Testdatei</filename>
         </ns1:getImage>
      </soapenv:Body>
   </soapenv:Envelope>
```

Abbildung 16: Anfrage in SOAP

Für das SOAP Encoding, auch als Section 5 Encoding bezeichnet, nach dem Abschnitt in der Spezifikation von SOAP/1.0, gelten folgende Zuordnungen:

SOAP	Java
boolean	boolean/java.lang.Boolean
byte	byte/java.lang.Byte
base64Binary	byte[]
decimal	java.math.BigDecimal
double	double/java.lang.Double
float	float/java.lang.Float
int	int/java.lang.Integer
integer	java.math.BigInteger
long	long/java.lang.Long
short	short/java.lang.Short
string	java.lang.String
dateTime	java.util.Calendar/java.util.Date
QName	javax.xml.namespace.QName

Abbildung 17: Mapping SOAP/Java

3.4.3.4 SOAP Fault

Eine SOAP Fehler Nachricht (Fault) besteht aus vier Teilen. Der Faultcode ist ein Code, um den Fehler zu identifizieren, der Faultstring enthält eine lesbare Beschreibung des Fehlers, Faultactor enthält Imformationen darüber wer den Fehler erzeugt hat und Detail beinhaltet eine anwendungsspezifische Fehlermeldung für das Body Element [49].

```
<soapenv:Envelope
  xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/"
  xmlns:xsd="http://www.w3.org/2001/XMLSchema"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
  <soapenv:Header/>
  <soapenv:Body>
    <soap:Fault>
      <faultcode>…</faultcode>
      <faultstring>…</faultstring>
      <faultactor>…</faultactor>
      <detail>…</detail>
    </soap:Fault>
  </soapenv:Body>
</soapenv:Envelope>
```

Abbildung 18: SOAP Nachricht – Fehler

3.4.3.5 SOAP Attachment

Attachments (englisch: „Anhang“, „Anlage“) sind in SOAP auch möglich. Die Schwierigkeit bei den Anhängen liegt darin, dass SOAP ein XML Dokument ist und die Spezifikation für XML nur ein Subset von Zeichen von US-ASCII zulässt. Deshalb muss alles was im Zeichenvorrat nicht diesem Subset entspricht kodiert werden. Dies geschieht mittels Base64 [50].

Das Format für diese kodierten Daten ist das gleiche Format wie bei der Übertragung von Bildern und anderen Anhängen in E-Mails verwendet wird. Es ist das Multipurpose Internet Mail Extensions (MIME) Format RFC 2045 [51], RFC 2046 [52] und RFC 2387 [53].

Eine komplette Antwort inklusive HTTP/1.1-Header mit Anhang als MIME:

```
HTTP/1.1 200 OK
Content-Type: multipart/related; type="text/xml";
start="<3BA4D8C876DB2B4C111A372FD854651F>";    boundary="----
=_Part_5_32803057.1048808908343"Date: Thu, 27 Mar 2003 23:48:29 GMTServer:
Apache Coyote/1.0Connection: close

------=_Part_5_32803057.1048808908343
Content-Type: text/xml; charset=UTF-8
Content-Transfer-Encoding: binary
Content-Id:
   <3BA4D8C876DB2B4C111A372FD854651F>
      <?xml version="1.0" encoding="UTF-8"?>
         <soapenv:Envelope
xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/"
xmlns:xsd="http://www.w3.org/2001/XMLSchema"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
            <soapenv:Body>
               <ns1:getImageResponse
soapenv:encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
xmlns:ns1="urn:ImageServiceRPC">
                  <return href="cid:F279414F5DD90191644D34E06F6D40AF"/>
               </ns1:getImageResponse>
            </soapenv:Body>
         </soapenv:Envelope>
------=_Part_5_32803057.1048808908343
Content-Type: application/octet-stream
Content-Transfer-Encoding: binary
Content-Id:
         <F279414F5DD90191644D34E06F6D40AF>0123456789ABCDEF
------=_Part_5_32803057.1048808908343--
```

Abbildung 19: Antwort in SOAP (Binärdaten via MIME)

Ein weiteres Format, welches schon von axis unterstützt wird, aber nicht im Standard SOAP/1.1 definiert ist, ist Direct Internet Message Encapsulation (DIME) initiiert von Microsoft. Dieses Format ist noch nicht verabschiedet und liegt als Draft-Version bei der RFC. DIME wird ab der Version SOAP/1.2 als Format für Attachments akzeptiert werden.

Der Unterschied zu MIME ist, dass mit DIME keine Kodierung der Binärdaten notwendig ist. Das macht DIME natürlich um einiges schneller. Wie MIME bietet auch DIME die Möglichkeiten von Inhaltstypen („image/jpeg“, „application/octet-stream“ oder „text/html“). Diese sind von der Internet Assigned Numbers Authority (IANA) vorgegeben und können nicht adhoc durch neue Typen ergänzt werden. Dieses Manko beseitigt DIME. Mit DIME ist es möglich neu Inhaltstypen per URI zu spezifizieren [54], [55].

Eine komplette Antwort inklusive HTTP/1.1-Header mit Anhang als DIME:

```
HTTP/1.1 200 OK
Content-Type: application/dime
Date: Fri, 28 Mar 2003 00:17:49 GMT
Server: Apache Coyote/1.0
Connection: close

    ) )  Õuuid:714C6C40-4531-442E-A498-3AC614200295
http://schemas.xmlsoap.org/soap/envelope/   <?xml version="1.0"
encoding="UTF-8"?>
   <soapenv:Envelope
xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/"
xmlns:xsd="http://www.w3.org/2001/XMLSchema"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
      <soapenv:Body>
         <ns1:getImageResponse
soapenv:encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
xmlns:ns1="urn:ImageServiceRPC">
            <return href="D346863778E16494B69945ABC9F06694"/>
         </ns1:getImageResponse>
      </soapenv:Body>
   </soapenv:Envelope>
D346863778E16494B69945ABC9F06694application/octet-stream0123456789ABCDEF
```

Abbildung 20: Antwort in SOAP (Binärdaten via DIME)

Eine dritte Variante wird mit SOAP/1.2 möglich sein. Diese ist wohl die Einfachste, weil sie es die Anhänge quasi persönlich abzuholen erlaubt nach dem Stellen der Anfrage und dem Empfang der Antwort mittels z.B. HTTP GET. Dazu wird einfach die URL aus der Antwort herausgelesen und der „Anhang“ abgeholt [56].

3.4.4 WSDL

Das Kommunikationsprotokoll und das Nachrichtenformat sind hinreichend spezifiziert. Nun fehlt nur noch eine Beschreibung der Kommunikation. Die Lösung dafür soll die Web Services Description Language (WSDL) [57] sein. Dieser Standard gehört nicht unmittelbar zu SOAP ist aber eine nützliche Erweiterung der Web Services Activty-Gruppe.
WSDL ist ein XML-Format zu Beschreibung von Netzwerkservices als ein Satz von Endpunkt-Operationen mit Hilfe von Nachrichten. Diese Nachrichten können entweder dokumentorientierte oder prozedurorientierte Informationen enthalten.
In einer WS Beschreibung sind folgende Informationen über einen WS enthalten:

Tag	Beschreibung
Types	Ein Container für Datentypendefinitionen.
Message	Eine abstrakte Typendefinition der Daten die kommuniziert werden sollen.
Operation	Eine abstrakte Beschreibung der Aktion, die vom Service unterstützt wird.
Port Type	Eine abstrakte Beschreibung der Operationen, die von einem oder mehreren Endpunkten unterstützt wird.
Binding	Ein konkretes Protokoll und Datenformatspezifikation für einen bestimmten Porttyp.
Port	Ein Endpunkt mit definierter Kombination aus Binding und Netzwerkadresse.
Service	Eine Kollektion von ähnlichen Endpunkten.

Abbildung 21: WSDL Dokumentenstruktur

Ein wichtige Eigenschaft dieser Beschreibungssprache ist die automatische Proxy Generierung durch Port, Binding und Port Type.

```
<?xml version="1.0" ?>

<definitions name="urn:AddressFetcher"
             targetNamespace="urn:AddressFetcher2"
             xmlns:tns="urn:AddressFetcher2"
             xmlns:typens="urn:AddressFetcher2"
             xmlns:xsd="http://www.w3.org/1999/XMLSchema"
             xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/"
             xmlns="http://schemas.xmlsoap.org/wsdl/">

  <!-- type defs -->
  <types>
    <xsd:schema targetNamespace="urn:AddressFetcher2"
                xmlns:xsd="http://www.w3.org/1999/XMLSchema">

      <xsd:complexType name="address">
        <xsd:all>
          <xsd:element name="streetNum" type="xsd:int"/>
          <xsd:element name="streetName" type="xsd:string"/>
          <xsd:element name="city" type="xsd:string"/>
          <xsd:element name="state" type="xsd:string"/>
          <xsd:element name="zip" type="xsd:int"/>
          <xsd:element name="phoneNumber" type="xsd:string"/>
        </xsd:all>
      </xsd:complexType>
    </xsd:schema>
  </types>

  <!-- message declns -->
  <message name="AddEntryRequest">
    <part name="name" type="xsd:string"/>
    <part name="address" type="typens:address"/>
  </message>
…
  <!-- port type declns -->
  <portType name="AddressBook">
    <operation name="addEntry">
      <input message="tns:AddEntryRequest"/>
    </operation>
…
```

```
  </portType>

  <!-- binding declns -->
  <binding name="AddressBookSOAPBinding" type="tns:AddressBook">
    <soap:binding style="rpc"
      transport="http://schemas.xmlsoap.org/soap/http"/>
    <operation name="addEntry">
      <soap:operation soapAction=""/>
      <input>
        <soap:body use="encoded" namespace="urn:AddressFetcher2"
          encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"/>
      </input>
      <output>
        <soap:body use="encoded" namespace="urn:AddressFetcher2"
          encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"/>
      </output>
    </operation>
…
    </operation>
  </binding>

  <!-- service decln -->
  <service name="AddressBookService">
    <port name="AddressBook" binding="tns:AddressBookSOAPBinding">
      <soap:address
location="http://localhost:8080/axis/services/AddressBook"/>
    </port>
  </service>

</definitions>
```

Abbildung 22: WSDL Beispiel AddressFetcher (axis samples)

3.4.5 UDDI

Ein weiterer Standard, der sich aus der WS Activity-Gruppe herausgebildet hat ist der Standard Universal Description, Discovery and Integration (UDDI). Dieser Standard soll es ermöglichen seine entwickelten WS zu veröffentlichen. Im Prinzip sind UDDI die „Gelben Seiten“ von WS.

Nach dem registrieren eines WS bei einem Registrar (IBM, Microsoft oder SAP u.a.) und der Veröffentlichung der WSDL kann man den Registrierservice nach diesem und anderen WS durchsuchen. Hat man einen passenden WS gefunden kann man sich die Referenz auf die WSDL vom Server geben lassen und mit Hilfe der WSDL-Datei einen Klient generieren, der den WS in Anspruch nimmt und die entsprechenden Daten liefert, die gesucht wurden.

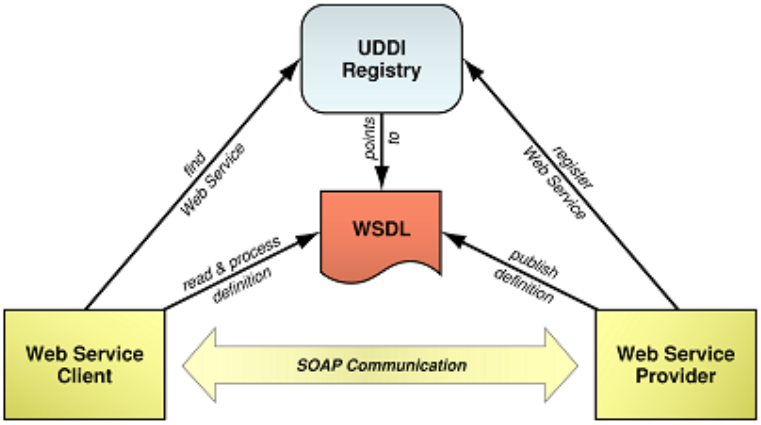

Abbildung 23: Zusammenhang zwischen WS, WSDL und UDDI

Die Bedeutung bei der Nutzung von öffentlichen WS mag noch nicht sehr groß sein, da es noch keinen automatisierten Weg gibt die gewonnen Informationen zu nutzen – man muss die Daten immer noch selber bewerten. Aber bei einem Einsatz im Unternehmen, wo jede Abteilung spezifische WS anbietet kann UDDI durchaus hilfreich sein.

3.5 Beispiel in Java

Nachfolgend sind die Implementierungen eines Services und zweier Clients zu sehen.

```
…
public class ImageService {
  public DataHandler getImage(String fileName) {
    // Attachments are sent back by default as a MIME stream if no
attachments
    // were received. If attachments are received the same format that was
    // received will be the default stream type for any attachments
    // sent back.
    // The following two commented lines would force any attachments
    // sent back to be in DIME format.
    Message rspmsg =
AxisEngine.getCurrentMessageContext().getResponseMessage();
    rspmsg.getAttachmentsImpl().setSendType(Attachments.SEND_TYPE_DIME);

    // Create the data handler for the attached file.
    DataHandler dataHandler = new DataHandler(new
FileDataSource(fileName));

    return dataHandler;
  }
}
```

Abbildung 24: ImageService.java

Die beiden Clients sind die Implementierungen des gleichen RPC/encodeden, entfernten Aufrufes. Der erste Client benutzt dabei das axis API von Apache. Der Zweite das von Sun favorisierte API SOAP with Attachments API for Java (SAAJ).

```
…
public class RPCClient {
  public static void main(String[] args) {
    try {
      // Create the data handler for the attached file.
      Service service = new Service();
      Call call = (Call) service.createCall();

      // Set the target service host and service location.
      call.setTargetEndpointAddress(
        new URL("http://host:8080/axis/services/urn:ImageServiceRPC"));

      // This is the target services method to invoke.
      call.setOperationName(new QName("urn:ImageServiceRPC", "getImage"));
      QName qNameFile = new QName("urn:ImageServiceRPC", "DataHandler");

      // Add serializer/deserializer for attachment.
      call.registerTypeMapping(DataHandler.class, qNameFile,
        JAFDataHandlerSerializerFactory.class,
        JAFDataHandlerDeserializerFactory.class);

      // Add the file name.
      call.addParameter("filename", XMLType.XSD_STRING, ParameterMode.IN);
      call.setReturnType(qNameFile);
      call.setUsername("");
      call.setPassword("");

      String fileName = args[0];
      // Add the parameter value.
      Object ret = call.invoke(new Object[] { fileName });
      if (ret == null) {
        System.out.println("Received null ");
        throw new AxisFault("", "Received null", null, null);
      }
      if (ret instanceof String) {
        System.out.println(
          "Received problem response from server: " + ret);
        throw new AxisFault("", (String) ret, null, null);
      }
      if (!(ret instanceof DataHandler)) {
```

```
        // The wrong type of object that what was expected.
        System.out.println(
          "Received problem response from server:"
            + ret.getClass().getName());
        throw new AxisFault("", "Received problem response from server:"
            + ret.getClass().getName(), null, null);
      }
      DataHandler rdh = (DataHandler) ret;
      // Get the filename.
      String receivedFileName = rdh.getName();
      if (receivedFileName == null) {
        System.err.println("Could not get the file name.");
        throw new AxisFault("", "Could not get the file name.",
          null, null);
      }
    }
    catch (Exception exception) {
      System.err.println("ERROR : " + exception);
      exception.printStackTrace();
    }
  }
}
```

Abbildung 25: RPCClient.java (axis API)

```
…
public class SAAJRPCClient {
  public static void main(String[] args) {
    try {
      SOAPConnectionFactory soapConnectionFactory =
        SOAPConnectionFactory.newInstance();
      SOAPConnection soapConnection =
soapConnectionFactory.createConnection();

      MessageFactory messageFactory = MessageFactory.newInstance();
      SOAPMessage soapMessage = messageFactory.createMessage();
      SOAPPart soapPart = soapMessage.getSOAPPart();

      String fileName = args[0];
      SOAPEnvelope requestEnvelope = soapPart.getEnvelope();
      SOAPBody body = requestEnvelope.getBody();
```

```
      SOAPBodyElement bodyElement =
        body.addBodyElement(
          requestEnvelope.createName("getImage", "ns1",
"urn:ImageServiceRPC"));
      SOAPElement element =

bodyElement.addChildElement(requestEnvelope.createName("filename"));
      element.addTextNode(fileName);

      String targetEndpointAddress =
        "http://host:8080/axis/services/urn:ImageServiceRPC";
      SOAPMessage returnedSOAPMessage =
        soapConnection.call(soapMessage, targetEndpointAddress);
      Iterator iterator = returnedSOAPMessage.getAttachments();
      if (!iterator.hasNext()) {
        // The wrong type of object that what was expected.
        System.out.println("Received problem response from server");
        throw new AxisFault("", "Received problem response from server",
          null, null);
      }
      DataHandler rdh =
        (DataHandler) ((AttachmentPart)iterator.next()).getDataHandler();

      // Get the filename.
      String receivedFileName = rdh.getName();
      if (receivedFileName == null) {
        System.err.println("Could not get the file name.");
        throw new AxisFault("", "Couldn't get the file name.", null, null);
      }
    }
    catch (Exception exception) {
      System.err.println("ERROR : " + exception);
      exception.printStackTrace();
    }
  }
}
```

Abbildung 26: SAAJRPCClient.java (SAAJ API)

3.6 Zusammenfassung

Die Komponententechnologie SOAP dient vor allem dem Austausch von Nachrichten und zum Aufruf von RPC in verteilten Systemen. Durch die Übertragung der Informationen im plattformunabhängigen Format XML prädestiniert sich SOAP gerade dafür plattform- und sprachenunabhängig zu entwickeln.
Durch den Einsatz von Basistechnologien für das Internet kann bei SOAP auf teure Middleware, wie bei CORBA der ORB, verzichtet werden. Nachteilig wirkt sich nur der große Overhead an Daten durch XML auf die Verarbeitungsgeschwindigkeit aus.
Die zurzeit noch rudimentären Sicherheitsmechanismen werden mit der kommenden Version von SOAP verbessert werden. Ein großer Vorteil von SOAP liegt darin, dass SOAP durch Firewalls getunnelt werden kann.

4 Vorbereitungen und Installationen

Für die Untersuchung der Komponententechnologien CORBA und SOAP müssen einige Vorbereitungen getroffen werden, die in den nachfolgenden zwei Kapiteln beschrieben wird.

4.1 CORBA

Zur Untersuchung von CORBA gibt es eine ganze Reihe von Implementationen. Die Bekannteste und wohl auch am meisten verbreitete ORB-Implmentierung ist Orbix von IONA [58].
Die Orbix Produktlinie ist eine der führenden CORBA e-Business Infrastrukturen. Mit tausenden Kunden, die Orbix 3, OrbixWeb 3 und ObrixOTM 3 in zeitkritischen Applikationen in den Schlüsselindustrien wie Finanzen, Telekommunikation, Banken, Regierungen, Informationstechnologie und im Krankenwesen einsetzen [59].
Andere Implementationen sind zum Beispiel: Paragon Software OAK CORBA 2.0 ORB [60], Vertel e*ORB 2.0 [61] oder Olivetti & Oracle Research Laboratory OmniORB 2.6.1 [62].
Da Res Medicinae aber unter der GNU General Public Licence (GPL) steht, kommt ein kommerzielles Produkt für dieses Projekt nicht in Frage.

Die bekannteste Implementierung ist sicherlich JacORB [63]. JacORB steht unter der GNU Library General Public Licence (GLGPL) [64] und wäre für das Projekt einsetzbar. Die einzige Einschränkung gegenüber der GPL ist, dass man den Code von JacORB nicht verändern darf. Da man das Produkt nur frei benutzen will und nicht selber anzupassen braucht, ist dies aber nicht wirklich eine Einschränkung.

Für die Testimplementationen und die Integration in Res Medicinae habe ich mich aber gegen JacORB entschlossen, weil JacORB einen höheren Aufwand für die Installation und die Administration des ORB's bedeutet und ziemlich umfangreich ist.
Zudem wird bei jeder Installation von Sun Microsystem's Java 2 Runtime Environment (J2RE), Java 2 Platform, Standard Edition (J2SE) oder Java 2 Platform, Enterprise Edition (J2EE) ein eigener ORB mitgeliefert.
Nach der Installation einer Sun Java Virtual Machine (JVM) muss der Dienst nur noch gestartet werden. Vorausgesetzt das Home-Verzeichnis von Java ist in der Systemumgebung eingetragen, lässt sich der ORB wie folgt starten:

```
%Java_Home%\bin\orbd -ORBInitialPort 1050 -ORBInitialHost server
```

Der ORBInitialPort ist variabel, sollte aber standardmäßig einen Port größer 1024, hier Port 1050 verwenden. Der Parameter ORBInitalHost gibt den Server an, auf dem der Dienst (ORBDaemon) installiert werden soll.
Dieser Dienst enthält folgende Service: Bootstrap Service, einen Transient Naming Service und einen Persistent Naming Service. Diese Service dienen zum Auffinden eines registrierten Dienstes im Netz.
Nach dem der Daemon läuft kann man den Server dem Naming Service bekannt machen:

```
%Java_Home%\bin\java org.resmedicinae.application.healthcare.review
.corba.server.ImageServer -ORBInitialPort 1050 -ORBInitialHost server
```

Nun ist der ImageServer bereit Anfragen von einem Client zu empfangen.

```
%Java_Home%\bin\java org.resmedicinae.application.healthcare.review
.corba.client.Client -ORBInitialPort 1050 -ORBInitialHost server
```

Der Client kann sich nun lokal auf der Maschine oder irgendwo im lokalen Netz bzw. im Internet befinden.

4.2 SOAP

Für die Installation eines SOAP-Servers gibt es bei Weitem noch nicht die Vielfalt wie bei den ORB's. So beschränkte sich die Suche nach einem brauchbaren Server eigentlich nur auf ein Projekt - Apache [65].
Die Apache Software Foundation (ASF) ist ein Open-Source Projekt mit einer Vielzahl von anderen Projekten, wie HTTP Server, Ant, Jakarta, Perl, PHP, TCL, SOAP, Web Services und XML.
Alle Projekte der ASF stehen unter der Apache Software License. Diese Lizenz beinhaltet die freie Nutzung der Produkte mit unwesentlichen Einschränkungen, die die Nutzung für das Projekt Res Medicinae erlauben.
Als erstes fällt einem natürlich das Projekt SOAP von Apache ins Auge, wenn man nach einem SOAP-Server sucht. SOAP wird aber nicht mehr wesentlich weiter entwickelt, sondern geht voll und ganz im neuen Projekt namens axis auf.
Axis ist ein SOAP Container für Jakarta Tomcat, einem Servlet Container. Es muss also ein Apache Web-Server mit Tomcat aufgesetzt werden und anschließend kann ganz einfach axis installiert werden.
Die Apache Installation gestaltet sich einfach. Unter Windows gibt es zum Beispiel eine eigene Installationsroutine. Einfach den Anweisungen folgen und ins gewünschte Verzeichnis installieren.
Danach kann man Tomcat eben so einfach mittels eines Wizards installieren. Alle notwendigen Einstellungen zur Konfiguration von Apache für Tomcat können über diesen Wizard erledigt werden.
Funktioniert Tomcat wie gewünscht, kann man nun axis installieren. Dazu muss man nur den gesamten Inhalt der Distribution entpacken und alles unter dem Verzeichnis

```
%axis_Home%\webapps
```

nach

```
%Tomcat_Home%\webapps
```

kopieren.

Zusätzlich müssen alle Bibliotheken, die von axis außerdem noch verwendet werden ins Verzeichnis

```
%Tomcat_Home%\common\lib
```

kopiert werden. Tomcat findet sie bei jedem Neustart wieder und trägt sie von Selbst in den Klassenpfad ein.

Folgende Bibliotheken müssen bzw. können mit kopiert werden:

Bibliothek	Name der Bibliothek	Optional
activation.jar	Sun Microsystems JavaBeans Activation Framework 1.0.2	
axis.jar	Apache Group axis 1.1	
axis-ant.jar	Apache Group axis 1.1 (Apache Group Ant 1.5.1)	
commons-discovery.jar	Apache Group axis 1.1	
commons-logging.jar	Apache Group axis 1.1	
jaxrpc.jar	Apache Group axis 1.1	
log4j-1.2.4.jar	Apache Group axis 1.1 (Apache Group Log4J 1.2.4)	
mail.jar	Sun Microsystems JavaMail 1.3	x
saaj.jar	Apache Group axis 1.1	
wsdl4j.jar	Apache Group axis 1.1	
xmlsec.jar	Apache Group XML-Security 1.0.4	x

Abbildung 27: axis Bibliotheken

Der Apache HTTP-Server und Tomcat lassen sich als Service registrieren, so dass der SOAP-Server automatisch bei jedem Neustart automatisch gestartet wird.

4.3 ImageJ

Um ein praktisches Beispiel für die Arbeit zu geben, wurde ein Prototyp in CORBA sowie in SOAP entwickelt. Als Gerüst dafür wurde ImageJ [2], [66] benutzt.
ImageJ ist eine Public Domain (PD) Applikation in Java, welche durch NIH Image [67], einem Bildbearbeitungsprogramm für den Mac, inspiriert wurde. Es ist ein Bildbearbeitungs-Toolkit, dessen Erweiterungen in Quellen für den nicht kommerziellen Einsatz kostenlos sind.

Dadurch das ImageJ in Java geschrieben wurde ergeben sich Vorteile wie die Plattform- und Systemunabhängigkeit. Es unterstützt folgende Graphikformate (Lesen und Schreiben) TIFF (unkomprimiert), GIF, JPEG, ASCII sowie (Lesen) BMP, DICOM (unkomprimiert) und FITS.
ImageJ unterstützt Plug-In's für das Lesen und Schreiben neuer Bildformate sowie für das Manipulieren von Bildern. Von sich aus besitzt ImageJ verschiedene Filter für das Glätten, Schärfer stellen, Erkennen von Kanten von Bildern usw. [68].

Für die Demonstration von CORBA und SOAP wurden zwei neue Plug-In's geschrieben. Jeweils eins für CORBA und eins für SOAP. Zu finden sind diese unter:

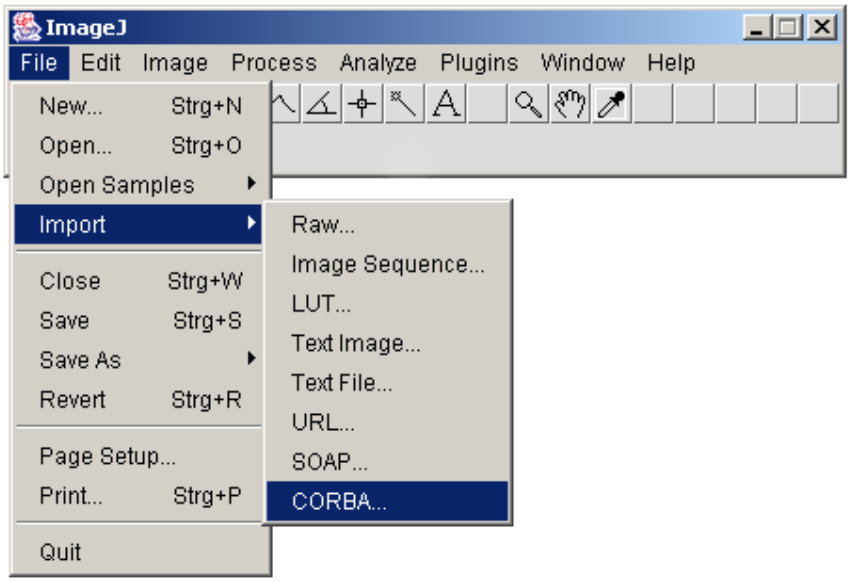

Abbildung 28: Import von Bildern via CORBA in ImageJ 1

Abbildung 29: Import von Bildern via CORBA in ImageJ 2

Zu konfigurieren sind der CORBA Server und der SOAP Client unter:

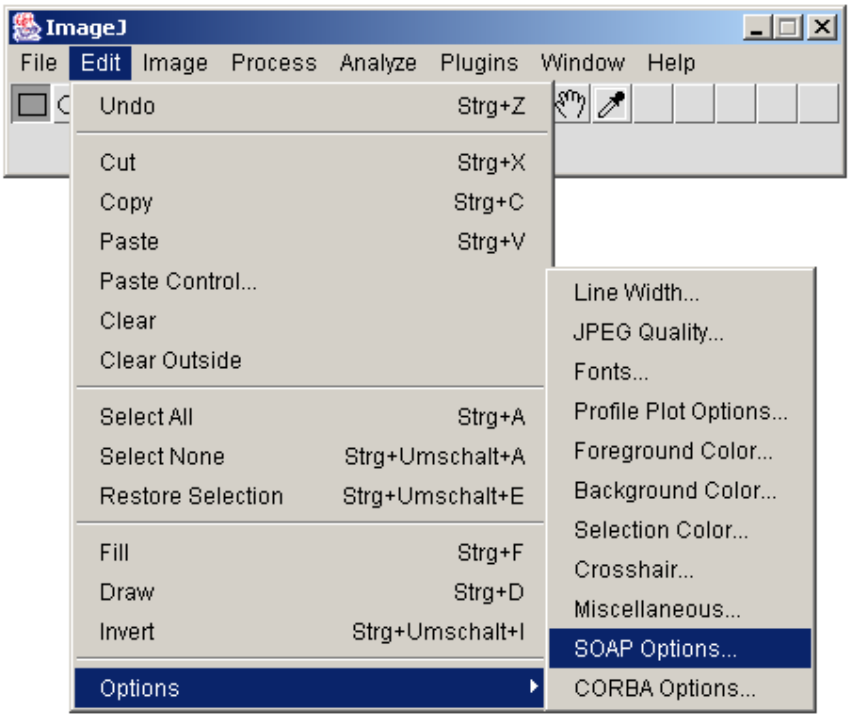

Abbildung 30: Optionen SOAP Service/CORBA Server in ImageJ

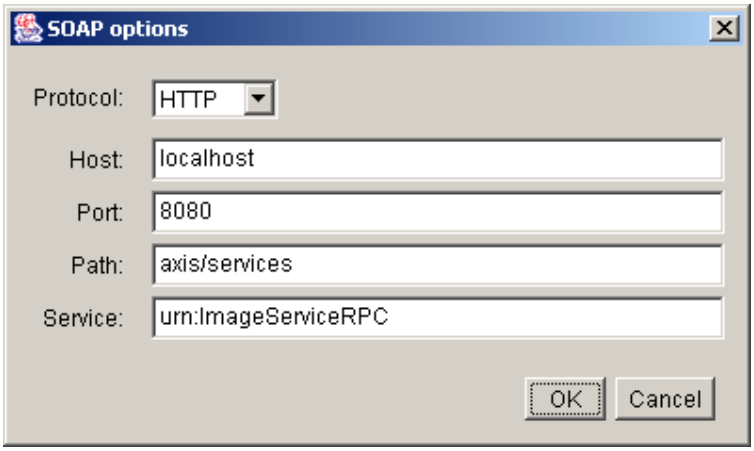

Abbildung 31: Optionen SOAP Service

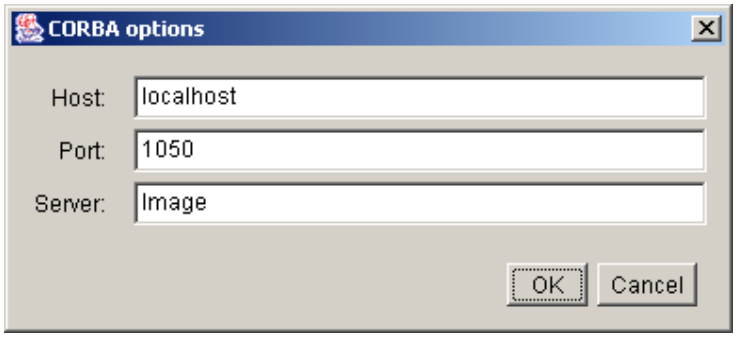

Abbildung 32: Optionen CORBA Server

5 Vergleich von CORBA und SOAP

Der Vergleich von CORBA und SOAP ist eigentlich nur schwer möglich, da SOAP keine verteilte Objektarchitektur ist. Aber für den Einsatzzweck in der medizinische Bildbearbeitung der hier betrachtet wird, nämlich der Austausch von Informationen wie Bildern aller Art, ist der Vergleich durchaus zulässig.
Beim Bearbeiten von Bildern am lokalen Rechner sollte die Leistung von heutigen Rechnern ausreichend sein, um einfache und komplexere Bildbearbeitungen vornehmen zu können wie Helligkeit ändern oder Klassifikationen. Selbst Volume Rendering dürfte für die meisten Rechner kein Problem mehr darstellen.
Wenn man aber zum Beispiel neuronale Netze trainieren möchte oder kleine Sequenzen rendern möchte wäre doch ein Cluster von Rechnern nötig, um in erträglicher Zeit damit fertig zu werden. Dafür bietet sich dann CORBA an, weil das die eigentliche Domäne von CORBA ist.
Kommt es also auf das verteilte Rechnen an, sollte SOAP außer Acht gelassen werden bei der Wahl der verwendeten Komponententechnologie.

5.1 Betriebswirtschaftliche Betrachtungen

Aus betriebswirtschaftlicher Sicht liegt der Vorteil auf Seiten von SOAP. Zwar gibt es auch freie und kostenlose ORB's, will man aber nicht auf kommerziellen Support verzichten, liegen die Kosten von CORBA doch über denen von SOAP. SOAP ist einfach die billigere Variante in monetärer als auch technologischer Hinsicht.
Zumeist ist außerdem schon ein Web-Server vorhanden, der entweder durch ein Update oder ein Plug-In dazu befähigt wird ein SOAP-Server zu werden.
Die Kosten für eine Implementierung eines CORBA Servers oder SOAP Service sind in etwa vergleichbar, da sich der Aufwand die Waage hält.

5.2 Betrachtungen aus Entwicklersicht

Programmiersprachen

Die Menge der von CORBA und SOAP unterstützten Programmiersprachen sind sicherlich ungefähr gleich. Bei Scriptsprachen sind mir allerdings nur SOAP Implementierungen bekannt. Der Schwerpunkt bei CORBA dürfte allerdings noch auf C/C++ liegen und der von SOAP auf Sun One und Java und Microsoft .NET und C#.

Implementierung

Aus Sicht des Entwicklers ist die Vorgehensweise nicht wirklich unterschiedlich, ob man nun einen CORBA Client/Server oder SOAP Client/Service implementiert.
Aus einer Schnittstellenbeschreibungssprache werden Kodegerüste für Client und Server bzw. Service generiert, die es gilt auszuimplementieren. Dabei ist es jedoch einfacher die IDL zu handhaben als WSDL. Der XML Overhead macht WSDL einfach ein wenig unleserlicher als die IDL.
Die Implementierung gestaltet sich dann nach einer umfangreichen Einarbeitung in das Thema für beide Technologien relativ gleich einfach. Was die Sache mit SOAP etwas komplizierter macht sind die unterschiedlichen API's. Einmal gibt es das API SOAP with Attachments API for Java (SAAJ) von Sun und das axis API von Apache sowie zig andere Varianten einen SOAP-Service anzufragen. Bedingt durch die Basistechnologien auf die SOAP aufbaut kann man eben auf vielfältige Weise einen Service von SOAP benutzen. Zum Beispiel reicht ein Tomcat Servlet Container und eine XML Engine, um einen Service effektiv und evtl. auch schneller als mit einem Framework wie das von axis zu nutzen.

Was man bei SOAP unbedingt beachten sollte ist die Art des SOAP Service, die man benutzen möchte. Also dokumentenbasiert (Document/literal) oder RPC-basiert (RPC/encoded) bzw. die Mischform dokumentenbasiertes RPC (RPC/literal).
Die Präferenzen des W3C erkennt man eindeutig in der neuen Spezifikation SOAP/1.2. In ihr sind die RPC basierten Methoden nicht mehr zwingend für einen SOAP-Service vorgeschrieben. Dennoch bin ich mir sicher, dass zumindest neue Versionen von axis, die

dann SOAP/1.2 vollständig unterstützen, die RPC nicht wieder ausbauen werden, weil es in Wahrheit so ist, dass kaum ein Service die dokumentenbasierte Variante benutzt.
Gründe dafür kann es viele geben. Einer wird aber sein, dass RPC sich einfacher implementieren lässt. Die Beschreibung eines solchen Dienstes in WSDL ist zwar umfangreicher da ja die Attribute mit den jeweiligen Datentypen versehen werden müssen, aber dadurch, dass SOAP die Serialisierung zumindest von einfachen und zusammengesetzten Datentypen übernimmt, spart man sich die Handhabung mit der XML Engine und somit das Parsen der Attribute in die jeweiligen Datentypen.

Werkzeuge

Wenn wir bei den Werkzeugen für CORBA und SOAP ausschließlich Java betrachten, so ist die Integration für CORBA bedingt durch den älteren Standard sehr gut. Schon seit den frühen Versionen von JBuilder [69] gab es eine Integration von CORBA. Seit der Version 5 auch für WebServices bzw. SOAP. In der aktuellen Version 9 von JBuilder Enterprise, die es als Trail Version zum Download gibt, lässt die Integration kaum Wünsche offen.
Auch andere Werkzeuge wie das freie Eclipse [70], ehemals von IBM entwickelt, unterstützt durch Plug-In's die Generierung von Stubs aus WSDL's. Ein anderes Plug-In unterstützt das Deployment auf axis.

Debugging

Wo beim debuggen von CORBA Schluss ist, fängt man bei SOAP erst an. Denn beim eigenen Kode in CORBA hört die Transparenz auf. Von da an obliegt alles dem ORB. Bei SOAP lässt sich zumindest beim Protokoll HTTP als Transportprotokoll alles schön über einen Proxy mitverfolgen. Dazu gibt es auch in axis ein Tool namens TCPMonitor. Mittels dieses Monitors ist es möglich die Anfragen bzw. die Antworten des Services im HTTP zu sehen.

5.3 Betrachtungen aus Sicht der Administratoren

Bei der Installation einer CORBA oder SOAP Implementation wird es wohl keine größeren Probleme geben. Anders sieht das bei der Sicherheit aus. SOAP lässt sich über einen Proxy betreiben und muss auch nicht getunnelt werden, da es ja HTTP ist. Den HTTP-Port 80 lassen die meisten Firewalls für Web-Server offen.
Dagegen gibt es Probleme bei CORBA. Der Standard CORBA-Port ist bei den meisten Firewalls nicht offen. Dies wird auch von vielen Firmen nicht gewünscht. Deshalb muss man CORBA tunneln. Dies geht nur über Produkte von Drittherstellern. Und selbst das läuft nicht ohne Probleme bei der Installation ab, wie sich in der Praxis von Firmen gezeigt hat.
Ansonsten zeigen sich beide Technologien unempfindlich gegenüber geänderten Ports.
Beide Technologien lassen sich auch hervorragend skalieren. Bei CORBA reicht es aus einen weiteren Rechner mit einem ORB zu installieren und den Server zu registrieren. Den Rest der Verteilung der Anfragen erledigt alles der ORB.
Bei SOAP reicht ebenfalls ein weiterer Rechner mit SOAP. Allerdings muss hier noch ein Load Balancer installiert sein, der die Anfragen unter Beachtung der Auslastung verteilt.

5.4 Performance

Ein weiteres Kriterium zur Entscheidung des Einsatzes von CORBA oder SOAP ist die Performance. Sind Anfragen zu langsam kann das ein Ausschlusskriterium sein. Ist ihre CPU-Last zu hoch können eventuell nicht genügend Nutzer gleichzeitig das Server bzw. Service nutzen.
Lasttests würden an dieser Stelle zu Weit führen, aber einfache Geschwindigkeitstests für CORBA und SOAP unter Berücksichtigung von Ressourcen wie CPU oder RAM werden nachfolgend beschrieben.

Die Testumgebung bestand aus zwei völlig identischen Computern für Client/Server bzw. Service:

CPU:	AMD Athlon 800 MHz
Speicher:	512 MB RAM
Festplatte:	SEAGATE ST318436LW UW-SCSI 18 GB
Netzwerk:	100 Mbps

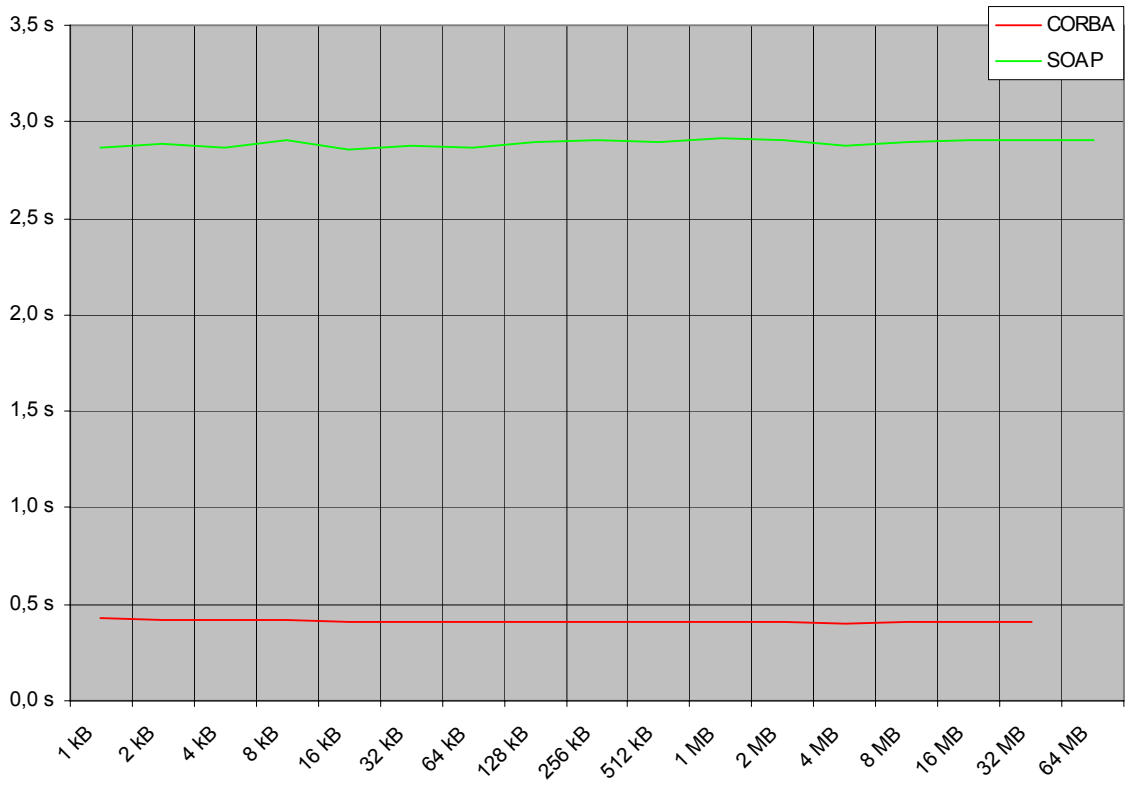

Abbildung 33: Performanceanalyse CORBA/SOAP (Preprocessing)

Der Test wurde in drei Abschnitte gegliedert: Preprocessing, Remote Call und Postprocessing. Unter Preprocessing fallen alle Schritte zu Vorbereitung der Anfrage. Im Falle von CORBA gehört das Kreieren und Initialisieren des ORB's, das Holen der Objektreferenz auf den Name Service sowie die Lokalisierung des Servers „Image" dazu.
Zum Preprocessing von SOAP gehören dazu: das Erzeugen eines Services, Calls sowie das Setzen der Parameter für den Aufruf wie URL, Operation Name, Login und Passwort. Die Abbildung 33: Performanceanalyse CORBA SOAP (Preprocessing) zeigt die benötigte Zeit in Abhängigkeit von der Größe des binären Datenpaketes, welches am Server angefragt werden soll.

Wie zu erwarten war, sind kein großen Schwankungen innerhalb der Verarbeitung von CORBA oder SOAP aufgetreten. Denn an den Anfragen zum CORBA-Server oder SOAP-Service ändert sich nicht wirklich etwas. Einen signifikanten Unterschied jedoch muss man zwischen CORBA und SOAP feststellen. SOAP ist knapp sechsmal langsamer als CORBA.
Das ist aber leicht zu erklären. Die Verarbeitung respektive die Erzeugung des SOAP XML Dokuments ist einfach zu zeitintensiv als das sie mit der Vorverarbeitung von CORBA mithalten könnte.
Den Großteil benötigt SOAP, um das Service respektive das Call Objekt zu instanziieren. Service und Call sind Objekte von Sun's Java API for XML-Remote Procedure Calls (JAX-PRC) die dazu benutzt werden Metadaten zu halten, die Notwendig sind um den Service aufrufen zu können.
Ist man hier bei einer zeitkritischen Anwendung angewiesen schneller zu sein, kann man sicher hier ansetzen.

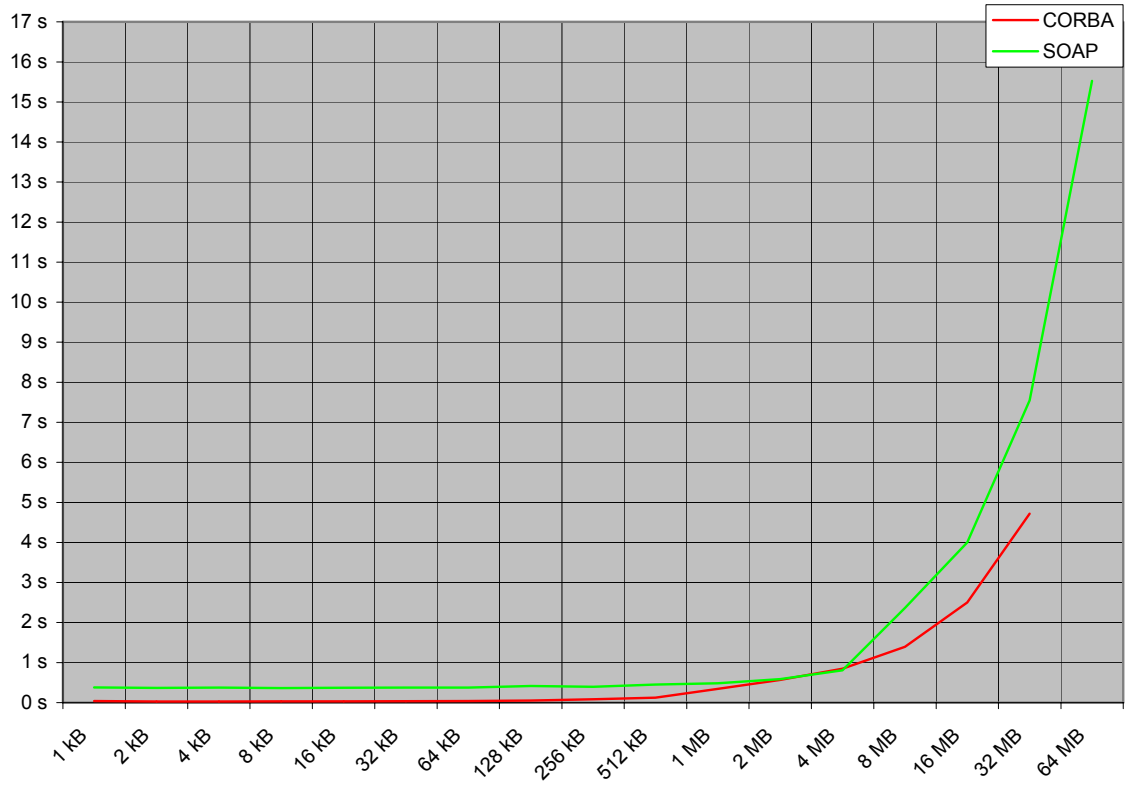

Abbildung 34: Performanceanalyse CORBA/SOAP (Remote Call)

Obige Abbildung 34: Performanceanalyse CORBA/SOAP (Remote Call) zeigt den eigentlichen Aufruf und somit die Übertragung der Daten. Hierbei ist zu sehen, dass die Transferprotokolle von CORBA und SOAP (hier HTTP) in etwa gleich gut sind.

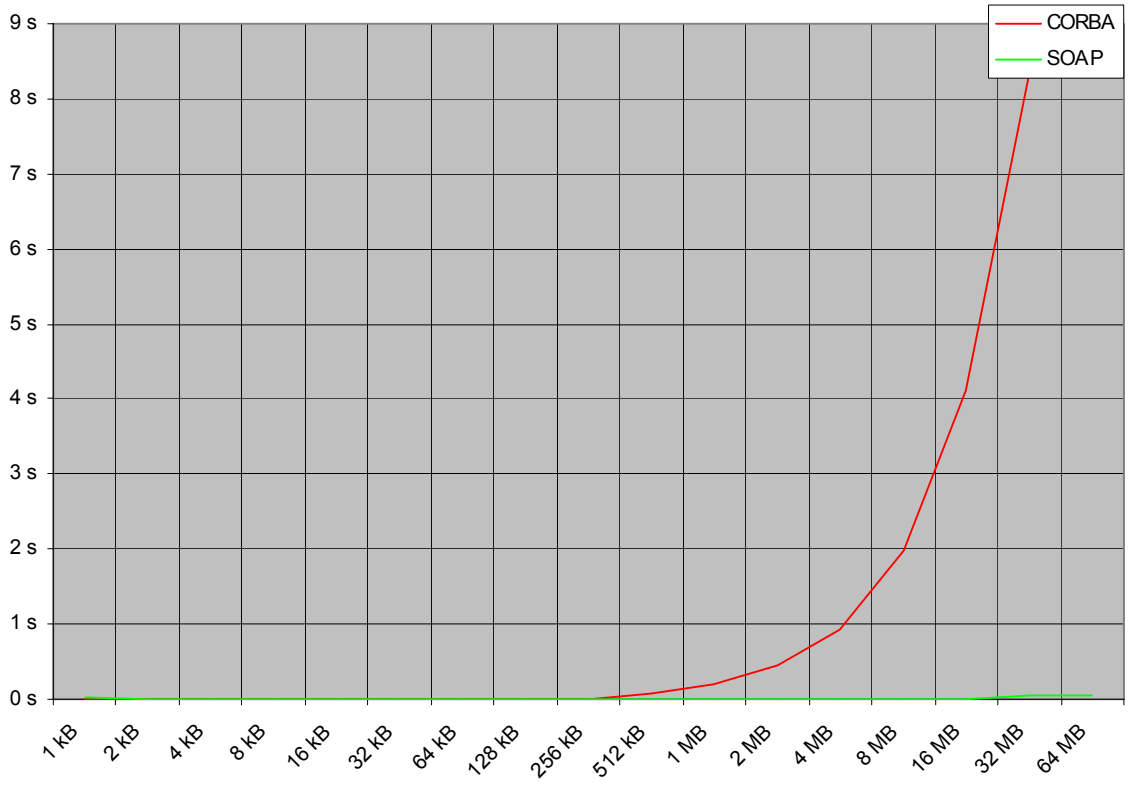

Abbildung 35: Performanceanalyse CORBA/SOAP (Postprocessing)

Beim Preprocessing zeigt sich ein leicht anderes Bild. Dies liegt darin begründet, dass axis seine Anhänge schon während des Übertragens speichert. Mittels CORBA Client erhält man jedoch ein so großes Byte Array wie Daten übertragen werden, was dann erst noch gespeichert werden muss. Das erklärt auch den Ausfall von CORBA bei der 64 MB Datei. Der RAM der VM des Servers hat wohl einfach nicht ausgereicht.

Die nachfolgende Abbildung 36: Performanceanalyse CORBA/SOAP (Overall) zeigt den Zeitverlauf über alle drei Phasen.

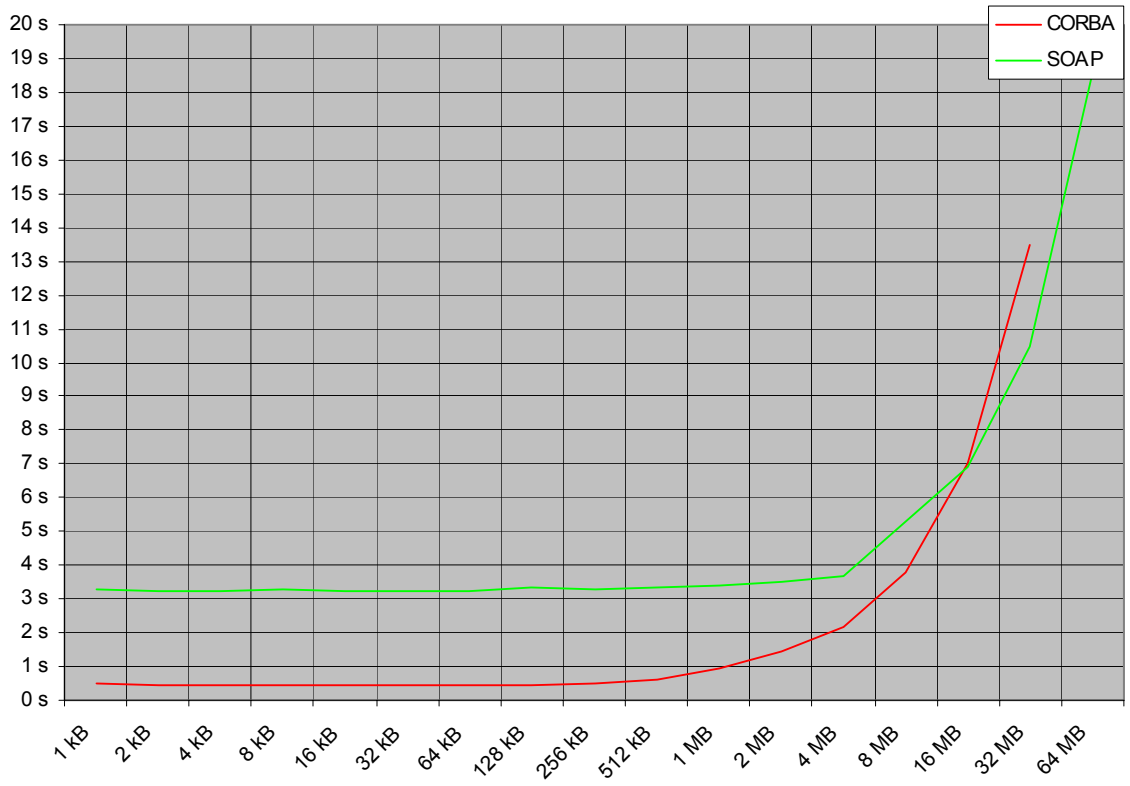

Abbildung 36: Performanceanalyse CORBA/SOAP (Overall)

5.5 Bewertung

Ist man auf Realtimeprozesse angewiesen, so sind wohl weder CORBA noch SOAP dafür geeignet. Die OMG hat zwar eine Spezifikation einer Realtime CORBA sie steht aber außerhalb der eigentlichen Spezifikation von CORBA. Auch in der Version 3 ist sie nicht vorgesehen. So fristet Realtime CORBA ein Nischendasein.

Für alle anderen Fälle eignen sich CORBA und SOAP gleich gut. Sieht man einmal davon ab, dass sich SOAP beim Erstellen der Anfrage gut sechs Mal soviel Zeit nimmt wie CORBA. Doch Optimierungspotential gibt es genug bei SOAP. Als erstes wäre die XML-Engine zu nennen. Diese bietet wohl das meiste Potential. Die Standard XML-Engine kann durch Crimson ersetzt werden bzw. man kann ganz auf eine Engine verzichten, wenn die Dokumente übersichtlich bleiben. Ein zweiter Ansatzpunkt leitet sich auch aus XML ab. Durch den Einsatz der dokumentenbasierten Form von SOAP kann man auch Zeit gewinnen.

Es mag paradox klingen, aber die Dokumente sind einfach schlanker, weil sie nicht die Beschreibung der Datentypen mit sich führen müssen. Somit ist weniger XML zu parsen und die Verarbeitung geht schneller.

Der finanzielle Aspekt geht zugunsten von SOAP aus. CORBA Middleware von kommerziellen Herstellern ist gegenüber SOAP sehr teuer. Zumal man sich auf einen bestimmten Hersteller festlegt, wenn man CORBA nutzt, weil es zwar einen Standard gibt, der aber nicht ausschließt, dass es Reibereien unter Produkten verschiedener Hersteller gibt.

Doch das schlagende Argument gegenüber CORBA ist die Fähigkeit von SOAP ohne Probleme durch Firewalls zu gelangen. CORBA tut sich da immer noch schwer. So wird oftmals zugunsten von SOAP auf CORBA verzichtet, weil es solche Probleme mit dem Tunneling von CORBA gibt.

6 Zusammenfassung

Der Austausch von größeren Binärdaten nicht nur im Umfeld der Medizin mittels der Komponententechnologien CORBA oder SOAP bietet mit beiden Verfahren Vor- und Nachteile. Wiegt man dieses Gegeneinander auf, zeigt sich ein leichter Vorteil zu Gunsten von SOAP. Die Argumente für SOAP sind eindeutig ihre einfachen Internettechnologien auf denen SOAP basiert. Dadurch sind sie leicht zu implementieren, egal auf welcher Plattform oder System- bzw. Programmierumgebung. SOAP ist dadurch auch eine kostengünstige Variante Daten in heterogener Umgebung auszutauschen.
CORBA andererseits ist bedingt durch den geringeren Overhead der Daten schneller in der Verarbeitung. Bei vielen Daten ist SOAP klar im Nachteil, da die zu versendenden Daten serialisiert werden müssen. Aus diesem Grund ist CORBA auch performanter als SOAP, was wiederum den Nachteil hat, dass das Protokoll nicht mehr lesbar ist.
Das komplexere Protokoll von CORBA ist der Grund, warum CORBA eine vergleichsweise kostenintensive Middleware ist. Bei der Sicherheit der Daten ist kein klarer Sieger auszumachen. Beide Komponententechnologien unterstützen eine verschlüsselte Übertragung. Und mit der Version 1.2 von SOAP wird auch ein Rollenkonzept mit Autorisierung und Authentifizierung eingeführt. Dieses ist bis dato nur nicht spezifiziert worden lies sich aber sehr gut selber abbilden.
Der größte Nachteil von CORBA ist jedoch ein auf den ersten Blick einfaches Problem. Die Datenpakete von CORBA werden von vielen Firewalls herausgefiltert und das tunneln durch HTTP ist nur schwer durch Software von Drittanbietern möglich. Für SOAP ist das natürlich keine Hürde, kann doch genau HTTP als Transportprotokoll benutzt werden.
Aber genau der Bereich, nämlich B2B Anwendungen über das Internet sind der boomende Markt aber wohl vorrangig für SOAP.

Zusammenfassend kann man sagen, dass für das Projekt Res Medicinae im Hinblick auf den Austausch von Daten über das Intranet oder Internet SOAP die bessere Wahl ist. Zudem gibt es für alle Komponenten freie Alternativen, was den Prinzipien von Open Source, wie der Software Res Medicinae, sehr entgegen kommt.

7 Ausblick

Nur weil SOAP mit aller Macht auf den Markt dringt, wird es nicht CORBA ersetzen. Vielmehr wird SOAP eine Alternative für die Integration von Geschäftsprozessen werden. Viele Firmen setzen heute schon auf SOAP als Technologie für diese Integration. So zum Beispiel SAP und Intershop. SAP ist dabei seine eigene Schnittstelle Open Catalog Interface (OCI) schrittweise auch als SOAP anzubieten. Diese durch SOAP angebotenen Kataloge können dann in die e-Commerce-Produkte von Intershop eingebunden werden. Derzeit unterstützen die Produkte von Intershop WebServices zwar nur proprietär, da diese Produkte aber auf Servlets aufbauen kann eine einfache Integration von SOAP jederzeit integriert werden. Ab dem dritten Quartal diesen Jahres wird es aber eine neue Version von Enfinity MultiSite, der Enterprise Plattform für e-Commerce von Intershop geben, die SOAP so integriert, dass es einfach per Mausklick möglich sein wird einen Service zu publizieren oder aufzurufen.

Dennoch wird der Markt für CORBA nicht zusammenbrechen. Vielmehr wird es auch auf lange Sicht eine Koexistenz zwischen SOAP und CORBA geben.

I. Akronymität

ACR	American College of Radiology
API	Application Program Interface
ASCII	American Standard Code for Information Interchange
ASF	Apache Software Foundation
ASL	Apache Software License
ASP	Active Server Pages
axis	Apache eXtensible Interaction System
B2B	Business to Business
B2C	Business to Constomer
BMP	BitMaP
BSI	Business System Integration
COMS	Cellular Management Operations System
CORBA	Common Object Request Broker Architecture
CPU	Central Processing Unit
CT	ComputerTomograpie
DCE	Distributed Computing Environment
DCOM	Distributed Component Object Model
DICOM	Digital Imaging and COmmunications in Medicine
DIME	Direct Internet Message Encapsulation
DIMSE	DICOM Message Service Element
DOM	Document Object Model
DTF	Domain Task Force
EAI	Enterprise Application Integration
EDI	Electronic Document Interchange
ESB	Enterprise Service Bus
FITS	Flexible Image Transport System
FTP	File Transfer Protocol
GIF	Graphic Image Format
GIOP	General Inter-ORB Protocol
GLGPL	GNU Library General Public Licence

GPL	(GNU) General Public License
GUI	Graphical User Interface
HTTP	HyperText Transfer Protocol
HTTPS	HyperText Transfer Protocol Secure
IANA	Internet Assigned Numbers Authority
IDL	Interface Definition Language
IIOP	Internet Inter-ORB Protocol
IIS	Internet Information Server
IMS	Interactive Multimedia Services
IOP	Inter-ORB Protocol
IOR	Interoperable Object Reference
J2EE	Java™ 2 Platform, Enterprise Edition
J2SE	Java™ 2 Platform, Standard Edition
JAX-RPC	Java API for XML-based RPC
JDK	Java™ Development Kit
JDK EE	Java™ Development Kit Enterprise Edition
JDK RE	Java™ Development Kit Runtime Environment
JDK SE	Java™ Development Kit Standard Edition
JNI	Java Native Interface
JPEG	Joint Picture Expert Group
JRE	Java™ Runtime Environment
JSP	JavaServer Pages
JVM	Java™ Virtual Machine
LAN	Local Area Network
MAC	Medium Access Control
MathML	Mathematical Markup Language
MIME	Multipurpose Internet Mail Extensions
MRT	MagnetResonanzTomographie
NEMA	National Electrical Manufacturers Association
NIH	National Institutes of Health
OCI	Open Catalog Interface
OMG	Object Management Group

ORB	Object Request Broker
OSF	Open Software Foundation
OSI	Open System Interconnection
PD	Public Domain
PHP	Hypertext Preprocessor
RAM	Random Access Memory
RFC	Request For Comments
RISC	Reduce Instruction Set Computer
RMI	Remote Method Invocation
RPC	Remote Procedure Calls
SAAJ	SOAP with Attachments API for Java
SAX	Simple API for XML
SFTP	Secure File Transfer Protocol
SGML	Standard Generalized Markup Language
SISC	Complex Instruction Set Computer
SMIL	Synchronized Multimedia Integration Language
SMTP	Simple Mail Transfer Protocol
SOA	Service-Oriented Architecture
SOAP	Simple Object Access Protocol
SSH	Secure SHell
SVG	Scalable Vector Graphics
TANGO	Traffic And Network Ground Operation
TCL	Tool Command Language
TCP/IP	Transmission Control Protocol/Internet Protocol
TIFF	Tag Image File Format
UDP	User Datagram Protocol
UML	Unified Modelling Language
URI	Uniform Resource Identifier
URL	Uniform Resource Locator
US-ASCII	United States-American Standard Code for Information Interchange
VM	Virtual Maschine
W3C	World Wide Web Consortium

Windows ME	Windows Millennium Edition
Windows NT	Windows New Technology
Windows XP	Windows eXPerience
WS	Web Services
WWW	World Wide Web
XML	eXtensible Markup Language
XML-RPC	eXtensible Markup Language-Remote Procedure Calls
XSD	XML Schema Declaration

II. Abbildungsverzeichnis

III. Quellenverzeichnis

[1] Christian Heller, „Res Medicinae – Introduction", http://resmedicinae.sourceforge.net/description/presentation/html/slide_3.html, 2002

[2] „ImageJ", http://rsb.info.nih.gov/ij/, ImageJ, 2003

[3] Christian Heller, „Res Medicinae", http://www.resmedicinae.org/, 2003

[4] Christian Heller, „Res Medicinae - Information in Medicine", http://resmedicinae.sourceforge.net/, 2003

[5] Jeremy Rosenberger, „CORBA in 14 Tagen", SAMS, Inprint der Markt&Technik Buch- und Software-Verlag GmbH, ISBN: 3-8272-2031-9, 1998

[6] Ciupke und Schmidt 1997 CIUPKE, Oliver ; SCHMIDT, Rainer: Components as Context-Independent Units of Software. In: MÜHLHÄUSER, Max (Hrsg.): Special Issues in Object-Oriented Programming. Heidelberg : dpunkt, 1997, S. 139–143

[7] „CORBA", http://www.corba.org/, Object Management, Group, 2003

[8] „White Pages", http://www.omg.org/homepages/index.htm, Object Management, Group, 2003

[9] „Finance Domain Task Force", http://fdtf.omg.org/, Object Management, Group, 2003

[10] „Telecommunications Domain Task Force", http://telecom.omg.org/, Object Management, Group, 2003

[11] „Healthcare Domain Task Force", http://healthcare.omg.org/, Object Management, Group, 2003

[12] „Transportation Domain Task Force", http://transport.omg.org/, Object Management, Group, 2003

[13] „Business Enterprise Integration Domain Task Force", http://bei.omg.org/, Object Management, Group, 2003

[14] Robert Orfali/Dan Harkley, „Client/Server Programming with Java and CORBA – Second Edition", Wiley Computer Publishing John Wiley & Sons, Inc., ISBN: 0-471-24578-X, 1998

[15] „Java Servlet Technology - The Power Behind the Server", http://java.sun.com/products/servlet/, Sun Microsystems, 2003

[16] „JavaServer Pages", http://java.sun.com/products/jsp/, Sun Microsystems, 2003

[17] „XML-RPC", http://www.xmlrpc.com/, UserLand Software, 1998

[18] „Digital Imaging and Communications in Medicine (DICOM)", http://medical.nema.org/, National Electrical Manufacturers Association, 2003

[19] „Digital Imaging and Communications in Medicine (DICOM) Part 7: Message Exchange", http://medical.nema.org/dicom/2003/03_07PU.PDF, National Electrical Manufacturers Association, 2003

[20] T. Berners-Lee MIT, LCS/R. Fielding, UC Irvine/H. Frystyk MIT, LCS, „RFC 1945: Hypertext Transfer Protocol -- HTTP/1.0", http://www.ietf.org/rfc/rfc1945.txt, RFC, 1996

[21] R. Fielding, UC Irvine/J. Gettys, Compaq, W3C/J. Mogul, Compaq/H. Frystyk, W3C, MIT/L. Masinter, Xerox/P. Leach, Microsoft/T. Berners-Lee, W3C, MIT, „RFC 2616: Hypertext Transfer Protocol -- HTTP/1.1", http://www.ietf.org/rfc/rfc2616.txt, RFC, 1999

[22] J. Franks, Northwestern University/P. Hallam-Baker, Verisign/J. Hostetler, AbiSource/S. Lawrence, Agranat Systems/P. Leach, Microsoft/A. Luotonen, Netscape Communications/L. Stewart, Open Market, „RFC 2617: HTTP Authentication: Basic and Digest Access Authentication", http://www.ietf.org/rfc/rfc2617.txt, RFC, 1999

[23] „HTTP/1.1 Specification Errata", http://purl.org/NET/http-errata, 2003

[24] „HTTP/1.1", http://www.apacheweek.com/features/http11, Apache Week, 1996

[25] „Extensible Markup Language (XML) – Introduction", http://www.w3.org/XML/#intro, W3C, 2003

[26] „Standard Generalized Markup Language (SGML) - SGML and XML as (Meta-) Markup Languages", http://xml.coverpages.org/sgml.html, Cover Pages, 2002

[27] Jason Levitt, „From EDI To XML And UDDI: A Brief History Of Web Services", http://www.informationweek.com/story/IWK20010928S0006, Information Week, 2001

[28] Don Box, „A Brief History of SOAP", http://webservices.xml.com/pub/a/ws/2001/04/04/soap.html, 2001

[29] „Web Services Activity", http://www.w3.org/2002/ws/, W3C, 2003

[30] „Web Services Architecture Working Group", http://www.w3.org/2002/ws/arch/, W3C, 2003

[31] „XML Protocol Working Group", http://www.w3.org/2000/xp/Group/, W3C, 2003

[32] „Web Services Description Working Group", http://www.w3.org/2002/ws/desc/, W3C, 2003

[33] „Web Services Choreography Working Group", http://www.w3.org/2002/ws/chor/, W3C, 2003

[34] Information Sciences Institute, University of Southern California, „RFC 793: Transmission Control Protocol", http://www.ietf.org/rfc/rfc793.txt, RFC, 1981

[35] Nilo Mitra, Ericsson, „SOAP Version 1.2 Part 0: Primer - The "role" Attribute", http://www.w3.org/TR/2003/PR-soap12-part0-20030507/#L6293, W3C, 2003

[36] „HTTP - Hypertext Transfer Protocol", http://www.w3.org/Protocols/, W3C

[37] Jonathan B. Postell, Information Sciences Institute, University of Southern California, „RFC 768: User Datagram Protocol", http://www.ietf.org/rfc/rfc768.txt, RFC, 1980

[38] Jonathan B. Postell, Information Sciences Institute, University of Southern California, „RFC 821: Simple Mail Transfer Protocol", http://www.ietf.org/rfc/rfc821.txt, RFC, 1982

[39] Jonathan B. Postell, J. Reynolds, Information Sciences Institute, University of Southern California, „RFC 959: File Transfer Protocol", http://www.ietf.org/rfc/rfc959.txt, RFC, 19895

[40] Chris Trytten, „XML im Detail - XML wird einen enormen Einfluss auf Ihre Datenbankentwicklung und -integration haben.", http://www.filemaker.de/xml/xml_examine.html, File Maker, 2003

[41] Tim Bray, Textuality and Netscape/Jean Paoli, Microsoft/C. M. Sperberg-McQueen, University of Illinois at Chicago and Text Encoding Initiative/Eve Maler, Sun Microsystems, Inc, „Extensible Markup Language (XML) 1.0 (Second Edition) - Origin and Goals", http://www.w3.org/TR/REC-xml#sec-origin-goals, W3C, 2000

[42] Dr. Guido Schryen, LuF Wirtschaftsinformatik, „Basistechnologien des Electronic Business", http://www.rwth-aachen.de/wi/lehre/lv/ws2002/EBBasistechnologien/BT03_Internetsprachen/doc/BT03 - Internetsprachen-825.htm, RWTH Aachen, 2003

[43] Martin Gudgin, Developmentor, „Schema for the SOAP/1.1 encoding", http://schemas.xmlsoap.org/soap/envelope, XML SOAP, 2001

[44] Martin Gudgin, Developmentor, „Schema for the SOAP/1.1 encoding", http://schemas.xmlsoap.org/soap/encoding, XML SOAP, 2001

[45] Don Box, DevelopMentor/David Ehnebuske, IBM/Gopal Kakivaya, Microsoft/Andrew Layman, Microsoft/Noah Mendelsohn, Lotus Development Corp./Henrik Frystyk Nielsen, Microsoft/Satish Thatte, Microsoft/Dave Winer, UserLand Software, Inc., „Simple Object Access Protocol (SOAP) 1.1 – SOAP Envelope", http://www.w3.org/TR/SOAP/#_Toc478383494, W3C, 2000

[46] Don Box, DevelopMentor/David Ehnebuske, IBM/Gopal Kakivaya, Microsoft/Andrew Layman, Microsoft/Noah Mendelsohn, Lotus Development Corp./Henrik Frystyk Nielsen, Microsoft/Satish Thatte, Microsoft/Dave Winer, UserLand Software, Inc., „Simple Object Access Protocol (SOAP) 1.1 – SOAP Header", http://www.w3.org/TR/SOAP/#_Toc478383497, W3C, 2000

[47] Don Box, DevelopMentor/David Ehnebuske, IBM/Gopal Kakivaya, Microsoft/Andrew Layman, Microsoft/Noah Mendelsohn, Lotus Development Corp./Henrik Frystyk Nielsen, Microsoft/Satish Thatte, Microsoft/Dave Winer, UserLand Software, Inc., „Simple Object Access Protocol (SOAP) 1.1 – SOAP Body", http://www.w3.org/TR/SOAP/#_Toc478383503, W3C, 2000

[48] Yasser Shohoud, „RPC/Literal and Freedom of Choice", http://msdn.microsoft.com/webservices/understanding/webservicebasics/default.aspx?pull=/library/en-us/dnwebsrv/html/rpc_literal.asp, Microsoft, 2003

[49] Don Box, DevelopMentor/David Ehnebuske, IBM/Gopal Kakivaya, Microsoft/Andrew Layman, Microsoft/Noah Mendelsohn, Lotus Development Corp./Henrik Frystyk Nielsen, Microsoft/Satish Thatte, Microsoft/Dave Winer, UserLand Software, Inc., „Simple Object Access Protocol (SOAP) 1.1 – SOAP Fault", http://www.w3.org/TR/SOAP/#_Toc478383507, W3C, 2000

[50] N. Borenstein, Bellcore/N. Freed, Innosoft „RFC 1341: MIME (Multipurpose Internet Mail Extensions) : Mechanisms for Specifying and Describing the Format of Internet Message Bodies", http://www.ietf.org/rfc/rfc1341.txt, RFC, 1992

[51] N. Freed, Innosoft/ N. Borenstein, „RFC 2045: Multipurpose Internet Mail Extensions (MIME) Part One: Format of Internet Message Bodies", http://www.ietf.org/rfc/rfc2045.txt, RFC, 1996

[52] N. Freed, Innosoft/ N. Borenstein, „RFC 2046: Multipurpose Internet Mail Extensions (MIME) Part Two: Media Types", http://www.ietf.org/rfc/rfc2046.txt, RFC, 1996

[53] E. Levinson, „RFC 2387: The MIME Multipart/Related Content-type", http://www.ietf.org/rfc/rfc2387.txt, RFC, 1998

[54] Jeannine Hall Gailey, „DIME; Sending Files, Attachments, and SOAP Messages Via Direct Internet Message Encapsulation", http://msdn.microsoft.com/msdnmag/issues/02/12/DIME/default.aspx, Microsoft Corp., 2002

[55] Matt Powell, „DIME: Sending Binary Data with Your SOAP Messages", http://msdn.microsoft.com/library/default.asp?url=/library/en-us/dnservice/html/service01152002.asp, Microsoft Corp. 2002

[56] John J. Barton, Hewlett Packard Labs/Satish Thatte, Microsoft/Henrik Frystyk Nielsen, Microsoft, „SOAP Messages with Attachments", http://www.w3.org/TR/SOAP-attachments, W3C, 2000

[57] Erik Christensen, Microsoft/Francisco Curbera, IBM Research/Greg Meredith, Microsoft/Sanjiva Weerawarana, IBM ResearchWeb, „Services Description Language (WSDL) 1.1", http://www.w3.org/TR/wsdl, W3C, 2001

[58 5] „IONA Orbix", http://www.orbix.com/, IONA Technologies, 2003

[59] „IONA Orbix", http://www.corba.org/vendors/pages/iona.html, Object Management, Group, 2001

[60] „Paragon Software OAK CORBA 2.0 ORB ", http://www.corba.org/vendors/pages/paragon.html, Object Management, Group, 2003

[61] „Vertel e*ORB™ 2.0 Mediation Software", http://www.corba.org/vendors/pages/verteleorb2.0.html, Object Management, Group, 2003

[62] „Olivetti OmniORB 2.6.1", http://www.corba.org/vendors/pages/olivetti.html, Object Management, Group, 2003

[63] „JacORB 1.4.1", http://www.jacorb.org/, JacORB, 2003

[64] „JacORB GNU Library General Public License", http://www.jacorb.org/lgpl.html, JacORB, 2003

[65] „Apache Software Foundation", http://www.apache.org/, Apache Software Foundation, 2003

[66] ImageJ Plugins, http://ij-plugins.sourceforge.net/, ImageJ, 2003

[67] NIH Image, http://rsb.info.nih.gov/nih-image/, NIH Image, 2003

[68] ImageJ Features, http://rsb.info.nih.gov/ij/features.html, ImageJ, 2003

[69] Borland, Inc., http://www.borland.com/, Borland, Inc., 2003

[70] Eclipse, http://www.eclipse.org/, Eclipse, 2003

IV. Thesen

- SOAP ist gegenüber CORBA eine einfache und kostengünstige Komponententechnologie.
- CORBA ist mit seinem Datenhandling performanter als SOAP, da keine Serialisierung in XML stattfindet.
- Das Erzeugen und Parsen von XML sind die bremsenden Faktoren bei der Verarbeitungsgeschwindigkeit von SOAP. Das Transportprotokoll HTTP von SOAP ist aber effizienter bei großen Datenmengen.
- SOAP wird CORBA nicht ersetzen können, insbesondere nicht auf dem Gebiet des verteilten Rechnens, dem Hauptanwendungsgebiet von CORBA.
- Durch die Verwendung verbreiteter Standards wie HTTP und XML ist SOAP wirklich plattformunabhängig. Anders als CORBA, wo es zwischen den einzelnen Implementationen von ORB's verschiedener Hersteller Probleme geben kann.
- CORBA wird sich hauptsächlich durch seine fehlende Unterstützung von Firewalls nicht gegen SOAP durchsetzen können.
- DICOM als Transportprotokoll für medizinische Daten wird nur für den Transport der Daten vom z.B. Tomographen zum Rechner Verwendung finden. Zur Rechner - Rechner Kommunikation wird es ersetzt werden durch andere Protokolle wie SOAP.
- Der weitaus größte Markt für WebServices bieten B2B Anwendungen. So wird es innerhalb kürzester Zeit eine Vielzahl von BSI-Lösungen auf Basis von SOAP geben.
- Bedingt durch die Integration von Geschäftsprozessen in heterogenem Umfeld werden WebServices rasch an Bedeutung gewinnen.
- WebServices bzw. SOAP ist eine einfache Methode Geschäftsprozesse zu integrieren (EAI) und ideal für das WWW.
- SOAP wird sich nicht so schnell auf dem B2C Sektor ausbreiten können, weil SOAP keine Standards für die Bezahlung der genutzten Services bietet.
- Open Source Software wie Apache HTTP-Server, Tomcat und axis eignen sich sehr gut als Ersatz für kommerzielle Produkte. Sie sind zum Teil, wie im Falle des Apache HTTP-Server, die Standard-Lösung mit einem Marktanteil weltweit von über 60%.

Ilmenau, den 09.07.2003

_____

Jens Kleinschmidt

V. Eidesstattliche Erklärung

Hiermit versichere ich, die vorliegende Diplomarbeit selbständig und nur unter Verwendung der angegebenen Hilfsmittel geschrieben zu haben.

Ilmenau, den 09.07.2003

_____

Jens Kleinschmidt

VI. Anhang

A. GNU Free Documentation License

GNU Free Documentation License

Version 1.2, November 2002

```
Copyright (C) 2000,2001,2002  Free Software Foundation, Inc.
59 Temple Place, Suite 330, Boston, MA  02111-1307  USA
Everyone is permitted to copy and distribute verbatim copies
of this license document, but changing it is not allowed.
```

0. PREAMBLE

The purpose of this License is to make a manual, textbook, or other functional and useful document "free" in the sense of freedom: to assure everyone the effective freedom to copy and redistribute it, with or without modifying it, either commercially or noncommercially. Secondarily, this License preserves for the author and publisher a way to get credit for their work, while not being considered responsible for modifications made by others.

This License is a kind of "copyleft", which means that derivative works of the document must themselves be free in the same sense. It complements the GNU General Public License, which is a copyleft license designed for free software.

We have designed this License in order to use it for manuals for free software, because free software needs free documentation: a free program should come with manuals providing the same freedoms that the software does. But this License is not limited to software manuals; it can be used for any textual work, regardless of subject matter or whether it is published as a printed book. We recommend this License principally for works whose purpose is instruction or reference.

1. APPLICABILITY AND DEFINITIONS

This License applies to any manual or other work, in any medium, that contains a notice placed by the copyright holder saying it can be distributed under the terms of this License. Such a notice grants a world-wide, royalty-free license, unlimited in duration, to use that work under the conditions stated herein. The "Document", below, refers to any such manual or work. Any member of the public is a licensee, and is addressed as "you". You accept the license if you copy, modify or distribute the work in a way requiring permission under copyright law.

A "Modified Version" of the Document means any work containing the Document or a portion of it, either copied verbatim, or with modifications and/or translated into another language.

A "Secondary Section" is a named appendix or a front-matter section of the Document that deals exclusively with the relationship of the publishers or authors of the Document to the Document's

overall subject (or to related matters) and contains nothing that could fall directly within that overall subject. (Thus, if the Document is in part a textbook of mathematics, a Secondary Section may not explain any mathematics.) The relationship could be a matter of historical connection with the subject or with related matters, or of legal, commercial, philosophical, ethical or political position regarding them.

The "Invariant Sections" are certain Secondary Sections whose titles are designated, as being those of Invariant Sections, in the notice that says that the Document is released under this License. If a section does not fit the above definition of Secondary then it is not allowed to be designated as Invariant. The Document may contain zero Invariant Sections. If the Document does not identify any Invariant Sections then there are none.

The "Cover Texts" are certain short passages of text that are listed, as Front-Cover Texts or Back-Cover Texts, in the notice that says that the Document is released under this License. A Front-Cover Text may be at most 5 words, and a Back-Cover Text may be at most 25 words.

A "Transparent" copy of the Document means a machine-readable copy, represented in a format whose specification is available to the general public, that is suitable for revising the document straightforwardly with generic text editors or (for images composed of pixels) generic paint programs or (for drawings) some widely available drawing editor, and that is suitable for input to text formatters or for automatic translation to a variety of formats suitable for input to text formatters. A copy made in an otherwise Transparent file format whose markup, or absence of markup, has been arranged to thwart or discourage subsequent modification by readers is not Transparent. An image format is not Transparent if used for any substantial amount of text. A copy that is not "Transparent" is called "Opaque".

Examples of suitable formats for Transparent copies include plain ASCII without markup, Texinfo input format, LaTeX input format, SGML or XML using a publicly available DTD, and standard-conforming simple HTML, PostScript or PDF designed for human modification. Examples of transparent image formats include PNG, XCF and JPG. Opaque formats include proprietary formats that can be read and edited only by proprietary word processors, SGML or XML for which the DTD and/or processing tools are not generally available, and the machine-generated HTML, PostScript or PDF produced by some word processors for output purposes only.

The "Title Page" means, for a printed book, the title page itself, plus such following pages as are needed to hold, legibly, the material this License requires to appear in the title page. For works in formats which do not have any title page as such, "Title Page" means the text near the most prominent appearance of the work's title, preceding the beginning of the body of the text.

A section "Entitled XYZ" means a named subunit of the Document whose title either is precisely XYZ or contains XYZ in parentheses following text that translates XYZ in another language. (Here XYZ stands for a specific section name mentioned below, such as "Acknowledgements", "Dedications", "Endorsements", or "History".) To "Preserve the Title" of such a section when you modify the Document means that it remains a section "Entitled XYZ" according to this definition.

The Document may include Warranty Disclaimers next to the notice which states that this License applies to the Document. These Warranty Disclaimers are considered to be included by reference in

this License, but only as regards disclaiming warranties: any other implication that these Warranty Disclaimers may have is void and has no effect on the meaning of this License.

2. VERBATIM COPYING

You may copy and distribute the Document in any medium, either commercially or noncommercially, provided that this License, the copyright notices, and the license notice saying this License applies to the Document are reproduced in all copies, and that you add no other conditions whatsoever to those of this License. You may not use technical measures to obstruct or control the reading or further copying of the copies you make or distribute. However, you may accept compensation in exchange for copies. If you distribute a large enough number of copies you must also follow the conditions in section 3.
You may also lend copies, under the same conditions stated above, and you may publicly display copies.

3. COPYING IN QUANTITY

If you publish printed copies (or copies in media that commonly have printed covers) of the Document, numbering more than 100, and the Document's license notice requires Cover Texts, you must enclose the copies in covers that carry, clearly and legibly, all these Cover Texts: Front-Cover Texts on the front cover, and Back-Cover Texts on the back cover. Both covers must also clearly and legibly identify you as the publisher of these copies. The front cover must present the full title with all words of the title equally prominent and visible. You may add other material on the covers in addition. Copying with changes limited to the covers, as long as they preserve the title of the Document and satisfy these conditions, can be treated as verbatim copying in other respects.
If the required texts for either cover are too voluminous to fit legibly, you should put the first ones listed (as many as fit reasonably) on the actual cover, and continue the rest onto adjacent pages.
If you publish or distribute Opaque copies of the Document numbering more than 100, you must either include a machine-readable Transparent copy along with each Opaque copy, or state in or with each Opaque copy a computer-network location from which the general network-using public has access to download using public-standard network protocols a complete Transparent copy of the Document, free of added material. If you use the latter option, you must take reasonably prudent steps, when you begin distribution of Opaque copies in quantity, to ensure that this Transparent copy will remain thus accessible at the stated location until at least one year after the last time you distribute an Opaque copy (directly or through your agents or retailers) of that edition to the public.
It is requested, but not required, that you contact the authors of the Document well before redistributing any large number of copies, to give them a chance to provide you with an updated version of the Document.

4. MODIFICATIONS

You may copy and distribute a Modified Version of the Document under the conditions of sections 2 and 3 above, provided that you release the Modified Version under precisely this License, with the Modified Version filling the role of the Document, thus licensing distribution and modification of the Modified Version to whoever possesses a copy of it. In addition, you must do these things in the Modified Version:

- **A.** Use in the Title Page (and on the covers, if any) a title distinct from that of the Document, and from those of previous versions (which should, if there were any, be listed in the History section of the Document). You may use the same title as a previous version if the original publisher of that version gives permission.
- **B.** List on the Title Page, as authors, one or more persons or entities responsible for authorship of the modifications in the Modified Version, together with at least five of the principal authors of the Document (all of its principal authors, if it has fewer than five), unless they release you from this requirement.
- **C.** State on the Title page the name of the publisher of the Modified Version, as the publisher.
- **D.** Preserve all the copyright notices of the Document.
- **E.** Add an appropriate copyright notice for your modifications adjacent to the other copyright notices.
- **F.** Include, immediately after the copyright notices, a license notice giving the public permission to use the Modified Version under the terms of this License, in the form shown in the Addendum below.
- **G.** Preserve in that license notice the full lists of Invariant Sections and required Cover Texts given in the Document's license notice.
- **H.** Include an unaltered copy of this License.
- **I.** Preserve the section Entitled "History", Preserve its Title, and add to it an item stating at least the title, year, new authors, and publisher of the Modified Version as given on the Title Page. If there is no section Entitled "History" in the Document, create one stating the title, year, authors, and publisher of the Document as given on its Title Page, then add an item describing the Modified Version as stated in the previous sentence.
- **J.** Preserve the network location, if any, given in the Document for public access to a Transparent copy of the Document, and likewise the network locations given in the Document for previous versions it was based on. These may be placed in the "History" section. You may omit a network location for a work that was published at least four years before the Document itself, or if the original publisher of the version it refers to gives permission.
- **K.** For any section Entitled "Acknowledgements" or "Dedications", Preserve the Title of the section, and preserve in the section all the substance and tone of each of the contributor acknowledgements and/or dedications given therein.
- **L.** Preserve all the Invariant Sections of the Document, unaltered in their text and in their titles. Section numbers or the equivalent are not considered part of the section titles.
- **M.** Delete any section Entitled "Endorsements". Such a section may not be included in the Modified Version.

- **N.** Do not retitle any existing section to be Entitled "Endorsements" or to conflict in title with any Invariant Section.
- **O.** Preserve any Warranty Disclaimers.

If the Modified Version includes new front-matter sections or appendices that qualify as Secondary Sections and contain no material copied from the Document, you may at your option designate some or all of these sections as invariant. To do this, add their titles to the list of Invariant Sections in the Modified Version's license notice. These titles must be distinct from any other section titles.
You may add a section Entitled "Endorsements", provided it contains nothing but endorsements of your Modified Version by various parties--for example, statements of peer review or that the text has been approved by an organization as the authoritative definition of a standard.
You may add a passage of up to five words as a Front-Cover Text, and a passage of up to 25 words as a Back-Cover Text, to the end of the list of Cover Texts in the Modified Version. Only one passage of Front-Cover Text and one of Back-Cover Text may be added by (or through arrangements made by) any one entity. If the Document already includes a cover text for the same cover, previously added by you or by arrangement made by the same entity you are acting on behalf of, you may not add another; but you may replace the old one, on explicit permission from the previous publisher that added the old one.
The author(s) and publisher(s) of the Document do not by this License give permission to use their names for publicity for or to assert or imply endorsement of any Modified Version.

5. COMBINING DOCUMENTS

You may combine the Document with other documents released under this License, under the terms defined in section 4 above for modified versions, provided that you include in the combination all of the Invariant Sections of all of the original documents, unmodified, and list them all as Invariant Sections of your combined work in its license notice, and that you preserve all their Warranty Disclaimers.
The combined work need only contain one copy of this License, and multiple identical Invariant Sections may be replaced with a single copy. If there are multiple Invariant Sections with the same name but different contents, make the title of each such section unique by adding at the end of it, in parentheses, the name of the original author or publisher of that section if known, or else a unique number. Make the same adjustment to the section titles in the list of Invariant Sections in the license notice of the combined work.
In the combination, you must combine any sections Entitled "History" in the various original documents, forming one section Entitled "History"; likewise combine any sections Entitled "Acknowledgements", and any sections Entitled "Dedications". You must delete all sections Entitled "Endorsements."

6. COLLECTIONS OF DOCUMENTS

You may make a collection consisting of the Document and other documents released under this License, and replace the individual copies of this License in the various documents with a single copy that is included in the collection, provided that you follow the rules of this License for verbatim copying of each of the documents in all other respects.
You may extract a single document from such a collection, and distribute it individually under this License, provided you insert a copy of this License into the extracted document, and follow this License in all other respects regarding verbatim copying of that document.

7. AGGREGATION WITH INDEPENDENT WORKS

A compilation of the Document or its derivatives with other separate and independent documents or works, in or on a volume of a storage or distribution medium, is called an "aggregate" if the copyright resulting from the compilation is not used to limit the legal rights of the compilation's users beyond what the individual works permit. When the Document is included in an aggregate, this License does not apply to the other works in the aggregate which are not themselves derivative works of the Document.
If the Cover Text requirement of section 3 is applicable to these copies of the Document, then if the Document is less than one half of the entire aggregate, the Document's Cover Texts may be placed on covers that bracket the Document within the aggregate, or the electronic equivalent of covers if the Document is in electronic form. Otherwise they must appear on printed covers that bracket the whole aggregate.

8. TRANSLATION

Translation is considered a kind of modification, so you may distribute translations of the Document under the terms of section 4. Replacing Invariant Sections with translations requires special permission from their copyright holders, but you may include translations of some or all Invariant Sections in addition to the original versions of these Invariant Sections. You may include a translation of this License, and all the license notices in the Document, and any Warranty Disclaimers, provided that you also include the original English version of this License and the original versions of those notices and disclaimers. In case of a disagreement between the translation and the original version of this License or a notice or disclaimer, the original version will prevail.
If a section in the Document is Entitled "Acknowledgements", "Dedications", or "History", the requirement (section 4) to Preserve its Title (section 1) will typically require changing the actual title.

9. TERMINATION

You may not copy, modify, sublicense, or distribute the Document except as expressly provided for under this License. Any other attempt to copy, modify, sublicense or distribute the Document is void, and will automatically terminate your rights under this License. However, parties who have received

copies, or rights, from you under this License will not have their licenses terminated so long as such parties remain in full compliance.

10. FUTURE REVISIONS OF THIS LICENSE

The Free Software Foundation may publish new, revised versions of the GNU Free Documentation License from time to time. Such new versions will be similar in spirit to the present version, but may differ in detail to address new problems or concerns. See http://www.gnu.org/copyleft/.
Each version of the License is given a distinguishing version number. If the Document specifies that a particular numbered version of this License "or any later version" applies to it, you have the option of following the terms and conditions either of that specified version or of any later version that has been published (not as a draft) by the Free Software Foundation. If the Document does not specify a version number of this License, you may choose any version ever published (not as a draft) by the Free Software Foundation.