universität wien

# DIPLOMARBEIT

Titel der Diplomarbeit

„Job-Management in einem Grid-System, realisiert als Web-Servlet und MIDlet“


Verfasser

Jürgen Weichselbaum


angestrebter akademischer Grad

Magister der Sozial- und Wirtschaftswissenschaften (Mag.rer.soc.oec.)

Wien, im November 2008

| | |
|---|---|
| Studienkennzahl lt. Studienblatt: | A175 |
| Studienrichtung lt. Studienblatt: | Wirtschaftsinformatik |
| Betreuer: | Ao. Univ.-Prof. Dr. Siegfried Benkner |
| | Mag. Gerhard Engelbrecht |

# Danksagung

Diese Seite möchte ich meiner Frau, meinen Kindern und meinen Eltern widmen, die mich bei der Anfertigung meiner Diplomarbeit unterstützt haben.

Besonders bedanken möchte ich mich bei Mag. Gerhard Engelbrecht für die tatkräftige Unterstützung bei der Erstellung meiner Diplomarbeit. Vielen Dank für die hilfreichen Anregungen, die Geduld und für die vielen Stunden des Korrekturlesens.

# Ehrenwörtliche Erklärung

Hiermit versichere ich, die vorliegende Diplomarbeit ohne Hilfe Dritter und nur mit den angegebenen Quellen und Hilfsmitteln angefertigt zu haben. Alle Stellen, die den Quellen entnommen wurden, sind als solche kenntlich gemacht worden. Diese Arbeit hat in gleicher oder ähnlicher Form noch keiner Prüfungsbehörde vorgelegen.

Wien, am _____

Datum, Unterschrift

# Inhaltsverzeichnis

1 Einleitung ..... 1
1.1 Problemstellung, Motivation und Zielsetzung ..... 2
1.2 Aufbau der Arbeit ..... 2
2 Grundlagen & Technologien ..... 3
2.1 Web Services ..... 4
2.1.1 Simple Object Access Protocol - SOAP ..... 5
Geschichtlicher Abriss ..... 6
Arbeitsweise von SOAP ..... 6
Aufbau von SOAP-Nachrichten ..... 7
Nachrichten-Inhalt in SOAP ..... 8
SOAP und Attachments ..... 10
MIME ..... 11
DIME ..... 12
MTOM ..... 12
2.1.2 Web Service Description Language - WSDL ..... 13
Aufbau eines WSDL-Dokuments ..... 14
2.1.3 UDDI – Universal Description, Discovery and Integration ..... 16
Datenstrukturen für UDDI ..... 17
Finden bzw. Zugriff auf ein Service ..... 18
2.2 Grid Computing ..... 19
2.2.1 Definition ..... 19
2.2.2 Geschichte ..... 22
2.2.3 Virtuelle Organisation ..... 22
2.2.4 Kategorisierung von Grids ..... 23
Kategorisierung auf Grund des Applikationstyp ..... 23

Kategorisierung auf Grund der Größe ..... 24
2.2.5 Architektur ..... 24
Fabric Schicht: Interfaces für die lokale Kontrolle ..... 25
Connectivity layer: Kommunikation und Sicherheit ..... 26
Resource layer: Nutzung einzelner Ressourcen ..... 27
Collective layer: Koordinierung mehrerer Ressourcen ..... 27
Application layer: ..... 28
2.2.6 OGSA ..... 28
OGSI ..... 30
WSRF (Web Services Resource Framework) ..... 31
2.3 Datenzugriff und Datenintegration ..... 32
2.3.1 Heterogene, verteilte Datenquellen ..... 32
2.3.2 OGSA-DAI Projekt ..... 33
2.3.3 OGSA-DAI Software ..... 33
Response und Perform-Dokumente ..... 34
OGSA-DAI Architektur ..... 35
OGSA-DAI Data Resources ..... 37
2.4 Heterogene Clients ..... 38
2.4.1 Java Portabilität ..... 38
2.4.2 J2ME - Java 2 Micro Edition ..... 39
2.4.3 CLDC - Connected Limited Device Configuration ..... 40
2.4.4 MIDP - Mobile Information Device Profile ..... 41
2.4.5 Erstellung einer MIDlet-Suite ..... 43
2.4.6 Zustände eines MIDlets ..... 44
3 VGE ..... 46
3.1 Architektur ..... 46
3.2 Service Zugriff Modell ..... 47
Registry Service ..... 47

Certificate Authorities (CAs) & PKI ..... 47
Mehrphasiger Zugriff ..... 48
3.3 Service Infrastruktur ..... 48
3.3.1 VGE Service Komponentenmodell ..... 48
3.3.2 Applikationsservices ..... 49
Application Execution Service-Komponente ..... 50
Data Transfer Service-Komponente ..... 51
QoS Management Service-Komponente ..... 51
Data Staging Service-Komponente ..... 51
3.3.3 Datenservices ..... 52
Query Execution Service-Komponente ..... 53
Data Transfer Service-Komponente ..... 53
3.4 Client Infrastruktur ..... 53
3.5 Provisioning ..... 56
Client & Service Komponenten ..... 56
Client-Service-Interaktion ..... 57
Service-Provisioning ..... 58
3.5.1 Individuelle Service-Konfiguration ..... 60
Application Service ..... 61
Data Service ..... 63
3.5.2 Automatischer Deployment-Prozess ..... 65
Applikationsservice ..... 66
Datenservice ..... 66
3.5.3 Client-Provisioning ..... 67
4 Erweiterung des VGE ..... 69
4.1 Ziele des Job Managers ..... 69
Erweiterungsszenarien ..... 69
4.2 Anforderungen ..... 71

4.3 Die Entititäten User und Job ... 72
4.3.1 User ... 72
4.3.2 Job ... 72
4.3.3 User-Job-Diagramm ... 72
4.4 Lösungsansatz und Architektur ... 73
4.4.1 Architektur ... 73
4.4.2 Job Monitoring Komponente ... 73
4.4.3 Job Manager Data Service ... 73
4.4.4 Job Manager Client ... 74
4.4.5 Mobile Job Manager Client ... 74
4.4.6 VGE-Service Architektur ... 74
4.4.7 VGE-Client Architektur ... 74
4.4.8 Konzept des Job Managers ... 74
4.5 Technische Anforderungen ... 75
4.6 Interaktionen ... 76
5 Implementierung ... 78
5.1 Job Manager Data Service ... 78
5.1.1 User ... 78
5.1.2 Job ... 78
5.1.3 Datenbankmodell ... 79
5.1.4 Data Resource - Realisierung mittels MySQL ... 80
5.2 Client-API ... 81
5.2.1 Low-level API Database Agent ... 82
5.2.2 High-level APIs JobManager und UserManager ... 82
5.2.3 Komponenten, APIs & Interfaces ... 83
5.3 Job Monitoring-Komponente ... 83
5.4 Job Manager Client ... 84
5.4.1 Job-Management ... 84

5.4.2 User-Administration ..... 85
5.5 Mobile Job Manager Client ..... 85
5.5.1 kSOAP ..... 86
5.6 Klassenübersicht bzw. –beschreibung ..... 86
5.6.1 Allgemeine Klassen & Interfaces ..... 86
5.6.2 Job Monitoring-Komponente ..... 87
5.6.3 Job Manager Client ..... 87
API für J2SE ..... 87
Web-Servlet „Job Manager Client“ ..... 88
5.6.4 Mobile Job Manager Client ..... 89
API für J2ME ..... 89
MIDlet „Mobile Job Manager Client“ ..... 90
5.7 Client-Entwicklung ..... 91
5.7.1 Job Manager Paket ..... 91
5.7.2 Client-Entwicklungsumgebung für den Job Manager Client ..... 92
5.7.3 MIDlet Mobile Job Manager Client ..... 93
5.8 Client-Provisioning ..... 94
5.8.1 JobManager-Property Datei ..... 94
5.8.2 Job Montioring-Komponente ..... 95
5.8.3 Web-Interface „Job Manager Client“ ..... 95
5.8.4 MIDlet „Mobile Job Manager Client“ ..... 95
5.9 Screenshots des Job Manager Clients & Mobile Job Manager Clients ..... 96
5.9.1 Screenshots des Job Manager Clients ..... 97
5.9.2 Screenshots des Mobile Job Manager Clients ..... 98
5.10 Mobile Job Manager Client-Tests ..... 100
5.10.1 Sun Java Wireless Toolkit ..... 100
5.10.2 LG U8630 ..... 100
5.10.3 TCPMonitor ..... 100

6 Prakt. Einsatzgebiete ..... 102
6.1 Anwendungsfälle / Einsatz in bestehenden Projekten ..... 102
6.2 Einsatz im Columbus Projekt ..... 102
7 Resümee ..... 103
Aufgabenstellung ..... 103
Verwendete Werkzeuge ..... 103
Vorgangsweise ..... 103
Erweiterung ..... 104
Fazit ..... 104
8 Samples/Listings: ..... 105
8.1 SOAP-Listings ..... 105
8.1.1 SOAP-/HTTP-Request und Response einer VGE-Client/Service-Kommunikation ..... 105
8.1.2 SOAP-Request mit Attachment ..... 107
8.1.3 SOAP-Response mit Attachment ..... 109
8.1.4 Ein Beispiel für MIME und DIME-Verfahren ..... 111
8.1.5 SOAP-Nachricht mittels MTOM ..... 111
8.2 WSDL (Teil einer VGE Service WSDL-Beschreibung) ..... 113
8.3 UDDI ..... 116
8.3.1 SOAP-Nachricht zum Speichern einer UDDI Business Entity ..... 116
8.4 Implementierungsunterschiede auf Grund von J2ME ..... 118
java.io-Package ..... 118
Table B.57. Interfaces of the java.io Package ..... 118
Table B.58. Classes of the java.io Package ..... 118
java.lang-Package ..... 119
Table B.68. Interfaces of the java.lang Package ..... 119
Table B.69. Classes of the java.lang Package ..... 119
java.util-Package ..... 120
Table B.91. Interfaces of the java.util Package ..... 120

Table B.92. Classes of the java.util Package ..... 120

9 Zusätzliche Listings ..... 121

9.1 SQL-Anweisungen zum Erzeugen der Datenbank ..... 121

9.2 JavaDoc ..... 122

9.2.1 JobManager für J2SE ..... 122

9.2.2 DatabaseAgent für J2SE ..... 127

9.2.3 JobManager für J2ME ..... 129

9.2.4 DatabaseAgent für J2ME ..... 131

10 Abkürzungsverzeichnis ..... 133

11 Literaturverzeichnis ..... 135

Abstrakt ..... 139

Lebenslauf ..... 140

# Abbildungsverzeichnis

Abbildung 1: SOA-Tempel ..... 3
Abbildung 2: Web Service basierte SOA (Service-Oriented Architecture) ..... 4
Abbildung 3: SOAP im TCP/IP Protokollstapel ..... 7
Abbildung 4: Aufbau einer SOAP-Nachricht ..... 8
Abbildung 5: Dime-Nachricht mit 3 Attachments (1 Attachment als Chunkdatensatz) ..... 12
Abbildung 6: Nachrichtentypen laut WSDL-Spezifikation ..... 14
Abbildung 7: Abstrakte und konkrete WSDL-Definitionen ..... 15
Abbildung 8: Datenstrukturen in UDDI ..... 17
Abbildung 9: Zusammenhang WSDL & UDDI ..... 18
Abbildung 10: Terminologie bzgl. Grid-Größe ..... 20
Abbildung 11: Eigenschaften eines Grid Systems ..... 21
Abbildung 12: Reale und virtuelle Organisationen ..... 22
Abbildung 13: Knowledge Grids, Semantic Grids, Semantic Web, Grids und Internet ..... 23
Abbildung 14: Grid Architektur ..... 24
Abbildung 15: APIs, SDKs & Protokolle ..... 28
Abbildung 16: Konzeptuelle Sicht einer Grid-Infrastruktur ..... 29
Abbildung 17: Grid-Services in OGSA & OGSI ..... 30
Abbildung 18: Die Entwicklung von Grid- und Webtechnologien konvergiert zu WSRF ..... 31
Abbildung 19: Perform und Response Dokumente ..... 34
Abbildung 20: OGSA-DAI Architektur ..... 36
Abbildung 21: Data Service Resource, Data Resource Accessor & Data Resource ..... 36
Abbildung 22: Java Plattform Versionen und ihr Einsatzgebiet ..... 39
Abbildung 23: J2ME Architektur ..... 41
Abbildung 24: Pakete der CLDC-Konfiguration und des MID-Profils ..... 41
Abbildung 25: API Selektion im Sun Java Wireless Toolkit (Screenshot) ..... 42
Abbildung 26: Klassenhierarchie für darstellbare Elemente des MIDP ..... 42
Abbildung 27: Entstehungsprozess einer MIDlet-Suite ..... 43
Abbildung 28: Lebenszyklus eines MIDlets ..... 45
Abbildung 29: VGE Architektur Überblick ..... 46
Abbildung 30: VGE-Grid mit Applikations und Daten Services ..... 47

Abbildung 31: VGE Application Service ..... 50
Abbildung 32: VGE Data Service ..... 52
Abbildung 33: Client-Komponenten ..... 54
Abbildung 34: Abstrakte Schichten des Client APIs ..... 55
Abbildung 35: Service- bzw. Client-Komponenten ..... 57
Abbildung 36: Services und Clients mit unterschiedlichen Komponenten/Funktionalitäten ..... 58
Abbildung 37: Provisioning Prozess ..... 60
Abbildung 38: Service-Konfiguration ..... 62
Abbildung 39: Hosting-Umgebung und zusätzliche Service-Komponenten ..... 62
Abbildung 40: Konfigurationsdaten der Datenressourcen ..... 64
Abbildung 41: Hosting-Umgebung und zusätzliche Service-Komponenten ..... 64
Abbildung 42: User-Job-Diagramm ..... 72
Abbildung 43: Job Manager Architektur ..... 73
Abbildung 44: Job Manager Konzept ..... 75
Abbildung 45: Interaktionen zwischen Job Manager Clients und VGE Entitäten ..... 76
Abbildung 46: Datenbankmodell ..... 79
Abbildung 47: Workflow der Nutzung eines Datenservices ..... 81
Abbildung 48: Agent- & Manager-Schicht ..... 82
Abbildung 49: Job Manager Komponenten / APIs / Interfaces ..... 83
Abbildung 51: Database Mapping der Tabellen jobs, users & administrate_user ..... 87
Abbildung 50: Job & User spezifische Ableitung ..... 87
Abbildung 52: JobManagerGridEventListener der Job Monitoring-Komponente ..... 87
Abbildung 53: API Klassen für J2SE ..... 88
Abbildung 54: Web-Servlet Klassen ..... 89
Abbildung 55: Klassenübersicht des Mobile Job Manager APIs ..... 90
Abbildung 56: Klassenübersicht des Mobile Job Manager Interfaces ..... 91
Abbildung 57: Job Manager Paket ..... 92
Abbildung 58: kSOAP Paket ..... 92
Abbildung 59: Jobs des eingeloggten Benutzers ..... 97
Abbildung 60: Jobs aller zur Administration berechtigten Benutzer ..... 97
Abbildung 61: Benutzer-Verwaltung ..... 98
Abbildung 62: Benutzer-Profil ..... 98
Abbildung 63: Login-Screen des Mobile Job Manager Clients ..... 99
Abbildung 64: Einstellungen des Mobile Job Manager Clients ..... 99
Abbildung 65: Laufende Jobs eines Benutzers ..... 99

Abbildung 66: Job-Details ..... 99
Abbildung 67: LG U8630 mit Mobile Job Manager Client ..... 100
Abbildung 68: Funktionsweise des TCPMonitors ..... 101
Abbildung 69: Kommunikation zwischen MIDlet und Job Manager Data Service im TCPMonitor ..... 101

## Listings

Listing 1 Struktur einer SOAP-Nachricht ..... 8
Listing 2: sqlQuery und sqlResult ..... 35
Listing 3: Applikationsdeskriptor des „Mobile Job Manager Clients“ ..... 43
Listing 4: MIDlet Quellcode ..... 45
Listing 5: VGE Client Code Beispiel für die Nutzung eines Applikationsservices ..... 56
Listing 6: Service-Deskriptor für ein Applikationsservice ..... 62
Listing 7: Service Properties ..... 63
Listing 8: Data Service Resource-Datei ..... 65
Listing 9: Web-Client Properities-Datei ..... 68
Listing 10: Beispiel-Code für die Nutzung des User- & JobManager APIs ..... 82
Listing 11: SOAP-/HTTP-Request und Response einer VGE-Client/Service-Kommunikation ..... 106
Listing 12: SOAP-Request mit Attachment ..... 108
Listing 13: SOAP-Response mit Attachment ..... 111
Listing 14: SOAP-Nachricht mit dem MIME-Verfahren ..... 111
Listing 15: SOAP-Nachricht mit dem DIME-Verfahren ..... 111
Listing 16: SOAP-Nachricht mit und ohne XOP-Serialisierung ..... 113
Listing 17: Teil einer VGE Service WSDL-Beschreibung ..... 115
Listing 18: SOAP-Nachricht zum Speichern einer UDDI Business Entity ..... 116
Listing 19: Lesen aller UDDI Business Einträge mit den Namen „XMethods“ & Ergebnis ..... 117
Listing 20: Lesen eines bestimmten UDDI Service Eintrages & Ergebnis ..... 117

# 1 Einleitung

Das Vienna Grid Environment (VGE) ist ein Service-orientiertes Software-System, um eine Grid Umgebung zu realisieren, welche sowohl HPC-basierte (High Performance Computing) Anwendungen als auch Datenressourcen als Services zur Verfügung stellen kann. VGE basiert auf Web Service Standards (z.B. SOAP, WSDL) und wurde am Institut für Scientific Computing der Universität Wien entwickelt.

Das Ziel dieser Arbeit ist die Planung, Realisierung und Evaluierung von funktionalen Erweiterungen der VGE Umgebung.

Ein Schwerpunkt der Arbeit ist das Hinzufügen eines Client Job Managers um orts- und client-unabhängig laufende und abgeschlossene Jobs eines eindeutig identifizierbaren Benutzers zu verwalten. Dieser ermöglicht

- die Statusabfrage von auf verschiedenen Clients gestarteten Jobs,
- die Abfrage von Ergebnissen eines beendeten Jobs und
- den Abbruch laufender Jobs.

Der Job Manager soll als zusätzliche Client-Komponente realisiert werden und kann optional mit bestehenden Clients verwendet werden.

Zur Speicherung der Daten verwendet der Job Manager ein Data Service, welches eine Archivierung bzw. Auswertung diverser Nutzungsdaten ermöglicht. Außerdem kann dem Benutzer dadurch eine chronologische Auflistung der benutzten Services für eine spätere Wiederverwendung angezeigt werden.

Die Umsetzung dieser zusätzlichen Client-Funktionalitäten bzw. –Komponenten sollen im Kontext realer Applikationen, wie z.B. Columbus[1] getestet werden. Eine Evaluierung mit mehreren Services und Clients (lokaler und externer) der implementierten Funktionserweiterungen bildet den Abschluss des ersten Teiles.

Ein weiterer Teil der Arbeit soll die Benutzerfreundlichkeit des VGE-Systems erhöhen und knüpft an den zuvor beschriebenen Job Manager an. Dazu soll ein „Mobile Job Manager“ für mobile Endgeräte realisiert werden. Diese Software für mobile Clients soll aus Sicherheitsgründen nur Statusabfragen und den Abbruch laufender Jobs ermöglichen.

[1] Columbus ist eine Sammlung von Programmen zur ab initio Berechnung der molekularen Elektronenstruktur.

## 1.1 Problemstellung, Motivation und Zielsetzung

Wenn ein Job von einem Client gestartet wird, so ist der Job an diesen Client gebunden und in einem dezentralen System wie dem VGE – abgesehen von dem Server, wo er ausgeführt wird - unbekannt. Eine Identifizierung des Jobs bzw. dessen Statusabfrage kann nur von dem Client durchgeführt werden, durch den die Initialisierung erfolgte. Anderen Clients ist der Job unbekannt.

Nur dieser Client kann die durch ihn initiierten Jobs identifizieren und Statusabfragen durchführen.

Weiters ist ein System-weiter Überblick über die Nutzung der VGE Services nicht möglich. Eine Archivierung der Job-Daten (Client-Adresse, aufgerufenes Service, Upload-, Start- & Downloadzeit) würde für die Administration des VGE einen zusätzlichen Vorteil bringen.

Durch den Job Manager soll

1. die Gebundenheit des Jobs an den Client aufgelöst und dessen Administration (z.B. Statusabfragen) von anderen Clients ermöglicht werden,
2. ein Überblick über die Nutzung der Clients und Services geschaffen werden und
3. eine Archivierung der Job-Daten erfolgen.

Der „Mobile Job Manager“ könnte auch als der „Personal Job Manager“ bezeichnet werden. Dieser ermöglicht die Statusabfrage, der von einem Benutzer gestarteten Jobs, mit Hilfe eines mobilen Endgerätes, wie einem PDA (Personal Digital Assistant) oder einem Mobiltelefon, das Java Applikationen ausführen kann.

## 1.2 Aufbau der Arbeit

Diese Diplomarbeit besteht aus einem theoretischen und einem praktischen Teil: Im theoretischen Teil der Arbeit werden im Kapitel 2 grundlegende Web Service-Technologien (SOAP, WSDL, UDDI, …) sowie Grid-Technologien (OGSA-DAI) im Kontext von Web und Grid Services behandelt, die im VGE verwendet werden. Das VGE, dessen Komponenten und Architektur werden im Kapitel 3 beschrieben. Im Kapitel 4 wird der Job Manager als Erweiterung des VGE-Systems erklärt.

Der praktische Teil der Arbeit startet mit Kapitel 5, welches aus einer Implementierungsbeschreibung des Job Managers, dem Datenbankmodell und einer Klassenbeschreibung besteht. Im Kapitel 6 werden die Ergebnisse vom Einsatz des Job Managers bei bestehenden Projekten erläutert. Kapitel 7 bildet den Abschluss der Diplomarbeit und beinhaltet ein Resümee.

# 2 Grundlagen & Technologien

Im E-Business ist es immer wichtiger, unternehmensweite bzw. unternehmensübergreifende Anwendungen anzubieten. SOA ist die am häufigsten verwendete Architektur, um sowohl die Infrastruktur für unternehmensübergreifende Anwendungen (B2B – Business-to-Business), als auch Lösungen zur Integration interner Anwendungen (Enterprise Application Integration) zu realisieren. SOA ermöglicht es, weltweit verfügbare Dienste anzubieten, die entweder direkt vom Endanwender genutzt werden oder als Baustein in eine neue Anwendung integriert werden können.

In **[Dos05]** wird SOA wie folgt definiert: „Unter einer Service-orientierten Architektur (SOA) versteht man eine Systemarchitektur, die vielfältige, verschiedene und eventuell inkompatible Methoden oder Applikationen als wiederverwendbare und offen zugreifbare Dienste repräsentiert und dadurch eine plattform- und sprachenunabhängige Nutzung und Wiederverwendung ermöglicht.“

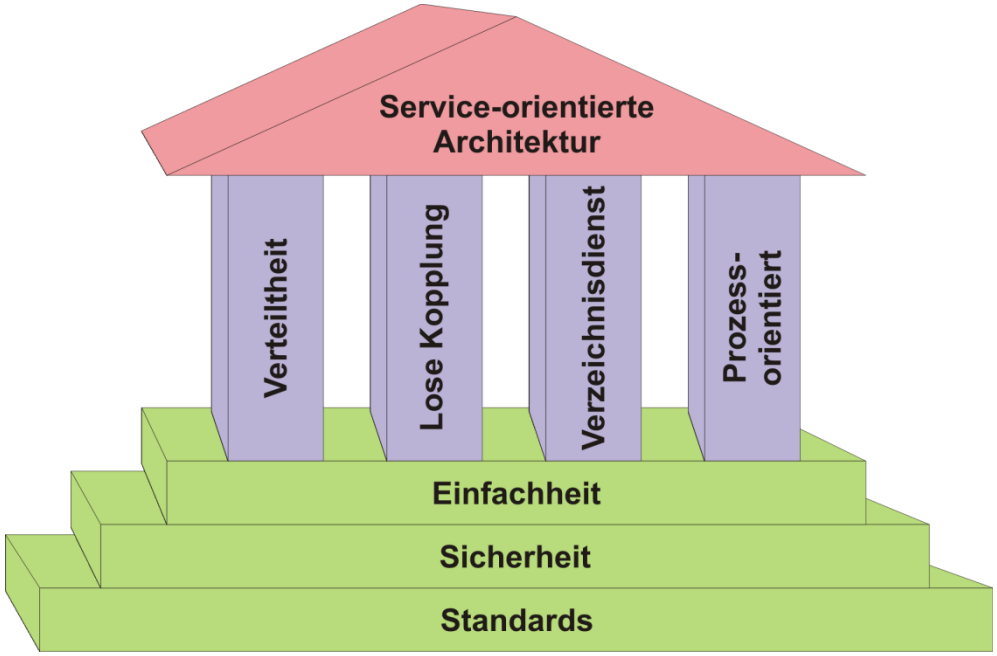



**Abbildung 1: SOA-Tempel. Quelle** [Dos05]

Abbildung 1 illustriert das Konzept einer Service-orientierten Architektur. Die Basis bilden offene Standards, Sicherheit und die die Einfachheit der angebotenen Services. Verteilte Dienste, deren lose Kopplung (*loose coupling*) und ihre prozessorientierte Struktur sind wesentliche Merkmale einer SOA. Anwendungen oder Dienste suchen autonom, zur Erfüllung ihrer Aufgabe, nach Diensten, welche die entsprechende Leistung dafür anbieten und den Präferenzen der Benutzer entsprechen. Dazu verwenden sie Verzeichnisdienste (*Repositories*), die ebenfalls eine tragende Säule im SOA-Tempel darstellen.

## 2.1 Web Services

Web Services sind eine konkrete technische Realisierung des Konzeptes einer Service-orientierten Architektur, die das Bereitstellen und automatisierte Nutzen von zur Verfügung gestellten Web Diensten ermöglicht. Die grundlegenden Komponenten einer SOA - die Kommunikation, die Dienstbeschreibung und das Benutzen von Verzeichnisdiensten - sind durch folgende Standards realisiert:

- SOAP dient zur Kommunikation mit Web Services, siehe Kapitel 2.1.1
- Web Service Description Language (WSDL) wird zur Beschreibung der Dienste eines Web Services verwendet, siehe Kapitel 2.1.2
- Universal Discovery Description and Integration (UDDI) beschreibt einen Verzeichnisdienst für Web Services, siehe Kapitel 2.1.3

Das W3C[2] definiert ein Web Service so: „A Web service is a software system designed to support interoperable machine-to-machine interaction over a network. It has an interface described in a machine-processable format (specifically WSDL). Other systems interact with the Web service in a manner prescribed by its description using SOAP messages, typically conveyed using HTTP with an XML serialization in conjunction with other Web-related standards.“ **[Haa04]**

Durch das Standardwebprotokoll SOAP wird Webbenutzern der Zugriff auf Web Services ermöglicht. Wie die Interaktion mit den Web Services erfolgen soll, ist durch dieses Protokoll beschrieben; in der Regel werden mittels einem WSDL-Dokument die Schnittstellen definiert und ermöglichen somit den Clientanwendungen die Kommunikation mit den Services. Web Services werden mittels UDDI registriert und gefunden.

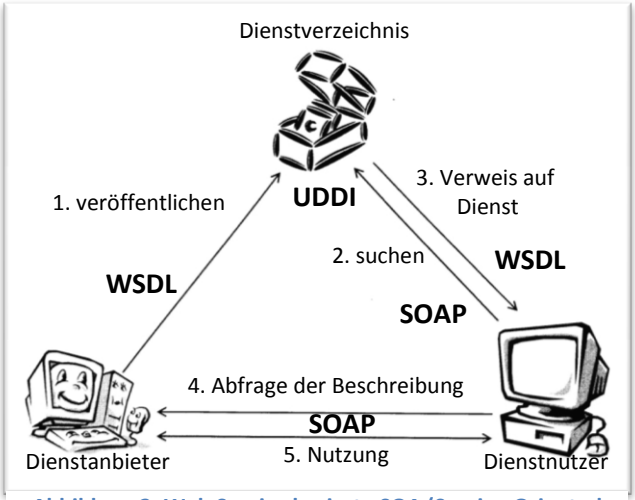



**Abbildung 2: Web Service basierte SOA (Service-Oriented Architecture). Quelle** [Dos05]

Dieses „publish-find-bind“-Prinzip ist in Abbildung 2 dargestellt und zeigt das Zusammenspiel bzw. den Einsatz von WSDL, UDDI und SOAP.

[2] Das World Wide Web Consortium (W3C) ist ein internationales Konsortium und entwickelt Web Standards. Zu den Mitgliedern zählen ein Team von Mitarbeitern, zahlreiche Organisationen und Unternehmen.

Web Services lassen sich durch folgende Punkte charakterisieren:

- Web Services basieren auf offenen Protokollen wie SOAP und WSDL.
- Web Services bzw. der Zugriff darauf ist durch deren Schnittstellenbeschreibung in WSDL selbsterklärend.
- Web Services können mit der UDDI Technologie registriert bzw. gefunden werden.
- Web Services können von anderen Web Services oder Applikationen benutzt werden. Eine Applikation kann somit eine Zusammensetzung aus mehreren Web Services sein.
- Web Services bzw. die verwendeten Protokolle basieren auf XML[3].

Ein wesentlicher Vorteil von XML Web Services besteht darin, dass sie sprachenunabhängig sind. Programme, entwickelt in unterschiedlichen Programmiersprachen, auf verschiedenen Plattformen können mittels standardisierter Protokolle miteinander kommunizieren. Ein weiterer Vorteil ist, dass diese mit den Standardwebprotokollen XML, HTTP[4] und TCP/IP[5] zusammenarbeiten.

Ein Großteil der Unternehmen besitzt eine Webinfrastruktur, in der für die benötigten Protokolle Standardprotokolle eingesetzt werden. Somit sind auch die notwendigen Voraussetzungen und Möglichkeiten gegeben, um Web Services sowohl als Lösung für unternehmensübergreifende Applikationen, als auch zur Integration interner Anwendungen zu verwenden.

Das Anbieten eines Web Services bzw. das zur Verfügung stellen, setzt ein entsprechendes Deployment voraus, d.h. ein Web Service wird in einem Web Service Container (i.A. Web Server) mit einem entsprechenden Namen gebunden und registriert. Nach dem Deployment ist das Web Service erreichbar und kann benutzt werden.

### 2.1.1 Simple Object Access Protocol - SOAP

SOAP ist ein auf XML basierendes Kommunikationsprotokoll, welches unabhängig vom Transportprotokoll die genaue Interaktion zwischen zwei Netzwerkentitäten definiert. Die wesentlichen Aspekte von SOAP sind:

- SOAP ist Plattform und Programmiersprachen unabhängig.
- SOAP basiert auf XML und ist somit erweiterbar.
- SOAP wird als W3C Standard entwickelt.

[3] XML - Extensible Markup Language
[4] HTTP - Hypertext Transfer Protocol
[5] TCP/IP - Transmission Control Protocol/Internet Protocol

Die SOAP-Spezifikation beinhaltet **[Cau01]** :

- eine Syntax zur Definition von Nachrichten als XML Dokument
- ein Model zum Austausch von SOAP Nachrichten
- ein Regelwerk zur Datendarstellung in SOAP Nachrichten
- eine Richtlinie für SOAP Nachrichten, die als Transportmedium HTTP benutzen
- eine Konvention, um mit SOAP Nachrichten Remote Procedure Calls durchführen zu können

### *Geschichtlicher Abriss*

Dave Winer entwickelte 1998 gemeinsam mit Microsoft die Spezifikation für XML-RPCs **[Win99]** . Als Weiterentwicklung entstand die erste Version von SOAP – Version 0.9, welche Ende 1999 veröffentlicht wurde. Im selben Jahr wurde sogleich noch die Version 1.0 veröffentlicht, was dazu führte, dass sich zahlreiche Firmen (DevelopMentor, IBM, Microsoft, Lotus Development Corp., Microsoft, UserLand Software - Dave Winer) der Entwicklung anschlossen. Die daraus entstandene Version 1.1 **[Box00]** liegt als W3C-Note vor. Die Nachfolgeversion – Version 1.2 **[Mit07]** – wurde vom W3C im Juni 2003 als Empfehlung (W3C Recommendation) veröffentlicht bzw. die überarbeitete Version 1.2 Second Edition im Juni 2007.

### *Arbeitsweise von SOAP*

SOAP ist ein Protokoll auf XML-Basis. Es setzt auf verschiedene Standards auf: auf Übertragungsprotokolle der Anwendungsschicht (wie HTTP, HTTPS[6], SMTP[7], FTP[8], … -.siehe Abbildung 3) und der Transportschicht (wie TCP). XML wird zur Repräsentation verwendet. Die häufigste eingesetzte Variante ist SOAP über HTTP und TCP – HTTP ist auch das einzige Protokoll, das in der SOAP-Spezifikation standardisiert ist.

Zwei wesentliche Beweggründe für die Entwicklung bzw. Verwendung von SOAP:

- HTML (Hypertext Markup Language ) ermöglicht keinen Datenaustausch oder RPCs. Durch SOAP wird dies ermöglicht.
- In den meisten Netzwerken sind Proxy-Server[9] und Firewalls vorhanden, welche die Kommunikation, wie sie z.B. in CORBA[10] oder JavaRMI[11] realisiert ist, verhindern. SOAP wird

[6] HTTPS - HyperText Transfer Protocol Secure
[7] SMTP - Simple Mail Transfer Protocol
[8] FTP - File Transfer Protocol
[9] Proxy-Server (kurz Proxies) ermöglichen Systemen, die keinen direkten Zugang zum Internet haben, den indirekten Zugriff.
[10] CORBA (Common Object Request Broker Architecture) ist eine Architektur für die Entwicklung verteilter Applikationen

vorwiegend über HTTP benutzt. HTTP ist ein Standard-Protokoll und wird von allen Internet Browsern und Servern verstanden. Auf Firewalls und Proxies ist das HTTP-Protokoll bzw. der gängige Port 80 bereits voreingestellt. Somit stellt die Nutzung von SOAP via HTTP kein Problem dar.

| | |
|---|---|
| Anwendungsschicht | **SOAP** |
| | HTTP, HTTPS, FTP, SMTP, POP, Telnet… |
| Transportschicht | TCP, UDP |
| Internetschicht | IPv4, IPv6, ICMP |
| Netzwerkschicht | Ethernet, Token Bus, Token Ring, FDDI, … |

**Abbildung 3: SOAP im TCP/IP Protokollstapel. Überarbeitet aus** [Cau01]

SOAP ist eine Spezifikation, die ein XML-Format für Nachrichten definiert und bestimmt die Regeln für das Nachrichtendesign. Es beschreibt, wie die Abbildung der Daten in der Nachricht zu erfolgen hat bzw. wie die Daten von der empfangenden Seite interpretiert werden müssen. SOAP beinhaltet keine Regeln zur Semantik applikationsspezifischer Daten, sondern bietet nur die Möglichkeit, beliebige applikationsspezifische Daten zu übertragen. In der Spezifikation ist ebenfalls festgelegt, wie Remoteprozeduraufrufe erfolgen müssen und ermöglicht dadurch die Entwicklung von verteilten, auf RPC-basierten Anwendungen. Der Client sendet den Namen der aufrufbaren Funktion und eventuelle Parameter in einem XML-Dokument an das Web Service; das Service schickt eine Nachricht mit den Ergebnissen des Funktionsaufrufes zurück.

Die SOAP-Nachricht wird auf der Senderseite erstellt, validiert und dann gesendet. Auf der Empfängerseite wird die Nachricht ausgewertet und verarbeitet (= parsen). Dazu wird ein Parser (z.B. SAX[12], DOM[13], …) und ein entsprechendes WS-Framework benutzt.

### *Aufbau von SOAP-Nachrichten*

Eine SOAP-Nachricht ist ein XML-Dokument mit einem standardisierten Aufbau mit folgenden Elementen:

- Envelope: Der *Envelope* identifiziert die XML-Nachricht als SOAP-Nachricht. Der *Envelope* umschließt die ganze SOAP-Nachricht und fängt mit der XML Deklaration an. Er besteht aus einem Element mit dem Schlüsselwort „*Envelope*“, dem ein Name zugewiesen werden muss. Außerdem muss dieses Element mittels eines Namensraumattributes auf ein XML-Schema für SOAP-Nachrichten (z.B. http://www.w3.org/2003/05/soap-envelope) referenzieren. Weitere Namensräume, die in der Nachricht benötigt werden, können hier festgelegt werden. Außerdem

[11] JavaRMI (Java Remote Method Invocation) ist die Funktionalität zur Durchführung von RPCs in Java
[12] SAX - Simple API for XML
[13] DOM - Document Object Model

besteht die Möglichkeit, Serialisierungsregeln für die SOAP Nachricht mit dem *encodingStyle* Attribut anzugeben. Der *Envelope* beinhaltet als Kind-Element einen *Body* und optional einen *Header* bzw. ein *Fault*-Element.

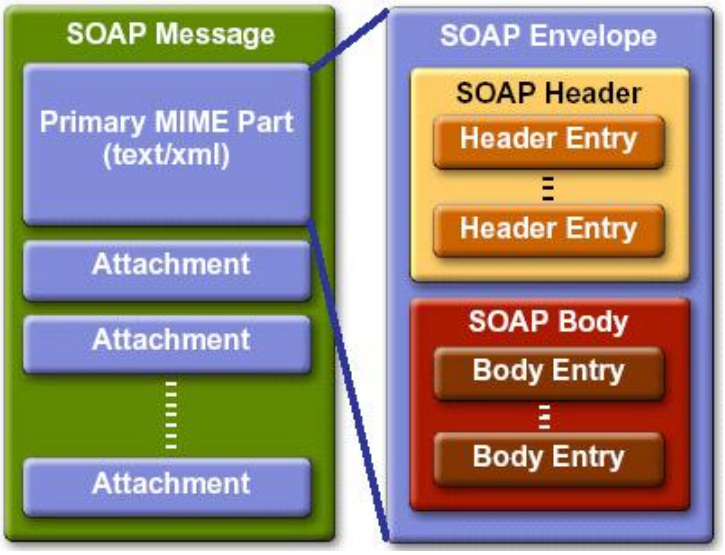



Abbildung 4: Aufbau einer SOAP-Nachricht. Quelle [Shi08]

```
<?xml version="1.0"?>
<soap:Envelope
    xmlns:soap="http://www.w3.org/
    2003/05/soap-envelope">

  <soap:Header>
  ...
  </soap:Header>

  <soap:Body>
    ...
    <soap:Fault>
    ...
    </soap:Fault>
  </soap:Body>

</soap:Envelope>
```

Listing 1 Struktur einer SOAP-Nachricht

- Header (optional): Im Header können Metainformationen fürs Transaktionshandling (Authentifizierung, Autorisierung), für die Verschlüsselung oder Routinginformationen angegeben werden.
- Body: Der Body beinhaltet die eigentliche Nachricht, d.h. die Request bzw. Response-Daten. Diese können sowohl Informationen zur Datenübertragung, als auch Anweisungen für RPCs sein. Ist der Inhalt nun ein RPC, so enthält er den aufzurufenden Methodennamen und eventuelle Parameter.
- Fault-Element (optional): Es beinhaltet Informationen über Fehler auf Grund einer nicht erfolgreichen Verarbeitung der SOAP-Nachricht.

Ein Beispiel für SOAP-Nachrichten findet man im Anhang – siehe „8.1.1 SOAP-/HTTP-Request und Response einer VGE-Client/Service-Kommunikation“.

### *Nachrichten-Inhalt in SOAP*

Im SOAP Standard gibt es mehrere Nachrichtenstile (*Styles*), die das Aussehen des Nachrichteninhaltes spezifizieren. Ein Nachrichtenstil beschreibt den Aufbau des Bodies einer SOAP Nachricht und die Repräsentation der zu übertragenden Daten.

Wie nun Clients und Services miteinander kommunizieren müssen, wird in der WSDL Beschreibung des Services bzw. dem Encoding[14] der Nachricht definiert. Die WSDL Spezifikation sieht für den Funktionsaufruf mittels SOAP (siehe Kapitel „2.1.2 Web Service Description Language - WSDL“) zwei Attribute vor: die Attribute *style* und *use*.

Attribut *style*: entweder *document* oder *rpc*; legt den Aufbau des SOAP Bodies fest

*rpc*: Der Nachrichteninhalt entspricht dem Request/Response Prinzip und beinhaltet Methodennamen und Parameter. Der SOAP Body beinhaltet als direktes Kind ein Element mit den Namen der aufzurufenden Methode, welches potentielle Parameter als weitere Kind-Elemente enthält. Ein entsprechender Response ist analog zur Request Nachricht.

*document*: *Document* wird für die Nachrichten-orientierte Kommunikation verwendet und ist die allgemeine Variante für den Nachrichtenaufbau. Es können beliebige XML Dokumente ausgetauscht werden. Der *document* Style kann jedoch genauso wie der *rpc* Style für RPCs verwendet werden.

Attribut *use*: entweder *encoded* oder *literal*; legt das Encoding fest

*encoded*: Encoding erfolgt auf Grund der Spezifikation im SOAP-Protokoll.

*literal*: Enconding erfolgt durch XML Schema Typen.

Theoretisch ergeben sich vier mögliche Nachrichtenstile, wobei nur drei verbreitet sind **[Coh03]** :

- SOAP RPC encoding (*rpc/encoded*): auch als „Section 5 encoding“ bekannt; definiert durch die SOAP 1.1 Spezifikation
- SOAP RPC Literal encoding (*rpc/literal*)
- SOAP document-style encoding: ist auch als message-style oder *document-literal* Encoding bekannt

[14] Unter Encoding versteht man die Angabe von Kodierungsregeln, die festlegen, wie die Darstellung von Methodenaufrufen und (deren) Daten einer Nachricht im XML Format erfolgen muss.

| | **encoded** | **literal** |
|---|---|---|
| **rpc** | + Standard in AXIS[15]/Java<br>+ weit verbreitet<br>+ unterstützt von .NET<br>+ SOAP-Body enthält Methodennamen<br>– nicht WS-I[16] verträglich | + unterstützt von AXIS<br>+ unterstützt von .NET<br>+ SOAP-Body enthält Methodennamen<br>+ WS-I verträglich<br>– wird sehr selten genutzt |
| **document** | Ø diese Kombination wird nicht genutzt<br>– nicht WS-I verträglich | + Standard in .NET/Microsoft<br>+ unterstützt von AXIS/Java<br>+ Empfehlung von WS-I<br>– Methodenname fehlt im SOAP-Body<br>– Probleme mit rekursiven Datenstrukturen (z.B. Graphen) |

**Tabelle 1: Hauptmerkmale der vier möglichen Binding Styles rpc/encoded, rpc/literal, document/encoded und document/literal. Quelle** [Prä06]

### *SOAP und Attachments*

SOAP basiert, wie bereits erwähnt, auf XML und daher sind SOAP Nachrichten reine Text-Nachrichten. Um Binärdaten (z.B. eine Grafik, ein PDF-Dokument) mit SOAP zu übertragen, gibt es zwei Möglichkeiten:

1. Die Daten werden transformiert bzw. kodiert und in den Body Teil des XML Dokumentes als echter Bestandteil aufgenommen. Dies wird auch als *inline* bezeichnet.
2. Die Daten werden nicht transformiert und in der ursprünglichen Form dem XML-Dokument als Anhang (Attachment) angehängt. Dies ist auch für Daten im Textformat möglich, da die Daten für die eigentliche Nachricht keine Bedeutung haben und so zur besseren Strukturierung des Nachrichteninhaltes als Anhang hinzugefügt werden.

Die erste Variante ist nur für kleine Datenmengen sinnvoll. Im W3C XML-Schema ist die Base64 Kodierung **[Jos06]** zur Transformation von 8-Bit-Binärdaten in 64 Zeichen des ASCII-Zeichensatzes[17] für XML-Dokumente spezifiziert. Diese Umwandlung von binären Daten hat zur Folge, dass das Datenvolumen im Gegensatz zur ursprünglichen Größe um ein Drittel ansteigt. Ein weiterer Nachteil von inline kodierten Binärdaten ist, dass diese Daten vom XML-Parser schon beim Lesen der Nachricht verarbeitet werden müssen. Dies führt dazu, dass mehr Speicherbedarf aufgrund des

[15] Axis ist ein Web Service Framework.
[16] Die Web Services Interoperability Organization (WS-I) ist ein Industrie-Konsortium, das Richtlinien bzgl. Web Service-Interoperabilität erstellt. **[WSI]**
[17] ASCII - American Standard Code for Information Interchange

größeren XML-Dokumentes notwendig ist und es zu einer längeren Rechenzeit durch das XML-Parsen, abhängig vom konkret verwendeten XML-Parser, kommt.

Werden die Daten als Attachment übertragen, so kommen derzeit folgende Standards zum Einsatz:

- MIME (Multipurpose Internet Mail Extension)
- DIME (Direct Internet Message Encapsulation)
- MTOM (Message Transmission Optimization Mechanism)

In der Nachricht kommt zuerst immer die eigentliche SOAP-Nachricht – der SOAP Envelope (vgl. Abbildung 4: Aufbau einer SOAP-Nachricht). Von diesem SOAP Envelope werden dann auf ein oder mehrere, danach folgende Attachments verwiesen. Die Referenzierungsart ist, abhängig vom Verfahren, unterschiedlich; in der Regel wird jedoch mittels eines Schlüsselwerts (z.B. mit dem *href*-Attribut bekannt aus HTML) von dem Envelope auf das Attachment referenziert.

### MIME

MIME steht für Multipurpose Internet Mail Extensions. Wie der Name schon sagt, kommt das Standardformat MIME von der Email Verwendung und spezifiziert das Versenden von Emails mit Attachments. MIME legt fest, wie ein so genanntes Multipart-Mail alle zu übertragenden Attachments in einer Datei übermittelt und wie die Trennung der einzelnen Teile voneinander erfolgt. Im Header der MIME Nachricht wird eine Seperator Zeichenkette festgelegt, welche zur Trennung der Nachricht in die einzelnen Teile bzw. Attachments dient. Dieser Seperator steht am Beginn jedes Teiles, gefolgt von zusätzlichen Metainformationen bzgl. Datenformat, Kodierung, etc. des jeweiligen Attachments. Der Typ der übermittelten Daten wird durch das *Content-Type*-Feld angegeben, die für die Übertragung gewählte Zeichenkodierung durch das *Content-Transfer-Encoding*-Feld.

Da dieses Verfahren nicht nur für Emails sinnvoll ist, wird es auch für die Übertragung von Daten bzw. Dateien im Web verwendet.

MIME wird in den RFCs (Requests for Comments) 2045, 2046, 2077, usw. behandelt:

- RFC2045: Multipurpose Internet Mail Extensions (MIME) Part One: Format of Internet Message Bodies **[Fre96]**
- RFC2046: Multipurpose Internet Mail Extensions (MIME) Part Two: Media Types **[Fre961]**
- RFC2077: The Model Primary Content Type for Multipurpose Internet Mail Extensions **[Nel97]**

Kapitel „8.1.2 SOAP-Request mit Attachment“ und „8.1.3 SOAP-Response mit Attachment“ beinhalten zwei Beispiele – eine Request und eine Response Nachricht, jeweils mit einem Attachment.

### DIME

DIME (Direct Internet Message Encapsulation) wurde von Microsoft und IBM entwickelt und ist ein ähnliches Format, um Nachrichten mit Attachments zu verschicken. Wie bei MIME kommt vor jedem Attachment ein Header, der eine Id beinhaltet und die als Referenz in der SOAP Nachricht angegeben wird (*href* Attribut wie bei MIME). Bei MIME heißt diese Id *Content-Id*, bei DIME ist es der *Identifier Header*.

Der wesentliche Unterschied zu MIME besteht darin, dass der Header bei DIME binär kodiert ist und die Länge des Attachmentblockes beinhaltet. Somit ist kein Seperator mehr notwendig– im Anhang befindet sich dazu ein Beispiel – „8.1.4 Ein Beispiel für MIME und DIME-Verfahren“.

Im ersten Datensatz wird das MB-Flag (*Message Begin*-Flag) und im letzten Datensatz das ME-Flag (*Message End*-Flag) gesetzt. Ein Attachment kann bei DIME auf mehrere Datensätze aufgeteilt werden; man spricht dann von einem Chunkdatensatz.

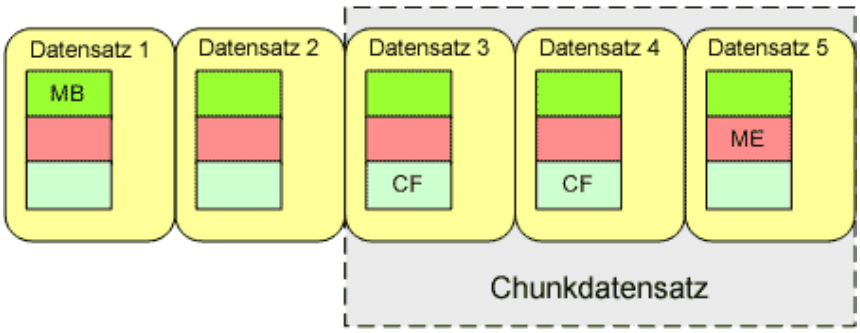



**Abbildung 5: Dime-Nachricht mit 3 Attachments (1 Attachment als Chunkdatensatz). Quelle** [Pow03]

MIME Nachrichten bieten für den Sender bzw. Ersteller der Nachricht den Vorteil, dass er die Nachrichtenlänge nicht im Voraus kennen muss (dies wurde bei DIME durch das CF-Flag – Chunk Flag – gelöst). Auf der Empfängerseite ist dies wiederum ein Nachteil, da jede Zeile der empfangenen Nachricht mit dem Seperator verglichen werden muss und ein Parsen der Nachricht gegenüber DIME länger dauert. Bei DIME muss zwar auf der Senderseite die Attachment- bzw. Datensatzlänge im vornherein bestimmt werden, ermöglicht auf der Empfängerseite jedoch ein schnelleres Lokalisieren des gewünschten Attachments.

### MTOM

MTOM (Message Transmission Optimization Mechanism Standard) Nachrichten sind im Wesentlichen wie Nachrichten mit den zuvor beschriebenen Attachment-Verfahren aufgebaut. Anstelle des *href* Attributes wird mit dem „*xop:Include*“ Element auf das Attachment verwiesen.

Der wesentliche Unterschied zu den vorigen Verfahren besteht in der Betrachtung der Architektur:

- Bei MIME und DIME wird auf das Attachment referenziert; es wird als separater Teil der Nachricht betrachtet. Eine Bearbeitung des Attachments erfolgt unabhängig von der eigentlichen SOAP-Nachricht.
- Bei MTOM wird das Attachment als logischer Bestandteil des XML-Dokumentes betrachtet - es ist aus konzeptueller Sicht einem *inline* Attachment gleichzusetzen. Die Namensgebung verdeutlicht dies mit der Bezeichnung „*include*“, da es nur ausgelagert ist. Tools sollen/müssen eine Bearbeitung des XML-Dokuments inklusive des Attachments ermöglichen.

Der zugrunde liegende Serialisierungsmechanismus ist das XOP-Verfahren (XML-Binary Optimized Packaging) und wird in einer eigenen Spezifikation **[Gud05]** definiert, da XOP auch in anderen Bereichen als SOAP zur Anwendung kommt. Ein XOP-Dokument wird erzeugt, indem Base64 kodierte Teile aus dem Dokument extrahiert und in ihrer ursprünglichen Form als Attachment angehängt werden.

Im Anhang „8.1.5 SOAP-Nachricht mittels MTOM“ ist ein Beispiel zur Übertragung der Nachricht bzw. des Attachments in unterschiedlichen Formaten - einmal mit *inline*-Kodierung und einmal mit MTOM.

### 2.1.2 Web Service Description Language - WSDL

SOAP ermöglicht in einer standardisierten Art und Weise, Nachrichten und deren Daten mit Web Services auszutauschen. SOAP stellt jedoch nur eine Definition für den Nachrichten- und Datenaustausch dar, bietet aber keine Möglichkeit zur Beschreibung, wie das Nachrichtenformat auszusehen hat. Um ein Web Service zu nutzen, muss man nun aber wissen, wie die Nachrichten aufgebaut sein müssen, um mit diesem erfolgreich zu kommunizieren. WSDL bietet die Möglichkeit, die Schnittstelle des Web Services zu beschreiben. Ein WSDL-Dokument ist ein XML-Dokument, das die Operationen und dessen Ein-/Ausgabeparameter definiert. WSDL ist ein W3C Standard; Version 1.1 **[Chr01]** liegt seit 2001 als Vorschlag vor, Version 2.0 **[Boo07]** wurde im Jahr 2007 als Empfehlung veröffentlicht.

Mit WSDL wird festgelegt, wie die Darstellungsform einer Anforderungsnachricht für ein Web Service bzw. die resultierende Antwortnachricht eindeutig auszusehen haben. Außerdem wird definiert, wo der Dienst verfügbar ist und mit welchem Kommunikationsprotokoll darauf zugegriffen werden kann. Somit sind für eine Clientanwendung alle notwendigen Charakteristika zur Benützung des Web Services gegeben.

Durch WSDL ist eine Beschreibung der Schnittstellen zu einem Web-Service möglich und mit Hilfe von SOAP kann dann unabhängig von Plattformen und Programmiersprachen darauf zugegriffen werden.

In WSDL wird vorzugsweise das XML-Schema als Datentypsystem zur Spezifizierung des Nachrichteninhaltes verwendet, es können aber auch alternative Datenmodelle benutzt werden.

### *Aufbau eines WSDL-Dokuments*

Ein WSDL-Dokument beinhaltet folgende Elemente:

- **Types:** *Types* ist notwendig, wenn komplexe Datentypen beschrieben werden müssen. Werden zusätzlich zu den Datentypen vom XML Schema noch eigene komplexe Datentypen benötigt, so können diese mit XML Schema Definitionen formuliert werden.
- **Message:** Hier werden die Nachrichten bzw. deren Nachrichteninhalt definiert, die bei den einzelnen Operationen ausgetauscht werden. Alle Message-Elemente innerhalb eines WSDL-Dokumentes müssen einen eindeutigen Namen haben. Im Normalfall werden für jede Operation zwei Nachrichten definiert – eine Request und eine Response Nachricht. Hat eine Nachricht Parameter, so muss für jeden Parameter ein *parts*-Element mit Namen und Typ (Datentyp vom XML Schema oder zuvor unter *types* definierter komplexer Datentyp) angegeben werden. Alle *parts*-Namen einer Message müssen eindeutig sein (analog den *Message*-Namen).
- **portType:** In dem *portType*-Element (meist ist nur ein *portType*-Element vorhanden) werden die Operationen des Services beschrieben. Für jede Methode werden hier die zuvor definierten Nachrichten im *Message*-Element mittels dem *operation*-Element zusammengefasst, wobei die Nachrichten vom Typ „*input*“, „*output*“ oder „*fault*“ sein können. Die Namen der *portType*-Elemente, sofern mehrere vorhanden sind, müssen eindeutig sein. Abbildung 6 zeigt die möglichen Nachrichtentypen, die sich in Abfolge und Anzahl der Nachrichten unterscheiden.

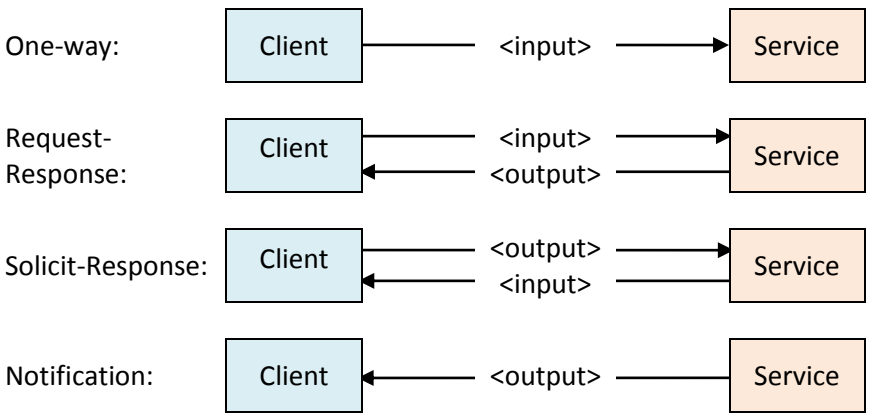



**Abbildung 6: Nachrichtentypen laut WSDL-Spezifikation. Quellen** [Kus02] **,** [Cau01]

- **binding:** Das *binding*-Element gibt die konkreten Nachrichtenformate und spezifischen Details an, wie die *portType*-Operationen mit dem jeweiligen Protokoll übertragen werden müssen. Mit dem *type*-Attribut wird auf einen *portType* verwiesen. Der Aufbau ist analog zum *portType*-Element. Im WSDL-Standard werden konkrete Bindings für SOAP, HTTP GET/POST, MIME definiert, wobei die häufigste Form von Bindings bei Web Services die Anbindung an das SOAP-Protokoll ist. Erfolgt eine Anbindung an das SOAP-Protokoll, gibt es zwei Attribute namens „*transport*“ und „*style*“. Transport definiert, welches Transportprotokoll für die Übertragung verwendet wird. Im Falle von HTTP hat dieses Attribut den Wert „http://schemas.xmlsoap.org/soap/http“. Das Attribut style kann den Wert „*rpc*“ oder „*document*“ annehmen.
- **service & port:** Mit diesen Elementen werden die Endpunkte des Services angegeben. In einem WSDL-Dokument können ein oder mehrere *service*-Elemente angegeben werden, die jeweils ein oder mehrere *port*-Elemente beinhalten. In den *port*-Elementen sind dann die Netzwerkadressen definiert. Ein *port*-Element hat die Attribute „*name*“ und „*binding*“, das *binding*-Attribut verweist auf das *binding*-Element.
- **documentation & import:** Dies sind 2 optionale Elemente.
  - **documentation:** *Documentation* kann in jedem anderem WSDL-Element verwendet werden und dient für Erklärungen bzw. für Kommentare.
  - **import:** Dadurch können andere WSDL-Dokumente und XML-Schema Definitionen inkludiert werden; ein modularer Aufbau wird dadurch ermöglicht.

Das Diagramm in Abbildung 7 stellt nochmals dar, in welcher Beziehung die einzelnen Dokumentabschnitte zueinander stehen. Die Pfeile geben die Beziehungen zwischen den einzelnen Dokumentabschnitten an.

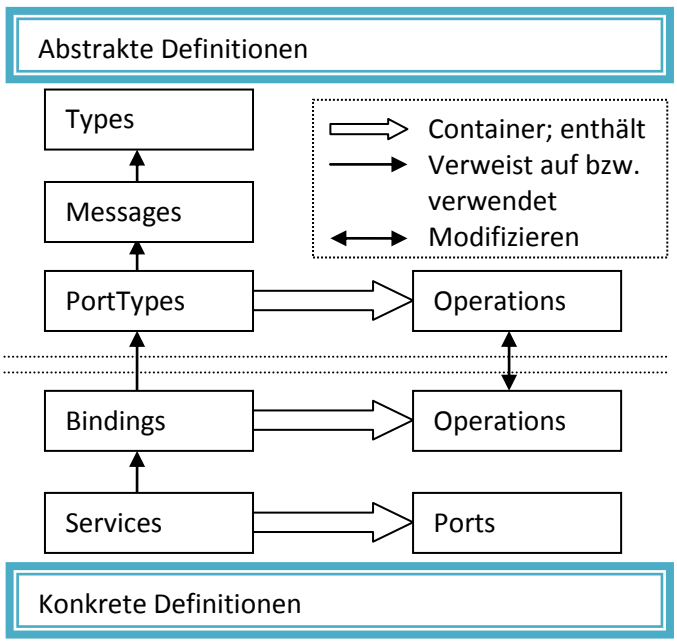



**Abbildung 7: Abstrakte und konkrete WSDL-Definitionen. Überarbeitet aus** [Tap]

### 2.1.3 UDDI – Universal Description, Discovery and Integration

In den bisherigen Kapiteln wurde beschrieben, wie man mit einem Web Service mittels SOAP Nachrichten austauschen kann bzw. wie mit einem WSDL-Dokument das Nachrichtenformat beschrieben werden kann. Jetzt stellt sich die Frage, wie man geeignete Web Services für die Entwicklung einer neuen Applikation findet. Man braucht also ein Verzeichnis, analog einem Telefonbuch für die Suche nach Telefonnummern, um darin nach dem gewünschten Service zu suchen. Die UDDI (Universal Description, Discovery and Integration) Spezifikation **[Cle04]** definiert Standards für webbasierte Registrierungsstellen, mit denen Web Services publiziert und gefunden werden können. Unternehmen bzw. Service Provider können Informationen über das Unternehmen selbst, Daten über die von ihnen zur Verfügung gestellten Services und technische Informationen, wie auf die Services zugegriffen werden kann, registrieren. Service Provider können somit UDDI nutzen, um für ihre Services zu „werben“. Service Benutzer können UDDI verwenden, um Services zu finden, die ihren Anforderungen entsprechen und um die notwendigen Service Daten zu erhalten, damit das Service benutzt werden kann.

Die Unternehmen IBM, Microsoft und Ariba waren die Initiatoren von UDDI. Die erste Version der UDDI-Spezifikation wurde im September 2000 veröffentlicht, UDDI v3.0.2 wurde von OASIS (Organization for the Advancement of Structured Information Standards) im Februar 2005 publiziert.

UDDI baut auf bestehende Internet Standards wie XML, HTTP, DNS (Domain Name System) auf, es ist Plattform unabhängig und benutzt SOAP als Transportschicht. UDDI-Server können als „*public*“ oder „*private*“ betrieben werden, wobei der Einsatzbereich vorwiegend intern ist. Einige der größten Mitwirkenden an der UDDI-Spezifikation (wie Microsoft, IBM und SAP) stellten ihre öffentlichen UDDI Server Ende 2006 ein. Ein öffentlicher UDDI-Server wird z.B. von der Universität Innsbruck, Semantic Technology Institute (STI) Innsbruck, betrieben (http://seekda.com/).

Die in UDDI gespeicherten Informationen lassen sich in drei Arten kategorisieren und spiegeln eine gewisse Ähnlichkeit zu einem Telefonbuch wider:

- **White pages:** Basisinformationen
  Die „*White pages*“ sind vergleichbar mit einem Telefonbuch und enthalten Daten über die Identität des Serviceanbieters. Dazu gehören Informationen wie die Adresse, Kontaktpersonen oder Informationen über das Geschäftsfeld.
- **Yellow pages:** Servicekategorisierung
  Die “*Yellow pages*” entsprechen dem Branchenverzeichnis. Die Services werden verschiedenen Kategorien zugeordnet und ermöglichen somit eine gezielte Suche.

- **Green pages:** Schnittstellenbeschreibungen eines Webservices

  In den *„Green pages“* findet man die technischen Daten der Web Services, die mit dem tModel (taxonomy model) gespeichert werden.

### *Datenstrukturen für UDDI*

Abbildung 8 zeigt die Datenstrukturen, welche für UDDI-Dienste definiert sind und wie Informationen über Unternehmen und deren angebotenen Dienste verwaltet werden können:

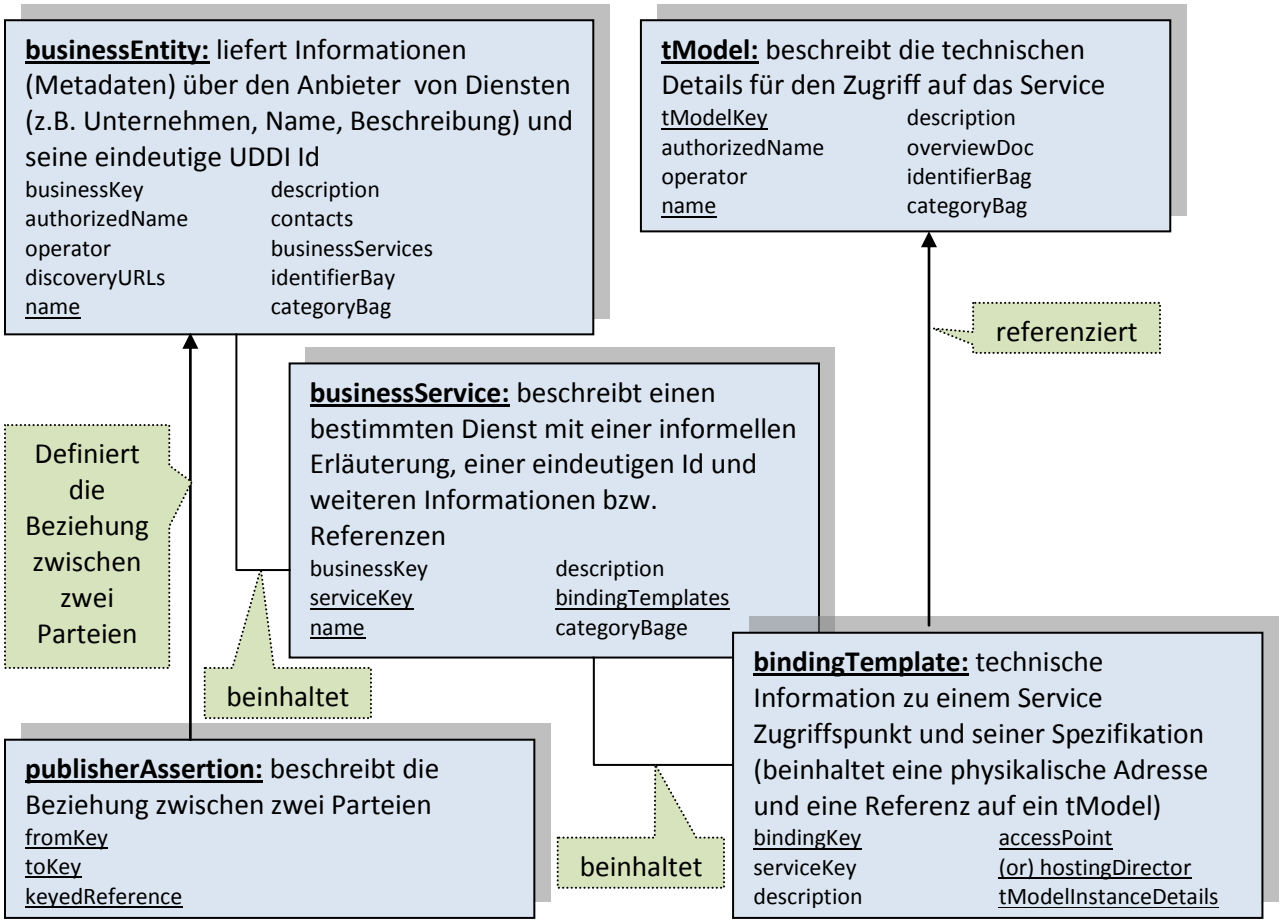



**Abbildung 8: Datenstrukturen in UDDI. Kombination aus den Quellen** [Kus02] **,** [New02]

UDDI-Elemente besitzen eindeutige Schlüssel – die UUIDs (Universally Unique Identifier), wie z.B. der *businessKey* eines *businessEntity* Elementes oder der *tModelKey*. Diese gewähren die Eindeutigkeit unter allen UDDI-Elementen und werden nach dem Algorithmus ISO/IEC 11578:1996 erzeugt.

Abbildung 9 stellt einen Überblick über den Zusammenhang zwischen WSDL und UDDI dar.

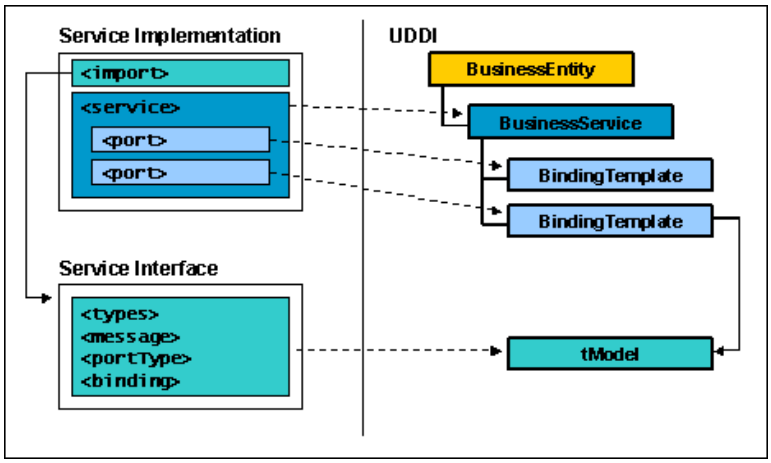



Abbildung 9: Zusammenhang WSDL & UDDI. Quelle [Bri01]

### *Finden bzw. Zugriff auf ein Service*

Für den Zugriff auf Einträge in einer UDDI Registry wird zwischen statischem und dynamischen Zugriff unterschieden. Bei einem statischen Zugriff erfolgt die Initiative durch den Nutzer des Services. Mit Hilfe eines UDDI-Browser wird das geeignete Service gesucht, die WSDL-Beschreibung des Services heruntergeladen und eine statische Anbindung an die Applikation implementiert. Der Vorteil liegt darin, dass die UDDI nicht mehr benötigt wird und durch das manuelle Auswählen genau der gewünschte Service gefunden wurde. Der Nachteil liegt in der engen Kopplung an das gefundene Service. Beim dynamischen Zugriff sucht die Applikation zur Laufzeit sich selbst das geeignetste Service durch Angabe diverser Parameter wie Preis, Verfügbarkeit oder Effizienz. Der Vorteil in dieser Variante liegt in der losen Kopplung zwischen Applikation und gefundenem Service. Ein Austausch des Services zu einem späteren Zeitpunkt durch ein die Anforderungen besser erfüllendes Service ist jederzeit möglich. Die Logik der Applikation ist aber weitaus komplexer, da der Zugriff auf das Service dynamisch auf Grund seiner WSDL-Beschreibung generiert werden muss.

## 2.2 Grid Computing

"The Internet is about getting computers to talk together;

Grid computing is about getting computers to work together."

Tom Hawk, IBM's general manager of Grid computing **[Bre07]**

### 2.2.1 Definition

Ian Foster und Carl Kesselmann definieren ein Grid System so:

> „Ein Computational Grid ist eine Hardware- und Software-Infrastruktur, die einen zuverlässigen, konsistenten, von überall erreichbaren und preiswerten Zugriff auf die Kapazitäten von Hochleistungsrechnern ermöglicht.“
>
> Englisches Original in **[Fos98]** , Übersetzung aus **[Gri]**

Einige Jahre später überarbeiteten die Autoren ihre Definition bzgl. Grid Computing:

> „Die gemeinsame Nutzung von Ressourcen, mit der wir uns hier beschäftigen, ist nicht primär der Austausch von Dateien, sondern vielmehr der direkte Zugriff auf Computer, Software, Daten und andere Ressourcen, wie sie bei einer Reihe von kollaborativen, problemlösenden und Ressourcen vermittelnden Strategien benötigt werden, die zurzeit in Industrie, Wissenschaft und im Ingenieurwesen auftauchen. Diese gemeinsame Nutzung von Ressourcen ist, notwendigerweise, in einem Höchstmaß kontrolliert, wobei die Anbieter und Konsumenten der Ressourcen klar und eindeutig festlegen, welche Ressourcen geteilt werden, wem die gemeinsame Nutzung erlaubt ist, und unter welchen Bedingungen die gemeinsame Nutzung erfolgt. Eine Menge von Individuen und/oder Institutionen, die sich durch solche Richtlinien zur gemeinsamen Nutzung von Ressourcen ergeben, formen das, was wir eine Virtuelle Organisation nennen.“
>
> Englisches Original in **[Fos04]** , Übersetzung aus **[Gri]**

Der Unterschied zwischen diesen beiden Definitionen besteht darin, dass in der früheren Version nur von Ressourcen in Form von Hochleistungsrechnern ausgegangen wurde. In der späteren Version wurden die dem Grid zugrunde liegenden Ressourcen relativiert – es ist nun von allgemeinen Ressourcen die Rede. Diese können nun auch Ressourcen wie Datenspeicher, technische Geräte & Experimente, etc. sein. Ein weiterer Unterschied zur ursprünglichen Definition besteht darin, dass die Nutzung der Ressourcen durch eine Virtuelle Organisation erfolgt. Der Begriff „Virtuelle Organisation“ wird später noch genauer erläutert.

Grid-Computing ist aus den Anforderungen bzgl. verteilten Rechnens entstanden. Mehrere Rechner, Supercomputer oder Cluster werden zu einem „virtuellen Supercomputer“ zusammengefasst. Sogar Grids selbst können wiederum zu einem Grid zusammengeschlossen werden. Der Unterschied zu Clustern oder Supercomputern besteht in der losen Kopplung zueinander. Eine Erweiterung oder Verkleinerung der zu einem Grid zusammengeschlossenen Elemente ist möglich. Ein weiterer Unterschied zu Supercomputern oder Clustern besteht darin, dass unterschiedliche Ressource Typen zu einem Grid zusammengefasst werden können.

Die Vision eines globalen Grids ist mit dem Internet vergleichbar. Das Internet besteht aus vielen Netzwerken, die wiederum aus kleineren Netzwerken bestehen, etc. Das globale Grid besteht aus vielen Grids, die wiederum aus kleineren Grids zusammengefügt sind, usw.

Ian Foster hat eine 3-Punkte-Checkliste **[Fos02]** erstellt, die die Eigenschaften eines Grid-Systems definiert:

Ein Grid-System …

- **… benutzt standardisierte, offene, allgemein verwendete Protokolle und Interfaces.**

  Ein Grid ist aufgebaut auf allgemeine Protokolle und Interfaces, die fundamentale Funktionen wie Zugriff auf Ressourcen, Authentifizierung, Autorisierung und Ressourcen-Findung zur Verfügung stellen. Im Gegensatz zu applikationsspezifischen Systemen werden bei Grid Systemen offene und standardisierte Protokolle und Interfaces verwendet.

- **… koordiniert Ressourcen, die nicht zentral kontrolliert werden.**

  Ein Grid integriert und koordiniert Ressourcen und Nutzer, die sich in unterschiedlichen Umgebungen befinden (z.B. Desktopcomputer im Gegensatz zu Zentralrechnern; Vernetzung von Rechnern in unterschiedlichen, separat administrierten Abteilungen eines Unternehmens bzw. von unterschiedlichen Unternehmen)

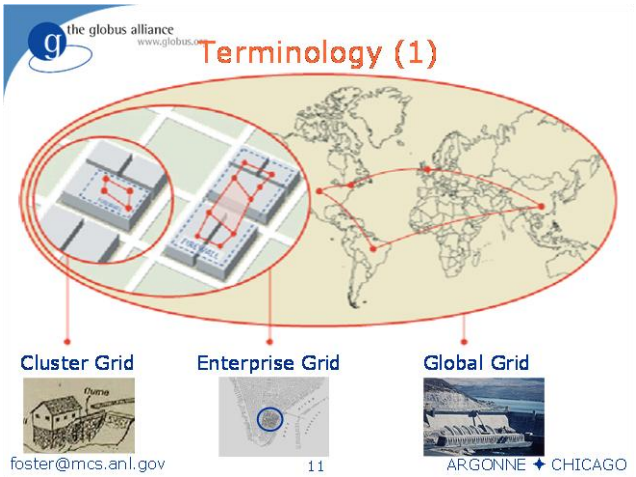



**Abbildung 10: Terminologie bzgl. Grid-Größe. Quelle** [Fos03]

- **… ermöglicht das Benutzen nicht-trivialer Dienstgüter.**

  Ein Grid ermöglicht die Nutzung der einzelnen Ressourcen im Verbund in einer Art und Weise, um verschiedene Qualitätsausprägungen von Diensten bzgl. Antwortzeit, dem Durchsatz, der Verfügbarkeit und der Sicherheit zu erzielen und/oder um komplexe Benutzererwartungen zu erfüllen. Der Nutzen der Ressourcen im Grid System muss eindeutig größer sein als die Summe einer separaten Nutzung der einzelnen Ressourcen.

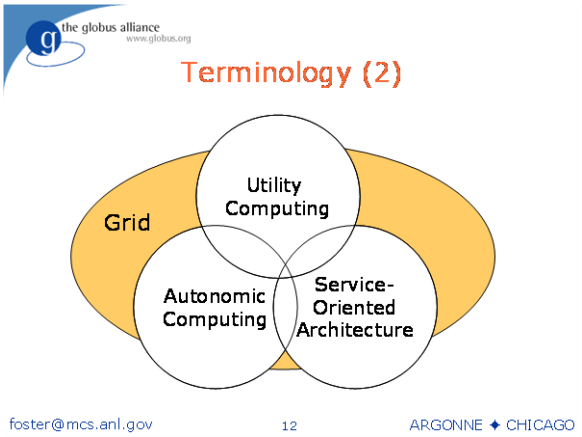



**Abbildung 11: Eigenschaften eines Grid Systems. Quelle** [Fos03]

Die Studie „Grid Characteristics and Uses: a Grid Definition“ **[Bot04]** hat folgende 10 Eigenschaften eines Grid Systems ermittelt:

- **Ressourcenumfang:** Ein Grid System muss bis zu einige Millionen Ressourcen verwalten bzw. zur Verfügung stellen können.
- **Geographische Verteilung:** Ressourcen eines Grid Systems können sowohl lokal gebündelt als auch durch weite Entfernungen getrennt sein.
- **Heterogenität:** Eine Nutzung von unterschiedlichsten Ressourcetypen muss möglich sein (Unterschiede bzgl. Ressource Typ, Software und Hardware: z.B. Speicher, wissenschaftliche Instrumente, PDA, Supercomputer, Router, Dateien, …)
- **Ressourcen Nutzung:** Die Ressourcen eines Grid Systems gehören unterschiedlichen Unternehmen und Organisationen – eine gemeinsame Nutzung ist möglich.
- **Lokale Administration:** Jede Organisation administriert ihre eigenen Ressourcen. Somit kommen unterschiedliche Administrations- und Sicherheitswerkzeuge bzw. Richtlinien zum Einsatz.
- **Ressourcen Koordinierung:** Eine Nutzung von Ressourcen im Verbund muss möglich sein.
- **Transparenter Zugriff:** Ein Grid System kann als EIN virtueller Computer betrachtet werden, der Rechen-, Speicherkapazität und andere Ressourcen zur Verfügung stellt.
- **QoS (Quality of Service):** Die Nutzung der Ressourcen unter Einhaltung diverser Qualitätskriterien wird durch das Grid System gewährleistet.

- **Einheitlicher Zugriff:** Ein Grid System setzt auf standardisierte Protokolle, Services und Interfaces auf. Es ermöglicht den einheitlichen Zugriff auf die Ressourcen und verbirgt deren Heterogenität.
- **Gewährleisteter Zugriff:** Grid Systeme sind ständig durch Änderungen der Ressourcen wie Ausfälle, Wartungen, Reorganisationen beeinflusst. Trotzdem muss der Zugriff auf die verfügbaren Ressourcen gewährleistet und deren optimale Nutzung möglich sein.

### 2.2.2 Geschichte

Der Begriff „*Grid*“ führt auf die Vergleichbarkeit mit dem Stromnetz (engl. „Power grid“) zurück. Die grundlegenden Eigenschaften der Stromversorgung entsprechen ebenfalls denen einer Grid-Umgebung: Zuverlässigkeit, hohe Verfügbarkeit, bedarfsgerechte und koordinierte Zuteilung der Ressourcen den jeweiligen Nutzern. Der Zugriff auf die Dienste erfolgt wie beim Stromnetz durch eine standardisierte Art und Weise (genormte Schnittstellen). Diese Parallelen führten zur Namensgebung „*Grid*“.

### 2.2.3 Virtuelle Organisation

Eine Virtuelle Organisation (VO) ist der Zusammenschluss mehrerer Parteien, die gemeinsam Ressourcen wie Software, Hardware, Daten und andere Ressourcen nutzen. Parteien können nun von einzelnen Personen bis zu ganzen Organisationen und Unternehmen sein. Durch Zugriffsregeln ist festgelegt, wie und welche Ressourcen durch die VO genutzt werden können. Eine Partei kann mehreren VOs angehören. Genauso ist es möglich, dass eine Ressource von mehreren VOs genutzt wird.

Nachfolgende Grafik verdeutlicht den Zusammenhang zwischen realen Organisationen (die drei Kreise in der Mitte) und den virtuellen Organisationen P & Q:

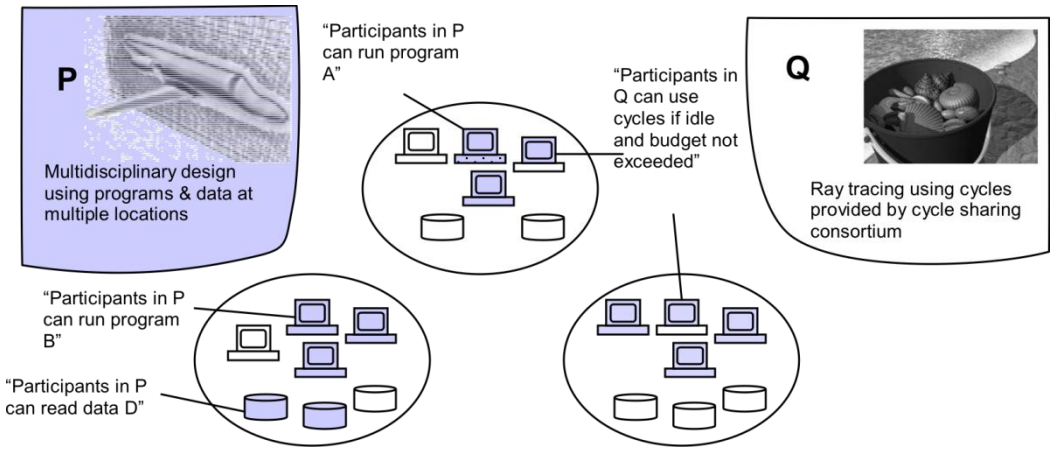



**Abbildung 12: Reale und virtuelle Organisationen. Quelle** [Fos]

### 2.2.4 Kategorisierung von Grids

#### *Kategorisierung auf Grund des Applikationstyp*

- **Computational Grids:** Mehrere Rechner werden zu einem Grid zusammengefasst und stellen die Rechenkapazitäten gemeinsam als Ressource zur Verfügung. Dies ist die häufigste Anwendung von Grid Systemen.
- **Data Grid:** Verteilte Datenbestände werden in einem Data Grid zu einer „virtuellen“ Superdatenbank zusammengefasst. Dies ist die zweithäufigste Anwendung von Grids, abgesehen von der Nutzung als Computational Grids.
- **Scavenging Grid:** Die Leistung eines Rechners (Workstation, PC, etc.) wird nur selten voll ausgenützt. Die meiste Zeit wird nur ein Bruchteil der maximalen Kapazität genutzt. Diese brachliegenden Rechenressourcen werden in Scavening Grids genutzt. Ein bekanntes Beispiel für Scavening Grids ist das SETI-Projekt[18] der Universität Berkeley, welches zur Suche von außerirdischen Radiosignalen dient. Die notwendige, enorme Rechenleistung zur Analyse der aufgezeichneten Radiosignale wird durch verteiltes Rechnen in Form ungenützter Rechenkapazitäten durch die SETI-Gemeinschaft erbracht.
- **Semantic Grids:** „Das Semantic Grid ist eine Grid-Weiterentwicklung, in dem Informationen und Services eine wohldefinierte Bedeutung in Form einer maschinell verarbeitbaren Beschreibung zugewiesen wird, um eine bestmögliche Nutzung durch Mensch und Computer zu erzielen.“ Übersetzt aus **[Sem]**
- **Knowledge Grids:** Knowledge Grids dienen zur Suche, Extraktion, Integration und Synthese von Wissen aus verteilten Quellen, stellen eine Architektur für PDKD Systemen (Parallel and Distributed Knowledge Discovery) dar und basieren auf Semantic Grids. **[Wah04]** , **[Zhu04]**

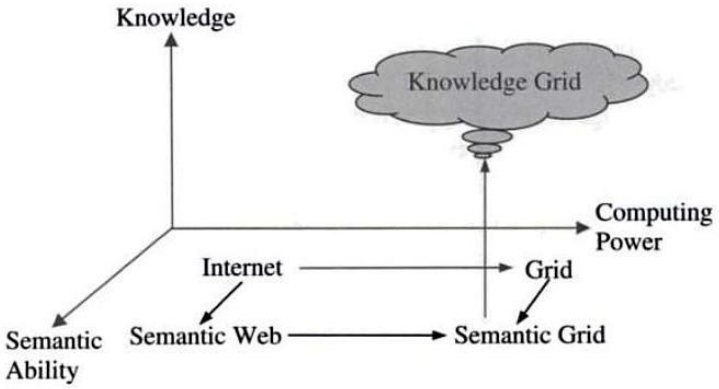



**Abbildung 13: Knowledge Grids, Semantic Grids, Semantic Web, Grids und Internet.**
**Quelle** [Zhu04]

[18] SETI - Search for extraterrestrial intelligence at home

### *Kategorisierung auf Grund der Größe*

Die Abbildung 10 illustriert Grids in unterschiedlichen Größen:

- **Cluster Grid:** Clustersysteme sind die kleinste Ausprägung eines Grids. Die Elemente sind innerhalb eines Raumes, vielleicht sogar innerhalb eines Racks untergebracht. Anforderungen bzgl. der Netzwerkarchitektur sind auf Grund des geringen Abstandes der einzelnen Elemente untereinander vernachlässigbar. Intensive Ressourcennutzung der Rechenkapazität kann jedoch schnell an die Grenzen des Grid Systems führen.
- **Enterprise Grid:** Als Enterprise Grids bezeichnet man die Zusammenfassung mehrerer Cluster Grids. Eine Nutzung erfolgt bereits abteilungs- bzw. unternehmensweit. Durch die unternehmensinterne Art des Grids ist eine Homogenität der Grid Elemente noch vorhanden. Trotzdem spielen Aspekte wie Autorisierung, Abrechnung bzw. Verrechnung bereits eine bedeutende Rolle.
- **Campus Grids:** Ähnlich zu einem Enterprise Grid gibt es an (technischen) Universitäten eine Vielzahl von PC-Labs, die zu Labor-Clustern zusammengefasst werden können. Als Campus Grids bezeichnet man die Instituts- oder Fakultätsweite Bündelung mehrerer Labor-Cluster. Der Vorteil von Campus Grids liegt in der hohen Anzahl an verfügbaren Rechnern mit geringer Auslastung.
- **Global Grids:** Diese Art von Grids lässt sich mit dem Internet vergleichen. Eine globale Ausdehnung, unterschiedlichste Benutzertypen und deren Autorisierung, Heterogenität der Ressourcen, verschiedenste Sicherheitskonzepte etc. stellen eine große Herausforderung an das Grid System bzw. dessen Komponenten.

## 2.2.5 Architektur

Eine Grid Architektur besteht laut Ian Foster aus mehreren Ebenen **[Fos]** :

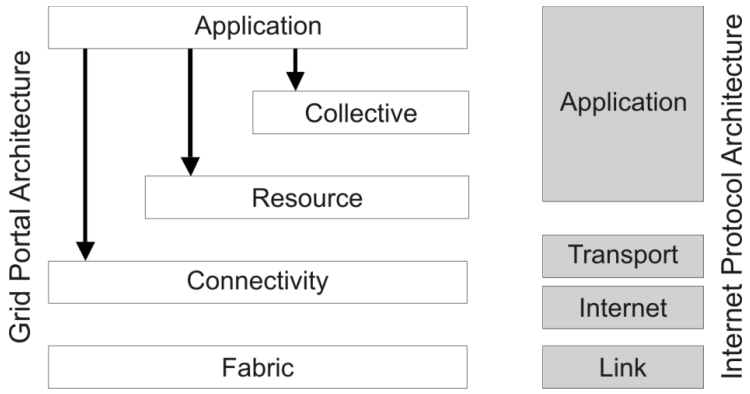



**Abbildung 14: Grid Architektur. Quelle** [Fos]

### *Fabric Schicht: Interfaces für die lokale Kontrolle*

Die *Fabric* Schicht beinhaltet die Ressourcen, die dem Grid zur Verfügung stehen. Dies können sowohl (physikalische) Ressourcen wie Rechnerkapazität, Speichersysteme, Netzwerk Ressourcen, etc. sein, als auch logische Ressourcen. Logische Ressourcen wären z.B. ein Cluster oder ein verteiltes Dateisystem – beide sind logische Einheiten, die wiederum aus anderen Ressourcen bestehen.

In der *Fabric* Schicht sind die lokalen, Ressource-spezifischen Operationen implementiert, um den höheren Schichten den Zugriff zu ermöglichen. Dabei ist zu beachten, dass wenige bzw. einfache Operationen in dieser Schicht angeboten werden, um in höheren Schichten eine einfachere und dienstübergreifende Nutzung zu ermöglichen.

Um Ressourcen im Grid effektiv zu nutzen bzw. zu reservieren, müssen folgende Anforderungen bei der Ressourcen-Findung (auf Grund diverser Kriterien wie Status, Eigenschaften) und Management erfüllt sein:

| Ressource Typ | Ressource Anforderungen |
|---|---|
| Rechenkapazitäten | Es gibt Mechanismen zum Starten und Beenden von Programmen bzw. für deren Statusabfrage. Um Kapazitäten zur Verfügung zu stellen, gibt es außerdem Operationen wie die Informationsbereitstellung der zu Grunde liegenden Hard- und Software bzw. Reservierung von Rechenkapazitäten (dies ist z.B. durch die Kontrolle der Anzahl laufender Jobs, der Systemauslastung, etc. möglich). |
| Speicherressourcen | Mechanismen zum Lesen und Schreiben von Daten, zur Komprimierung, für die Datensicherheit (wie Mirroring, Stripping) müssen vorhanden sein. Informationen bzgl. Soft- und Hardware (z.B. Speichergröße, Datendurchsatz) werden ebenfalls bereit gestellt, genauso wie Funktionen zur Reservierung und Zustandsermittlung (z.B. aktuelle Speicherauslastung). |
| Netzwerkressourcen | Mechanismen zur Kontrolle der Netzwerkressource bzw. des Datentransfers (z.B. mittels Priorisieren, Reservieren) und der Informationsbereitstellung sind notwendig. |

| | |
|---|---|
| Code-Repository | Beispiele für ein Code-Repository wären CVS[19], ClearCase; sie stellen Mechanismen zur Verwaltung von Software-Versionen zur Verfügung. |
| Kataloge | Kataloge sind ein Spezialfall der Speicherressourcen und bieten zusätzliche Such- und Aktualisierungsfunktionen. Relationale Datenbanken sind hierzu ein Beispiel. |

### *Connectivity layer: Kommunikation und Sicherheit*

Die *Connectivity* Schicht stellt Kommunikations- und Authentifizierungsprotokolle zur Verfügung. Durch die Kommunikationsprotokolle wird der Datenaustausch zwischen den Ressourcen in der *Fabric* Schicht ermöglicht. Die Authentifizierungsprotokolle ermöglichen durch kryptografische Sicherheitsmechanismen die sichere bzw. eindeutige Identifizierung von Benutzern und Ressourcen.

Als Kommunikationsprotokolle werden die Protokolle des TCP/IP-Protokollstapels (siehe Abbildung 14: Grid Architektur) verwendet: TCP & UDP[20] für die Transportschicht; IP & ICMP[21] für die Internetschicht; DNS, OSPF[22], RSVP[23] in der Applikationsschicht.

Bzgl. Authentifizierung müssen folgende Kriterien erfüllt sein:

| | |
|---|---|
| Einmaliges Anmelden (Single sign-on): | Ein Anmelden am Grid System reicht aus, um auf die berechtigten Ressourcen zugreifen zu können. Weitere Benutzeraktionen bzgl. Authentifizierung sind nicht notwendig. |
| Integration der lokalen Sicherheits-lösungen: | Da eine Virtuelle Organisation aus mehreren realen Organisationen besteht, sind innerhalb eines Grid Systems mehrere, unterschiedliche Sicherheitslösungen vorhanden. Ein „Zusammenspiel“ dieser Sicherheitslösungen ist erforderlich. |
| Delegation: | Ein Programm muss durch das Aufrufen durch den Benutzer die Berechtigung bekommen, auf die dafür notwendigen Ressourcen zugreifen zu können. Dies schließt auch die Übertragung der Rechte (Delegation) auf ein anderes Programm ein. |

[19] CVS - Concurrent Versions System
[20] UDP - User Datagram Protocol
[21] ICMP - Internet Control Message Protocol
[22] OSPF - Open Shortest Path First
[23] RSVP - Resource Reservation Protocol

| | |
|---|---|
| Benutzer-basiertes Sicherheits-konzept: | Um Ressourcen unterschiedlicher Organisationen gemeinsam nutzen zu können, sind keine Interaktionen bzgl. Sicherheit notwendig. Kann ein Benutzer die Ressource X der Organisation A und die Ressource Y der Organisation B nutzen, so hat diese auch die Berechtigung AB gemeinsam zu nutzen. |

### *Resource layer: Nutzung einzelner Ressourcen*

Die *Resource* Schicht ist für den Zugriff bzw. Verteilung der einzelnen Ressourcen zuständig. Es sind dazu Protokolle für Verhandlungen, Verbindungsinitiierung, Überwachung, Ab- und Verrechnung bzgl. der Ressourcennutzung definiert.

Es wird dabei zwischen Informations- und Managementprotokollen unterschieden:

- Informationsprotokolle: dienen nur zur Informations- und Statusabfrage der Ressource, wie Konfiguration, Auslastung, etc. und haben keine Zustandsänderung der Ressource zur Folge.
- Managementprotokolle: dienen zur Verhandlung bzgl. Ressourcennutzung und bewirken in Folge eventuell eine Zustandsänderung der Ressource. Es werden Anforderungen an die Ressource geregelt, als auch der Zugriff und die Ausführung von Operationen bzw. deren Überwachung.

### *Collective layer: Koordinierung mehrerer Ressourcen*

Der Unterschied zur *Resource* Schicht besteht darin, dass die *Collective* Schicht Interaktionen mit mehreren Ressourcen ermöglicht. Diese Schicht ermöglicht außerdem eine Vielfalt an zusätzlichen Funktionen, ohne dass die Ressourcen diese Dienste anbieten müssen. Einige Beispiele **[Fos]** :

- Verzeichnisdienste ermöglichen das Finden von Ressourcen und deren Eigenschaften. Die Suche kann mittels Ressourcenamen oder mittels –attributen erfolgen.
- Scheduling und Broker Dienste ermöglichen die gemeinsame Reservierung und Terminisierung (Co-Scheduling) von Aktivitäten mit mehreren Ressourcen.
- Monitoring und Diagnosefunktionen ermöglichen die Überwachung von Ressourcen in Bezug auf Fehler, Überlastung, etc.
- Datenreplikationen, um die Zugriffsperformance und Zuverlässigkeit zu erhöhen
- Suchfunktionen für Software ermöglichen die Auswahl der am besten geeigneten Software und Plattform aufgrund der gewünschten Anforderungen.

Die Dienste der *Ressource* Schicht stehen dem ganzen Grid System zur Verfügung. Das Anbieten der Dienste der *Collective* Schicht erstreckt sich vom ganzen Grid bis zu einzelnen Anwendungen oder Domänen innerhalb einer Virtuellen Organisation.

Die Implementierung der Funktionen der *Collective* Schicht kann als eigener Dienst mit dazugehörigen Protokollen oder in Form von SDKs (Software Development Kits) zur Einbindung in Anwendungen erfolgen.

### *Application layer:*

Die *Application* Schicht ist die oberste Schicht der Grid Architektur und beinhaltet die Anwendungen der Gridbenutzer, die auf die Dienste in den darunter liegenden Schichten zugreifen. Um diese Dienste zu nutzen, stellt jede Schicht entsprechende APIs und SDKs bereit und ermöglicht so den Zugriff mit den entsprechenden Protokollen.

In Abbildung 15 wird der Zugriff der Applikationen auf die Dienste der Collective Schicht bzw. auf die Dienste der Resource Schicht dargestellt. I.A., wird ein Ressourcenverbund benötigt, so erfolgt der Zugriff über die Collective Schicht. Werden jedoch nur einzelne Ressourcen benötigt, so wird direkt auf die Resource Schicht zugegriffen.

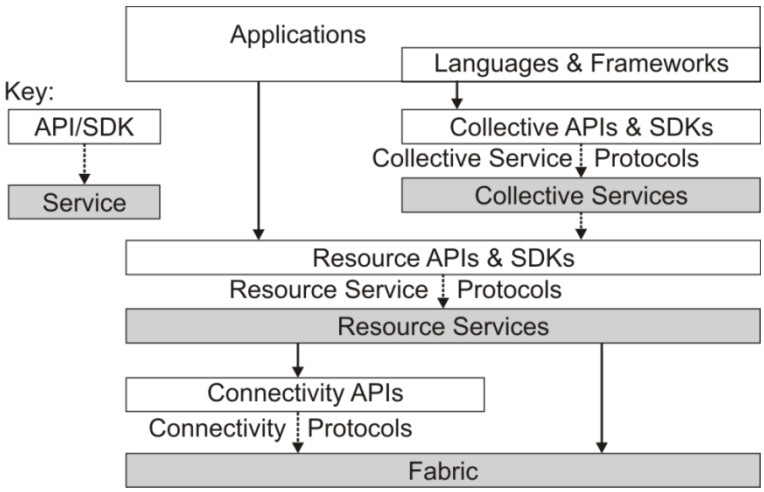



**Abbildung 15: APIs, SDKs & Protokolle. Quelle** [Fos]

## 2.2.6 OGSA

Das Open Grid Form (OGF) ist ein Konsortium für Standards, Spezifikationen und Empfehlungen im Bereich der Grid-Technologie. Ein wichtiger von OGF veröffentlichte Standard ist die Open Grid Services Architecture (OGSA) aus dem Jahre 2002, die eine allgemeine, offene Service-orientierte Grid-Infrastruktur spezifiziert. Diese Spezifikation ist jedoch nicht in technischer Hinsicht zu verstehen, sondern eher als Vision bzw. Idee aller benötigten Services einer derartigen Infrastruktur.

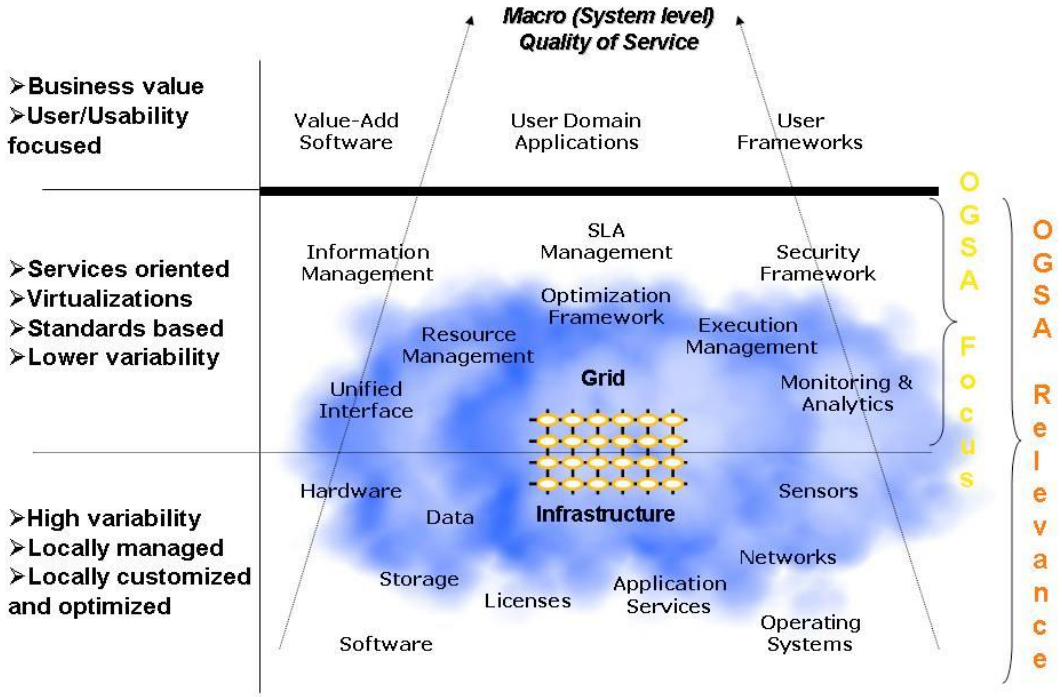



Abbildung 16: Konzeptuelle Sicht einer Grid-Infrastruktur. Quelle [Fos06]

Die Abbildung 16 zeigt die konzeptuelle Sicht einer Grid-Infrastruktur, die eine Vielzahl an unterschiedlichsten Services zur Verfügung stellt. Die Grafik ist in drei Bereiche unterteilt. Der unterste Bereich der Grafik entspricht den Basis-Ressourcen bzw. Basis-Services. Diese Basis-Ressourcen entsprechen logischen oder physikalischen Ressourcen wie z.B. CPUs, Netzwerke, Hauptspeicher, Festplatten. Beispiele für logische Ressourcen sind Software, Lizenzen und Betriebssysteme. Die Virtualisierungen dieser Ressourcen als Services werden auf Grund der engen Beziehung zur Ressource wie die Ressource selbst benannt und stellen somit Basis-Services dar.

Der Fokus von OGSA liegt auf den Services in der mittleren Schicht, welche Services mit größerer logischer Abstraktion beinhaltet. Diese Services können auch als Standard-Services bezeichnet werden und sind in den meisten Grid-Systemen in unterschiedlicher Ausprägung wiederzufinden.

Die oberste Schicht repräsentiert Applikationen oder andere Objekte, welche die Services der Mittelschicht benutzen.

Die Services der virtualisierten Ressourcen dienen als Dienstleister für die Services in den oberen Schichten bzw. werden von diesen auch verwaltet. Services in der Mittelschicht können ihre Funktionalität nur aufgrund der Basis-Services anbieten. Die Services der unteren Schicht sind zwar für OGSA relevant, die Vision von OGSA liegt aber in der Standardisierung der Services der Mittelschicht.

Diese Standard-Services lassen sich in folgende Kategorien unterteilen:

- Infrastructure Services
- Execution Management Services
- Data Services
- Resource Management Services
- Security Services
- Self-Management Services
- Information Services

Diese Standard-Services oder auch Grid-Services sind in OGSA nicht weiter spezifiziert. Es ist nur festgelegt, dass es Web Services mit definierten Interfaces sind und sie bestimmten Konventionen folgen. Durch die Open Grid Services Infrastructure (OGSI) werden diese Grid-Services spezifiziert.

### *OGSI*

Die OGSA beschreibt die Grid-Services einer Grid-Infrastruktur bzw. wie eine Grid-Architektur aufgebaut ist. Die Open Grid Services Infrastructure (OGSI), ist eine detailierte formale und technische Spezifikation der Grid Services in OGSA und definiert ein einheitliches Verhalten aller Grid Dienste bzw. eine standardisierte Interaktion (Interfaces) zwischen den Services.

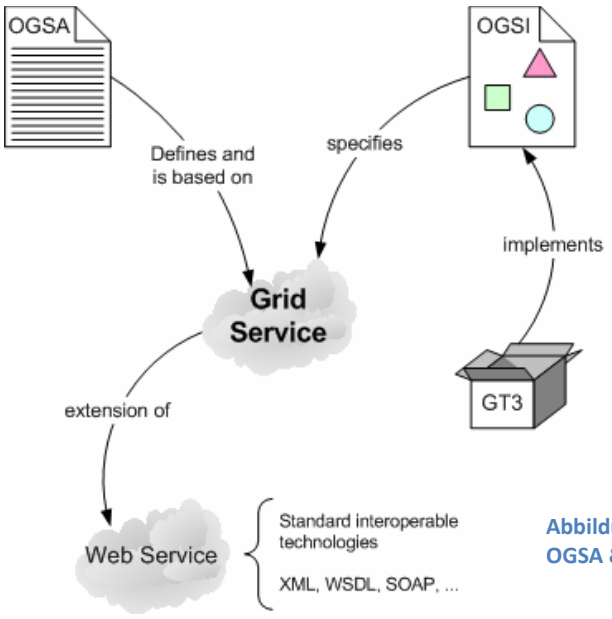



**Abbildung 17: Grid-Services in OGSA & OGSI. Quelle** [Sch05]

Web Services sind zustandslos, durch OGSI werden Web Services zu „Stateful Web Services“. Grid Services (ersichtlich in Abbildung 17) sind Web Services mit einer Menge von Erweiterungen wie dem Zustand eines Services, der Lebenszeitverwaltung, den Dienste-Daten und Dienste-Gruppen sowie der Schnittstellen-Definitionen.

Gründe, die zur Ablöse des OGSI-Standards durch das WSRF (Web Services Resource Framework) führten, waren u.a. folgende:

- das Konzept der Service-Instanzen: Im OGSI-Modell wurde für jeden Benutzer eine Instanz eines Grid-Services erzeugt.

- mangelnde Kooperation mit existierenden XML- und Web Service-Toolkits
- unzureichende Harmonie mit den Web Service Spezifikationen

### *WSRF (Web Services Resource Framework)*

Die WSRF-Definition von OASIS lautet: *„Defining an open framework for modeling and accessing stateful resources using Web services.“* **[OAS]**

Bei WSRF wurden die Grid-Dienste des OGSI-Modells – die „Stateful Web Services“ - durch Web Services ersetzt, die nun auf „Stateful Resources“ zugreifen.

Das WSRF umfaßt folgende fünf Spezifikationen zur standardisieren Interaktion mit „Stateful Resources“ über „Stateless Web Services“:

- WS-Resource: definiert eine WS-Resource als eine Ressource und ein dazugehöriges Web Service, welches den Zugriff auf die Ressource ermöglicht
- WS-ResourceProperties: definiert ein Interface um Properties zu einer Ressource zu verwalten
- WS-ResourceLifetime: definiert ein Interface um die Lebensdauer einer Ressource zu überwachen
- WS-BaseFaults: beschreibt einen Mechanismus zur einheitlichen Handhabung von SOAPFaults
- WS-ServiceGroup: beschreibt ein Interface zur Aggregation oder Gruppierung von WS-Resources zu einer ServiceGroup und ermöglicht dadurch Operationen auf eine Gruppe von WS-Resources

Die Abbildung 18 zeigt den unterschiedlichen Ursprung von Grid- und Web-Technologie. Globus Toolkit 1 (GT) und GT 2 sind Sammlungen von Tools, um Grid-Computing zu ermöglichen.

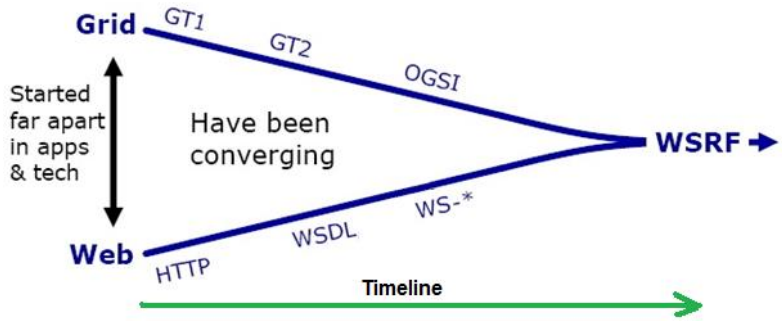



**Abbildung 18: Die Entwicklung von Grid- und Webtechnologien konvergiert zu WSRF. Überarbeitet aus** [Sch05]

In OGSI (und GT3) wurden Services als „Stateful Web Services“ realisiert, um eine Annäherung an die Web-Technologie zu erzielen. Erst WSRF führte zur Konvergenz der Web- und Gridtechnologie und dient nun als Basis für weitere Entwicklungen in beiden Bereichen. Globus Toolkit 4, UNICORE 6 und WSRF.Net sind Grid-Implementierungen, die auf den WSRF-Spezifikationen basieren.

## 2.3 Datenzugriff und Datenintegration

Der Datenzugriff und die -integration von heterogenen, meist sehr großen Daten stellt einen wichtigen Aspekt im Grid Computing dar. Folgender Beitrag aus der ORF-Futurzone vom 3. Okt. 2008 zeigt die zunehmende Bedeutung:

> **„LHC-Grid nimmt Betrieb auf - Datenverarbeitung für Teilchenbeschleuniger**
>
> Rund drei Wochen nach dem Start des Teilchenbeschleunigers Large Hadron Collider (LHC) hat das CERN am Freitag in Genf die ersten Daten aus dem Experiment im LHC Computing Grid, einem weltweiten Verbund von Rechenzentren verarbeitet.
>
> Das Grid stellt die benötigte Computerleistung (**mehr als 100.000 Prozessoren aus 140 Einrichtungen in 33 Ländern**) für die Auswertung der Daten aus dem LHC zur Verfügung.
>
> Der LHC liefert rund 5.000 Wissenschaftern in 500 Forschungsinstituten weltweit **15 Millionen Gigabyte im Jahr**.“
>
> http://futurezone.orf.at/it/stories/312163/

### 2.3.1 Heterogene, verteilte Datenquellen

Die Bedeutung der Verfügbarkeit von Daten als Grid Services nimmt zu. Die Grid-Technologie ermöglicht einen homogenen Zugriff auf heterogene und/oder verteilte Daten. Weiters sind Grid-Systeme konzipiert, enorme Datenmengen zu verarbeiten. Der Beitrag in der ORF Futurezone zeigt die immer wichtiger werdende Rolle von Grid-Systemen in der Forschung, als auch in der Wirtschaft. Im Falle von Cern wird durch die Grid-Technologie Wissenschaftern auf der ganzen Welt ein einheitlicher Zugriff auf die riesigen Datenmengen ermöglicht. Außerdem veranschaulicht dieses Beispiel, dass durch die Nutzung von Grid-Systemen nicht nur der weltweite Zugriff auf die Daten von Cern ermöglicht wird, sondern Wissenschafter dadurch auch beliebig untereinander auf die Untersuchungen und Resultate anderer Kollegen zugreifen und Daten austauschen können.

Die Arbeitsgruppe Open Grid Forum (OGF) entwarf auf Grund des OGSA-Standards eine Architektur, um den Zugriff auf heterogene und verteilte Daten in einem Grid-System zu ermöglichen. Das Ergebnis war die Open Grid Services Architecture - Data Access and Integration (OGSA-DAI), welche als Referenzimplementierung realisiert wurde und nun den defacto Standard darstellt, um heterogene verteilte Datenressourcen als Services anzubieten.

### 2.3.2 OGSA-DAI Projekt

Das Ziel des OGSA-DAI (Open Grid Services Architecture Data Access and Integration) Projektes ist die Entwicklung bzw. Weiterentwicklung einer Middleware Software, um den Zugriff auf heterogene, verteilte Datenquellen in einem Grid-System zu ermöglichen. OGSA-DAI ist ein Projekt der UK Database Task Force, welche eng mit der Open Grid Forum Database Access and Integration Services Working Group (Open Grid Forum DAIS-WG) zusammen arbeitet. Außerdem findet eine Zusammenarbeit mit einer Reihe von anderen Projekten (z.B. Globus, ADMIRE, OMII-Europe, NextGRID, SIMDAT BEinGRID) statt, um die Funktionalität der OGSA-DAI Software in einer Vielzahl an Grid-Systemen zu gewährleisten.

### 2.3.3 OGSA-DAI Software

OGSA-DAI ist ein Java Software System, welches auf WS-Standards basiert und ermöglicht den Zugriff auf Datenressourcen (relationale und XML Datenbanken) und deren Integration als Services in Grid-Systemen. Es bietet verschiedenste Interfaces und unterstützt den Zugriff auf gängige Datenbanksysteme (z.B. MySQL, Oracle). Es bietet diverse Komponenten um das Abfragen, Aktualisieren, Transformieren und Bereitstellen von Daten auf unterschiedlichste Arten zu ermöglichen. Außerdem wird ein Werkzeugsatz bereitgestellt, um Client Applikationen entwickeln zu können. OGSA-DAI wurde so entworfen, dass es erweiterbar ist und den Nutzern ermöglicht, eigene Funktionalitäten zu entwickeln und bereitzustellen.

OGSA-DAI unterstützt folgenden Funktionsumfang:

- den Zugriff auf Datenressourcen mittels Web Services: Der Zugriff auf gängige relationale und XML Datenbanken, sowie der Zugriff auf Dateien/Verzeichnisse wird ermöglicht. Unterstützte Datenquellen werden im Kapitel „OGSA-DAI Data Resources” behandelt.
- die Abfrage und Aktualisierung der Daten in jedem dieser Datenressourcen
- die Transformation von Daten mittels XSLT[24]. Eine Komprimierung und Dekomprimierung der Daten mittels ZIP und GZIP wird unterstützt.
- das Bereitstellen von Daten für Clients, andere OGSA-DAI Web Services (*Data Staging*), URLs[25], FTP Server, Grid FTP Server oder in Dateien

[24] XSL - Extensible Stylesheet Language, XSLT - XSL Transformation
XSLT ist eine Programmiersprache zur Transformation eines XML-Dokumentes und meist resultiert daraus wieder ein Dokument in XML-Syntax; es können aber auch Text-, Binärdateien, etc. erzeugt werden.
[25] URL - Uniform Resource Locator

- *„Uniform Interface“* - einen einheitlichen Zugriff auf die Web Services unabhängig von der Art der Datenressource, auf die zugegriffen wird. OGSA-DAI greift intern dann dem Datenressource Typ entsprechend zu.
- den Zugriff der Clients auf Metainformationen bzgl. der Datenressource (z.B. Schema)
- die Erweiterbarkeit der Web Services von OGSA-DAI, um nicht unterstützte Datenressourcen freizugeben und applikationsspezifische Funktionalitäten zu ermöglichen. Eine Erweiterung ist z.B. die Implementierung von Grid Data Mediation Service (GDMS), das den Zugriff auf mehrere heterogene Datenressourcen in Form einer virtuellen Datenressource ermöglicht. **[Wöh06]**

### *Response und Perform-Dokumente*

Die Kommunikation mit einer OGSA-DAI Datenressource erfolgt dokumentenorientiert mit *Perform*- und *Response*-Dokumenten und ist in Abbildung 19 dargestellt. Der Client sendet ein *Perform*-Dokument (XML-basiert) an das *Data Service*, welches dieses an die *Data Service Resource* weiterleitet. Die *Data Service Resource* interpretiert die Anfrage, führt die gewünschte Aktivität auf die *Data Resource* aus und liefert das Ergebnis in Form eines *Response*-Dokumentes zurück.

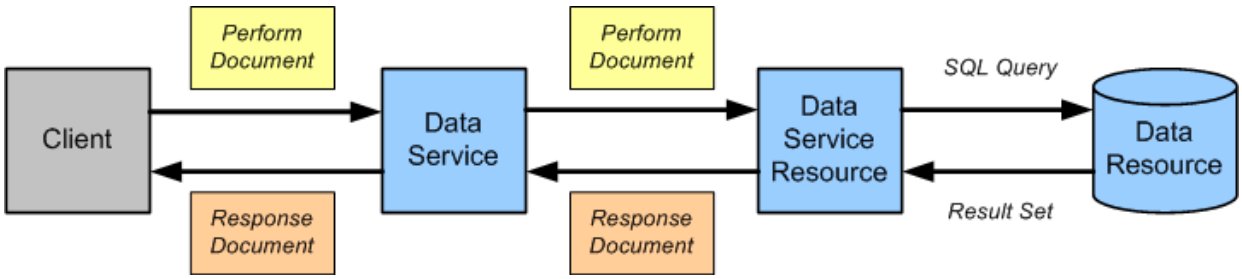



**Abbildung 19: Perform und Response Dokumente. Quelle** [OGS2]

Ein *Perform*-Dokument beinhaltet die gewünschte(n) Aktivität(en), die auf der *Data Resource* durchgeführt werden sollen. Aktivitäten können z.B. sein:

- **Relationale Aktivitäten:** Datenabfragen (*select*), Datenaktualisierungen (*insert*, *update*, …), Aufruf gespeicherter Prozeduren, Laden von Massendaten (*BulkLoad*), Konvertierung des Abfrageergebnisses in XML oder CSV[26] Format, …
- **XML Aktivitäten:** Abfragen und Aktualisierung auf XML-Datenbanken, …
- **Übertragen von Daten (*Delivery*):** Holen und Übertragen von Daten an eine URL, in eine Datei, mittels Grid-FTP, …

[26] Das Dateiform CSV (Comma-Separated Values) ist ein Format zur Speicherung von Daten in einer Textdatei. Die Daten werden meistens durch Kommas getrennt, jedoch sind andere Trennzeichen ebenfalls möglich.

- **Transformation von Daten:** Umwandlung der Ergebnisse durch eine XSL-Transformation, Komprimieren und Entpacken mit ZIP & GZIP, …

Listing 2 ist ein Beispiel für eine typische Aktivität auf einer Datenressource. Es werden Daten abgefragt (*sqlQueryStatement*) und diese im XML-Format in Form eines *WebRowSets* zur Verfügung gestellt.

```
1  <?xml version="1.0" encoding="UTF-8"?>
2  <perform xmlns="http://ogsadai.org.uk/namespaces/2005/10/types">
3      <documentation>This example performs a simple select statement to
           retrieve all rows from the specified database. The results are
           delivered within the response document.
4      </documentation>
5      <session/>
6      <sqlQueryStatement name="statement">
7          <expression>select * from littleblackbook where id='10'</expression>
8          <resultStream name="myOutputRS"/>
9      </sqlQueryStatement>
```

myOutputRS Stream

```
10     <sqlResultsToXML name="statementRSToXML">
11         <resultSet from="myOutputRS"/>
12         <webRowSet name="statementOutput"/>
13     </sqlResultsToXML>
14 </perform>
```

**Listing 2: sqlQuery und sqlResult. Quelle** [OGS3]

Im Anhang ist ein *Perform*-Dokument (siehe „8.1.2 SOAP-Request mit Attachment“, grüner Abschnitt in der SOAP-Nachricht) und das dazugehörige *Response*-Dokument (siehe „8.1.3 SOAP-Response mit Attachment“, grüner Abschnitt) im Zuge einer VGE-Kommunikation zu sehen.

### *OGSA-DAI Architektur*

Die OGSA-DAI Architektur besteht aus mehreren Ebenen. Eine schematische Darstellung dieser Architektur ist in Abbildung 20 gezeigt.

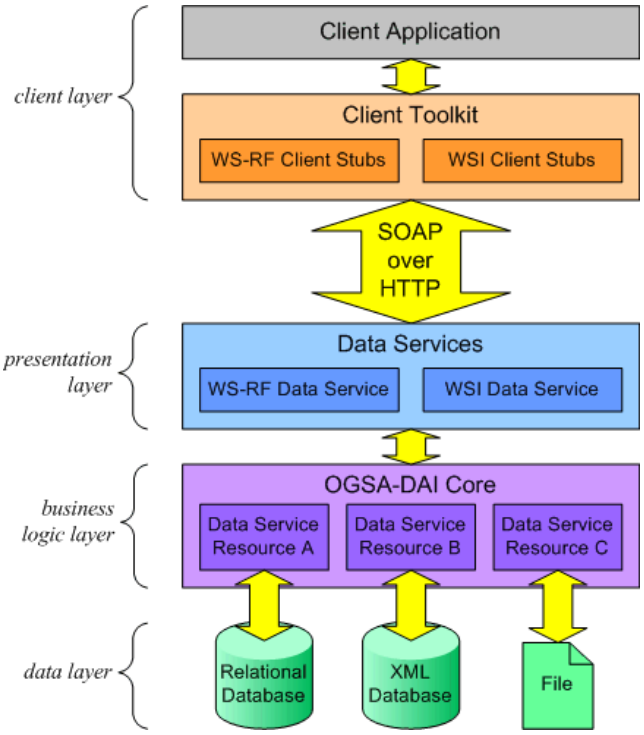



**Abbildung 20: OGSA-DAI Architektur. Quelle** [OGS]

**Data Layer:** Diese Schicht repräsentiert die eigentlichen Datenressourcen, auf die mit OGSA-DAI zugegriffen werden kann. Kapitel „OGSA-DAI Data Resources” enthält eine Auflistung der unterstützten Datenressourcen.

**Business Logic Layer:** Der Business Logic Layer stellt die Kernfunktionalität von OGSA-DAI dar.

Für jeden Datenressource-Typ gibt es einen entsprechenden *Data Resource Accessor* im *Business Logic Layer*, mit dem ein einheitlicher Zugriff auf die Datenressource für die *Data Service Resource* ermöglicht wird. OGSA-DAI ist erweiterbar, d.h. soll eine eigene bzw. derzeit nicht unterstützte Datenressource verwendet werden, so muss dafür ein separater *Data Resource Accessor* entwickelt werden.

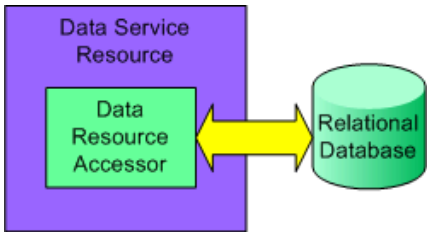



**Abbildung 21: Data Service Resource, Data Resource Accessor & Data Resource. Quelle** [OGS]

Für jede zu verwaltende *Data Resource* gibt es im *Business Logic Layer* eine *Data Service Resource*, die folgende Aufgaben hat:

- das Ausführen von *Perform*-Dokumenten
- das Erzeugen des *Response*-Dokumentes inklusive Status und eventuellen Ergebnisdaten

- den Zugriff auf die Datenressource mit dem entsprechenden *Data Resource Accessor*
- den Datentransport: Streaming zwischen Datenressource und Clients bzw. anderen *Data Service Resources*
- das Session Management
- das Property Management: Verwaltung von *Data Service Resource* Properties zu einer Datenressource. Dies sind normalerweise Metadaten, wie der Status einer Aktion auf eine Ressource.

**Presentation Layer:** Der Presentation Layer macht die Nutzung der OGSA-DAI Funktionalität in einem Grid-System möglich und stellt den Zugriff auf die *Data Service Resources* mittels Web Service Interfaces zur Verfügung.

Diese Schicht beinhaltet die *Data Services*, deren wichtigste Aufgabe es ist, die Kommunikation mit den Clients über die Web Services gemäß der WSDL-Schnittstelle zu ermöglichen und die Nachrichten bzw. Dokumente an die entsprechenden *Data Service Resources* weiterzuleiten.

**Client Layer:** Der Client kann auf die *Data Service Resource* durch Nutzung der entsprechenden *Data Services* in der *Presentation* Schicht zugreifen.

Die *Client* Schicht beinhaltet außerdem ein Java Client Toolkit, welches ein High-Level-API für den Zugriff auf *Data Services* bietet und dadurch auf einfache Weise das Erzeugen und Senden von Requests bzw. das Auswerten des Responses für Client Applikationen ermöglicht.

### *OGSA-DAI Data Resources*

OGSA-DAI Version 3.0 ermöglicht den Zugriff auf folgende Datenressourcen:

| Data Resource | Version |
|---|---|
| Relational | |
| MySQL | 5.0.15 |
| IBM DB2 | 8.1 FixPack 5 |
| Microsoft SQL Server | 2000 |
| Oracle | 10g Enterprise Edition Release 10.2.0.1.0 – Production |
| PostgreSQL | 8.1.4 |
| XML | |
| eXist | 1.1.1 |
| Dateien und Verzeichnisse | |
| Unix, Linux and Windows | |
| Sonstige | |
| OGSA-DAI Resource Gruppe | Dies ist eine OGSA-DAI Ressource, in der andere OGSA-DAI Ressourcen zu einer Ressource zusammgefasst wurden. |

**Tabelle 2: OGSA-DAI Data Resources. Quelle** [OGS1]

## 2.4 Heterogene Clients

Die Heterogenität ist nicht nur Service-seitig z.B. in Bezug auf Ressourcen vorhanden, sondern ist auch in den Clients wiederzufinden, die Web Services nutzen. Abgesehen von der Verwendung durch Computer, spielt die Nutzung von Web Services durch unterschiedlichste Geräte wie mobile Endgeräte, Haushaltsgeräte etc. eine immer wichtigere Rolle. Durch die Plattformunabhängigkeit von Java besteht die Möglichkeit, eine Client-Applikation für diverse Einsatzgebiete zu erstellen.

### 2.4.1 Java Portabilität

Ein Vorteil von Java ist die Plattformunabhängigkeit, die dem Motto von SUN: „Write once, run everywhere“ folgt. Diese Portabilität wird durch die Übersetzung des Quellcodes in einen Standard-Bytecode und die Ausführung dieses Bytecodes in einer Java Virtual Machine (JVM) erreicht. JVMs sind für viele Betriebssysteme und Plattformen verfügbar.

Das Prinzip der Portabilität wird unter anderem dadurch ermöglicht, dass es Java bzw. die Java Plattform in 3 verschiedenen Ausprägungen gibt:

- Java 2 Plattform, Enterprise Edition – J2EE: wird eingesetzt im Unternehmensbereich für sichere, hochverfügbare und skalierbare Anwendungen
- Java 2 Plattform, Standard Edition – J2SE: wird für den „normalen“ Einsatz verwendet
- Java 2 Plattform, Micro Edition – J2ME: dies ist die kleinste Ausprägungen und wird in mobilen Endgeräten eingesetzt.

Die Heterogenität der Clients (Clients für PCs/Servers versus Clients auf Mobile Information Devices – MIDs) spielt somit keine Rolle, da durch die Wahl der entsprechenden Java Plattform und der daraus resultierenden JVM, die Portabilität gewährleistet ist. William Wong schrieb einen Artikel **[Won02]** , in dem angeführt wird, dass das Motto anstelle „Write once, run everywhere“ korrekterweise „Write once, debug everywhere“ sein müsste, da eine Anpassung bzw. Adaptierung an die entsprechende Umgebung aufgrund der großen Heterogenität der Einsatzgebiete notwendig bzw. nicht vermeidbar ist. In den nächsten Kapiteln werden daher einige Themen bzgl. der Entwicklungen von Applikationen für MIDs behandelt.

### 2.4.2 J2ME - Java 2 Micro Edition

J2ME wurde speziell für Geräte mit begrenzten Ressourcen entworfen und wird daher für die Entwicklung von Java Applikationen in Mobiltelefonen, PDAs, Funkmeldeempfänger (Pager) als auch Set-Top Boxen, Fernsehgeräten mit Internetfunktion, etc. verwendet. In Abbildung 22 werden diverse Geräten mit den darauf eingesetzten Java Edititionen dargestellt.

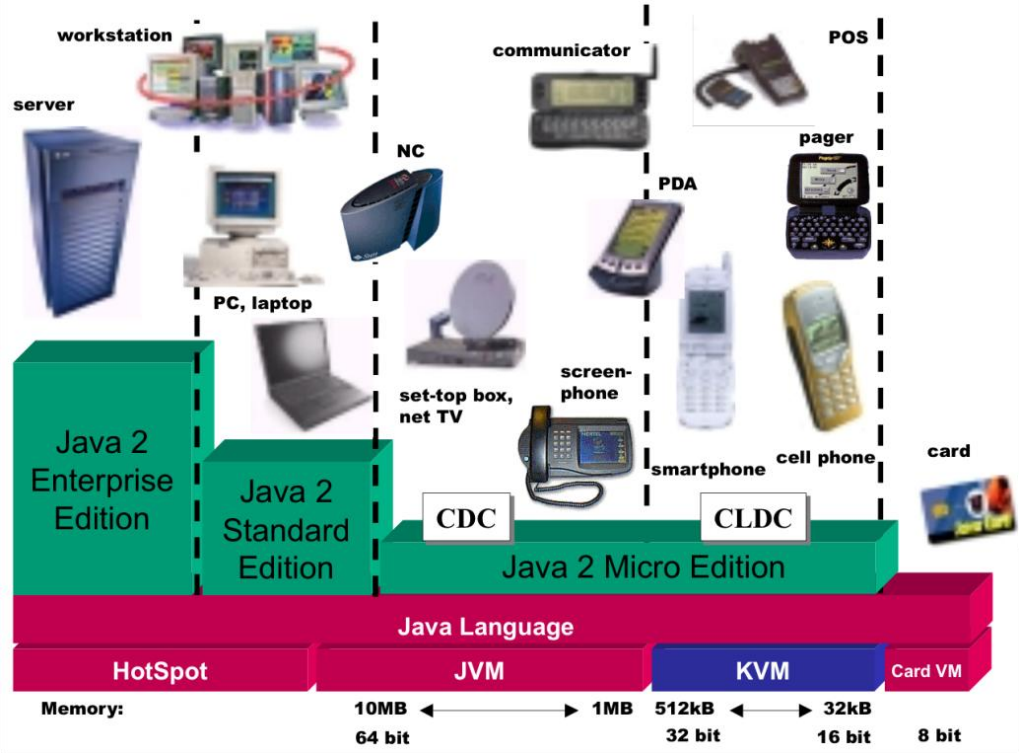



Abbildung 22: Java Plattform Versionen und ihr Einsatzgebiet. Quelle [J2M00]

Durch die Vielzahl von Geräten und das daraus resultierende breite Spektrum bzgl. Geräteeigenschaften, Ausstattung und Kapazitäten hat SUN die Konzepte *Konfiguration* und *Profile* eingeführt.

Durch eine **Konfiguration** ist die VM (Virtual Machine) und der minimale Satz an Klassenbibliotheken für alle Geräte einer Klasse festgelegt. Sie definiert das Minimum an Hardwareanforderungen, die ein Gerät erfüllen muss, um die J2ME-Laufzeitumgebung zu betreiben. Hierbei werden Mindestanforderungen bzgl.

- Hauptspeicher (Minimum, Typ),
- Prozessor (Typ),
- Netzwerkverbindung

festgelegt.

Ein **Profil** ist die Erweiterung einer Konfiguration, um bestimmte Leistungsmerkmale zu erfüllen. Die durch ein Profil zur Verfügung gestellten Klassenbibliotheken erweitern somit den, durch die Konfiguration bereits vorhandenen Grundstock von Klassen.

Es gibt die Konfigurationen CDC (Connected Device Configuration, **[Rig06]** ) und CLDC (Connected Limited Device Configuration, **[Rig07]** ) und das MIDP (Mobile Information Device Profile, **[Mil06]** ) Profil.

### 2.4.3 CLDC - Connected Limited Device Configuration

Geräte für die CLDC 1.1-Umgebung sind laut Spezifikation folgendermaßen charakterisiert:

- mindestens 192 KB (Kilobyte) Speicher:
  - mindestens 160 KB nichtflüchtiger Speicher für die KVM [27]und die CLDC-Bibliotheken
  - mindestens 32 KB flüchtiger Speicher für die Ausführung der KVM
- 16 oder 32-bit Prozessor
- geringer Stromverbrauch, Stromzufuhr über Batterien
- (falls vorhanden) eine Netzwerkverbindung mit geringer Bandbreite (die Datenrate für die Datenübertragung im GSM[28]-Kanal beträgt 9600 Bit pro Sekunde bzw. 14400 Bit pro Sekunde bei guten Funkverhältnissen)

In der CLDC wird außerdem festgelegt **[Szc06]** :

- Java Sprachumfang und die Features der Virtual Machine
- Grundlegende Java Bibliotheken (java.lang.*, java.util.*)
- Eingabe/Ausgabe
- Grundlegende Netzwerkfunktionen
- Sicherheit
- Internationalisierung

[27] KVM steht für Kilobyte Virtual Machine. Die JVM (Java Virtual Machine) wird auf Grund ihrer geringen Größe (ca. 50-80 KB) so benannt.
[28] GSM ist die Abkürzung für Global System for Mobile Communications

### 2.4.4 MIDP - Mobile Information Device Profile

Im Jahr 2002 veröffentlichte SUN das MIDP Profil 2.0. An einer MIDP 3.0 Version wird bereits gearbeitet und ist als JSR 271 (Java Specification Request) vorgesehen. Das MIDP erweitert die CLDC, um verschiedene Funktionen wie z.B. Netzwerkfunktionen, Speichermöglichkeiten, Benutzeroberflächen.

„Abbildung 23: J2ME Architektur“ und „Abbildung 24: Pakete der CLDC-Konfiguration und des MID-Profils“ zeigen den Zusammenhang zwischen Konfiguration und Profil.

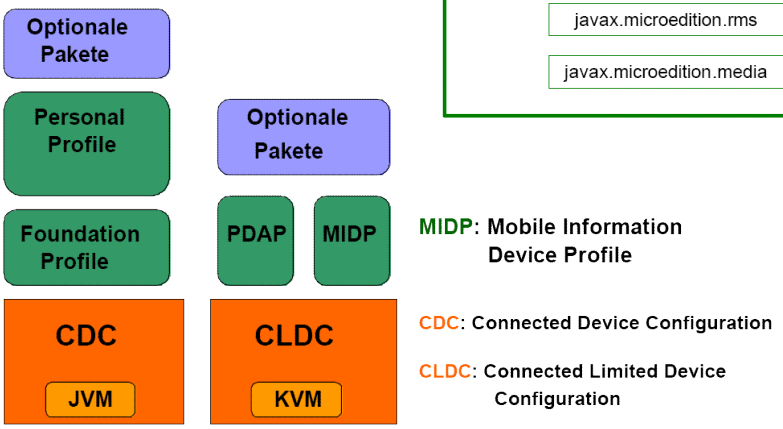



Abbildung 23: J2ME Architektur. Quelle [Ver04]

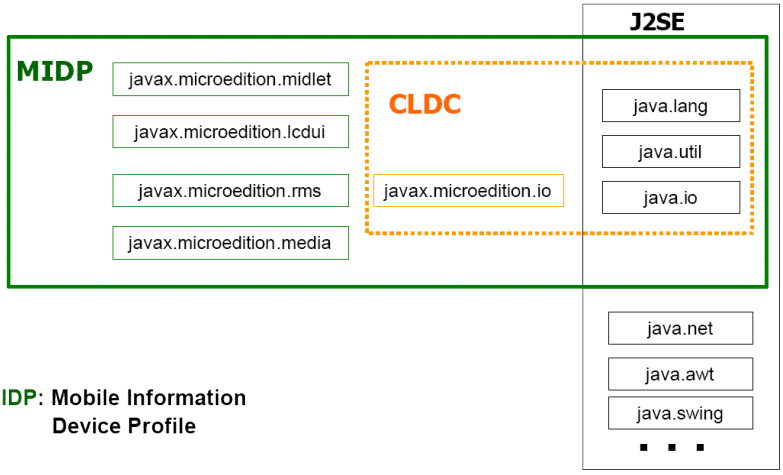



Abbildung 24: Pakete der CLDC-Konfiguration und des MID-Profils. Quelle [Ver04]

Die durch das MIDP zur Verfügung gestellten APIs können durch optionale Pakte noch erweitert werden, um spezifische, fortgeschrittene Funktionalitäten zu erhalten. Beispiele dafür sind XML Parser, SIP-Unterstützung, Bluetooth-Funktionalität, zusätzliche Kommunikationsprofile (wie SMS[29], HTTPS, Sockets, …), usw. In „Abbildung 25: API Selektion im Sun Java Wireless Toolkit“ sieht man die Möglichkeit, optionale zusätzliche APIs zu der Ziel-Plattform (Kombination aus CLDC, MIDP und bereits vorselektierten zusätzlichen APIs) auszuwählen.

[29] SMS – Short Message Service

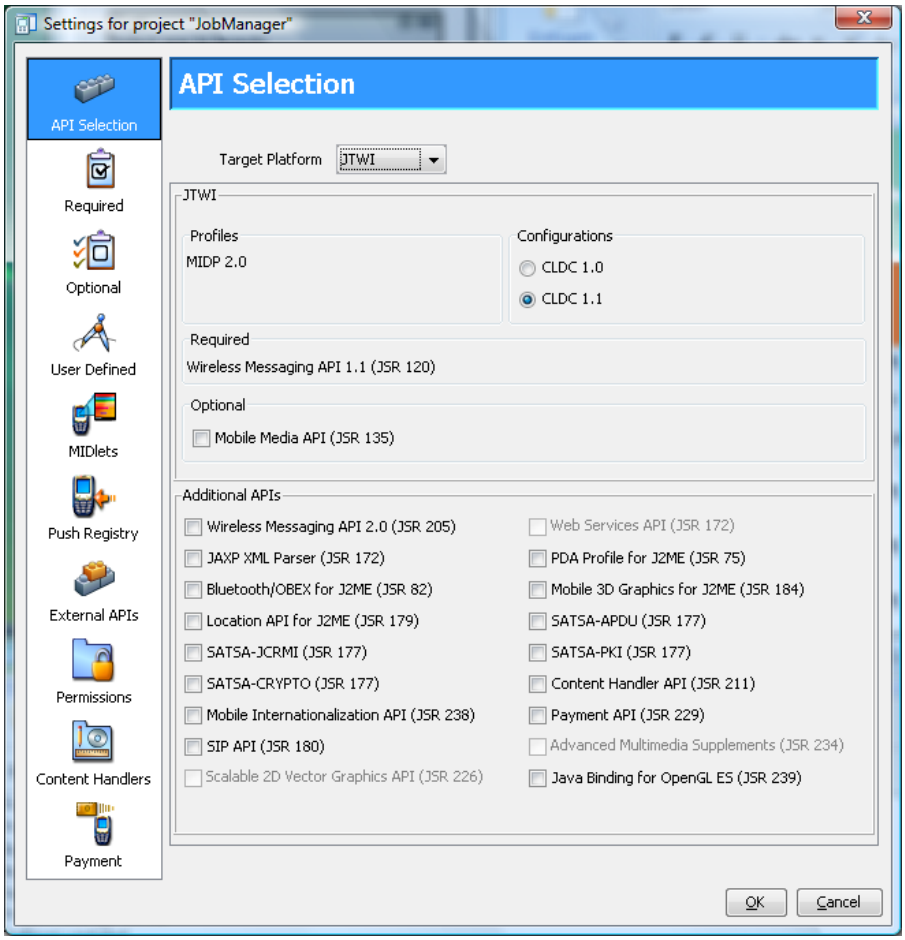



**Abbildung 25: API Selektion im Sun Java Wireless Toolkit (Screenshot)**

Durch MIDP werden dem Entwickler auch APIs für die Gestaltung der Benutzeroberfläche zur Verfügung gestellt. Sie bestehen aus einer kleinen Anzahl an User-Interface-Elementen, wobei zwischen Low- und High-Level-API unterschieden werden muss.

Das Low-Level-API basiert auf der Pixeldarstellung und bietet sämtliche Funktionen zur grafischen Darstellung (z.B. drawLine, drawString, setColor, fillRect, …). Das High-Level-API bietet hingegen fertige Elemente zur Benutzereingabe wie ChoiceGroup, DataField, List, …

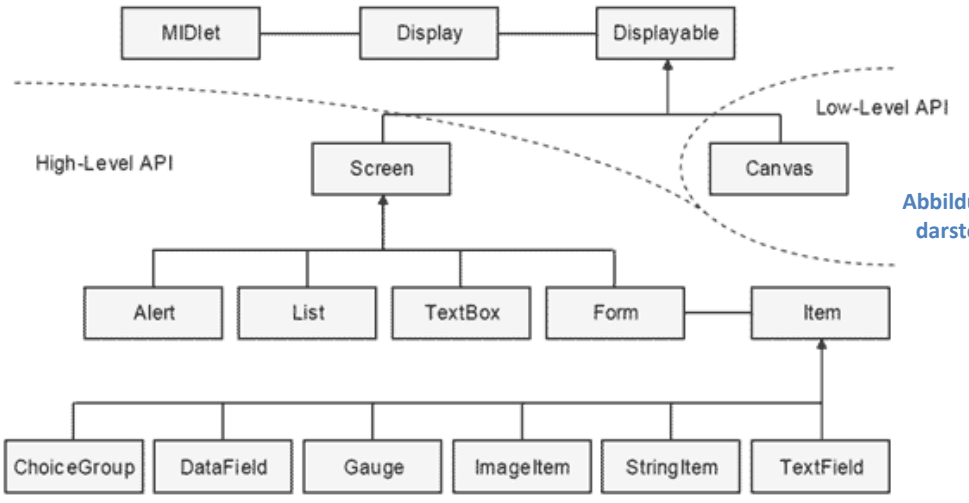



**Abbildung 26: Klassenhierarchie für darstellbare Elemente des MIDP. Quelle** [Szc06]

### 2.4.5 Erstellung einer MIDlet-Suite

MIDlets nennt man Java Applikationen, die für die MIDP Umgebung entworfen wurden. Ein oder mehrere MIDlets sind in einer MIDlet-Suite mit den dazugehörigen Ressourcen (z.B. Icons, Grafiken) zusammengefaßt, welche die kleinste installierbare Einheit auf einem MID (Mobile Information Device) bildet. In Abbildung 27 wird der Entwicklungsprozess dargestellt.

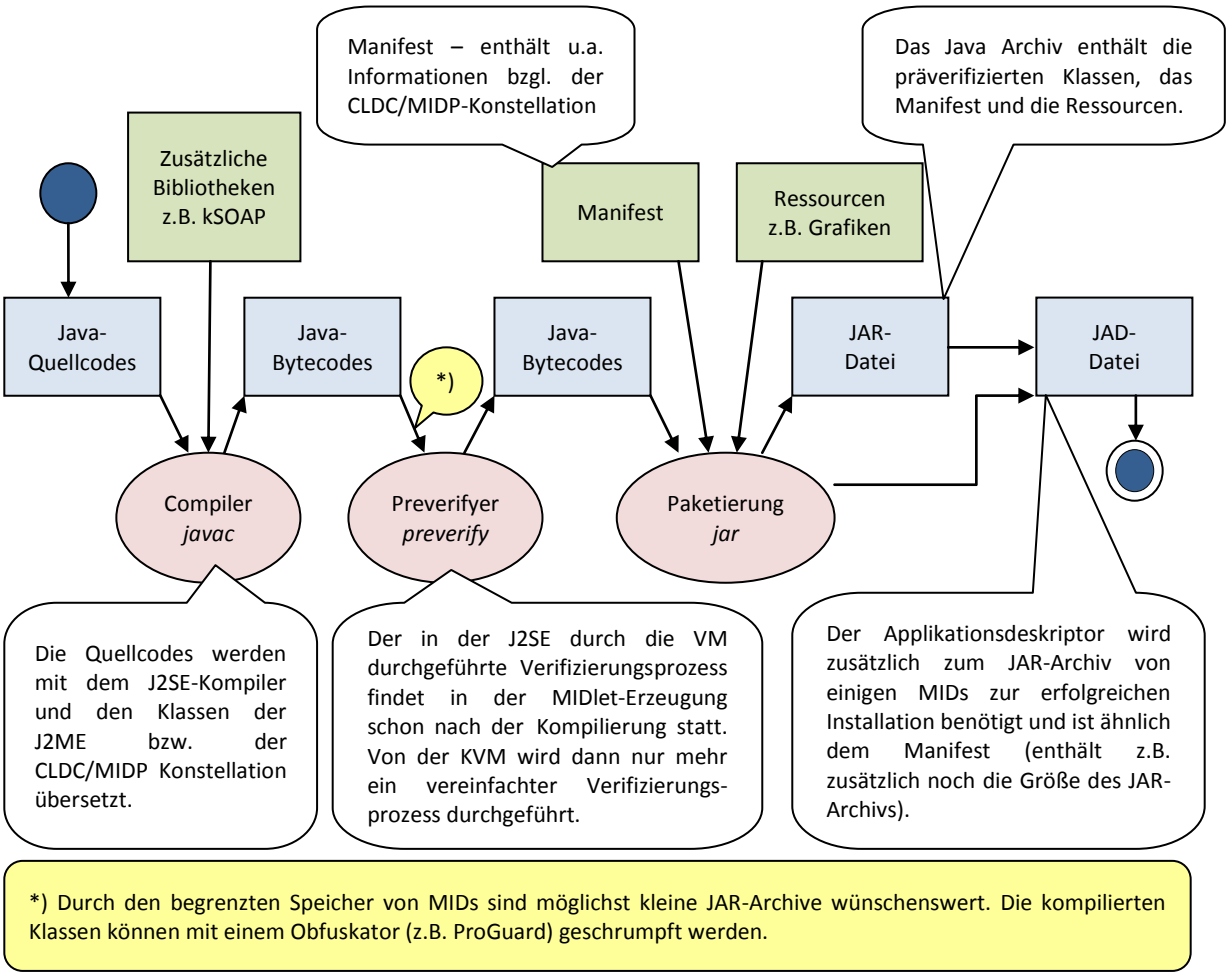



**Abbildung 27: Entstehungsprozess einer MIDlet-Suite. Überarbeitet aus** [Tob03]

Ein Beispiel für einen Applikationsdeskriptor (der Applikationsdeskriptor des Mobile Job Managers Clients, wie in Abbildung 44 dargestellt) ist in Listing 3 abgebildet:

```
 1  MIDlet-1: JobManager, /icons/JobManagerIcon.png,
        at.ac.univie.isc.jobmanager.mobile.MobileClient
 2  MIDlet-Description: Job Manager Client
 3  MIDlet-Icon: /icons/JobManagerIcon.png
 4  MIDlet-Jar-Size: 85804
 5  MIDlet-Jar-URL: JobManager.jar
 6  MIDlet-Name: JobManager
 7  MIDlet-Permissions: javax.microedition.io.Connector.http,
        javax.microedition.io.Connector.https
 8  MIDlet-Vendor: Weichselbaum Juergen
 9  MIDlet-Version: 1.0
10  MicroEdition-Configuration: CLDC-1.1
11  MicroEdition-Profile: MIDP-2.0
12
```

**Listing 3: Applikationsdeskriptor des „Mobile Job Manager Clients“**

Ein Applikationsdeskriptor (JAD-Datei) ist ähnlich dem Manifest eines Java Archives (JAR-Datei) und enthält zusätzlich Attribute wie z.B. die Größe und die URL des Java Archives. Für die Installation einer MIDlet-Suite auf einem mobilen Endgerät ist bei manchen Herstellern solch ein Applikationsdeskriptor notwendig. Soll eine MIDlet-Suite installiert werden, so wird zuerst (wenn notwendig) die JAD-Datei geladen und überprüft, ob diese MIDlet-Suite aufgrund von Konfiguration und Profil (Zeile 10 und 11) für das MID geeignet ist (*Over-the-Air Provisioning*). Nach erfolgreicher Überprüfung folgt das Herunterladen der JAR-Datei und die Installation.

### 2.4.6 Zustände eines MIDlets

Jedes MIDlet wird von der abstrakten Klasse *javax.microedition.midlet.MIDlet* abgeleitet, in der die folgenden Methoden definiert sind:

- `protected abstract void startApp() { … }`
- `protected abstract void pauseApp() { … }`
- `protected abstract void destroyApp(boolean b) { … }`

Um auf Zustandsänderungen reagieren zu können, werden diese drei Methoden vom System aufgerufen und müssen in der abgeleiteten Klasse implementiert werden. Als Auslöser für Zustandsänderungen ist z.B. ein eingehender Anruf zu verstehen.

Startet man ein MIDlet auf einem mobilen Endgerät, so erfolgt zuerst der Aufruf des leeren Konstruktors der MIDlet-Klasse, gefolgt von dem Aufruf des Konstruktors der abgeleiteten Klasse. In dem Konstruktor erfolgen normalerweise diverse Initialisierungen. Danach befindet sich das MIDlet im Zustand *Paused* bzw. *Loaded*. Beim ersten Mal wird dieser Zustand auch *Loaded* bezeichnet, danach wechselt das MIDlet zwischen den Zuständen *Active* und *Paused*. Vom System wird anschließend die *startApp*-Methode aufgerufen und das MIDlet wechselt in den Zustand *Active*. In der *startApp*-Methode erfolgt das erstmalige Laden von Ressourcen bzw. die Anzeige von Ausgaben und GUI[30]-Elementen. Zu beachten ist, dass die *startApp*-Methode nach jedem Zustandswechsel von *Paused* auf *Active* erneut aufgerufen wird. Initialisierungen und das Laden von Ressourcen sollten – in den meisten Fällen – nur beim ersten Aufruf erfolgen. Wird das MIDlet (durch den Benutzer oder durch das MIDlet selbst) beendet, so wird die *destroyApp*-Methode aufgerufen.

[30] GUI – Graphical User Interface; unter GUI-Elementen sind Eingabeelemente zur Interaktion mit den Benutzer zu verstehen wie Auswahllisten, Eingabefelder, etc.

Der gesamte Lebenszyklus eines MIDlets ist in Abbildung 28 beschrieben, der dazupassende Quellcode in Listing 4.

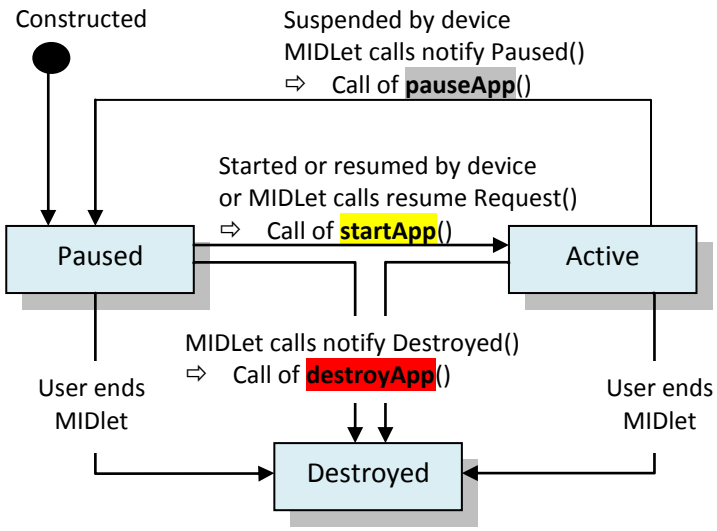



Abbildung 28: Lebenszyklus eines MIDlets. Überarbeitet aus [Top02]

```
import javax.microedition.midlet.*;

public class MobileClient
        extends MIDlet
{
    public MobileClient() {
    }

    public void startApp() {
    }

    public void destroyApp(boolean
                unconditional) {
    }

    public void pauseApp() {
    }
}
```

Listing 4: MIDlet Quellcode

# 3 VGE

Das Vienna Grid Environment System (VGE) ist eine Grid Umgebung und ermöglicht die Nutzung von bestehenden HPC (High Performance Computing) basierten Anwendungen bzw. den Zugriff auf Datenbestände als Grid-Services. Es benutzt Web Service Standards wie die zuvor behandelten Technologien SOAP, WSDL, UDDI als auch OGSA-DAI für Datenressourcen. Das VGE wurde vom Institut für Scientific Computing der Universität Wien entwickelt.

## 3.1 Architektur

Die VGE Architektur folgt einer klassischen, Service-orientierten Architektur mit Clients, Services, Registries und Zertifizierungsstellen (CAs):

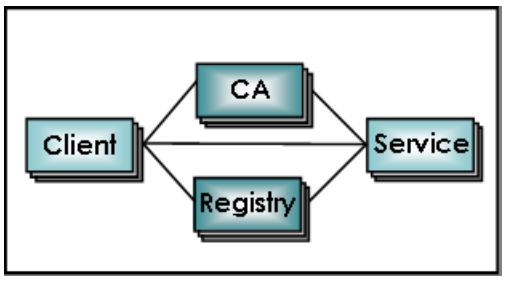



**Abbildung 29: VGE Architektur Überblick. Überarbeitet aus** [Ben]

In einer Grid Umgebung gibt es eine Vielzahl an Clients und Services, die miteinander interagieren. Außerdem gibt es noch eine oder mehrere Registries. Diese können benutzt werden, um Services zu finden. Das Suchen eines geeigneten Services in einer Registry ist optional; wenn ein Client bzw. der Benutzer einen passenden Serviceanbieter kennt, so ist das Benutzen einer Registry nicht erforderlich. Um die Sicherheit im VGE zu gewährleisten sind Zertifizierungsstellen (siehe Certificate Authorities (CA) erforderlich. Dadurch werden einerseits die VGE Entitäten (Clients und Services) eindeutig identifiziert, anderseits wird eine Verschlüsselung der Kommunikation ermöglicht.

Wie bereits erwähnt, basiert das VGE auf einer Service-orientierten Architektur. Ein Service stellt verschiedene Methoden für dessen Nutzung zur Verfügung. Diese Methoden ermöglichen z.B. einen Job auszuführen, den aktuellen Status abzufragen, etc. Sämtliche Interaktionen zwischen Clients (Client-Applikationen oder Aktionen eines Benutzers mittels eines Client-Interfaces) und Services werden vom Client initiiert. Der Client stellt dem Service sämtliche Inputdaten zur Verfügung, er startet den Job und ruft nach dessen Ausführung die Ergebnisse ab.

## 3.2 Service Zugriff Modell

Das VGE ermöglicht die Nutzung von Applikations- und Datenservices – siehe Abbildung 30. Applikationsservices ermöglichen die Nutzung von HPC Anwendungen als Services. Datenservices dienen für den Zugriff auf (verteilte) Datenbeständen in Form von relationalen Datenbanken, XML-Datenbanken, Dateien und Verzeichnissen. Wird in weiterer Folge von einem Job gesprochen, so kann dies ein Job eines Applikationsservices sein, der das Ausführen einer HPC Anwendung zur Folge hat oder ein Job eines Datenservices sein, durch den ein Zugriff bzw. eine Abfrage auf eine Datenressourcen erfolgt.

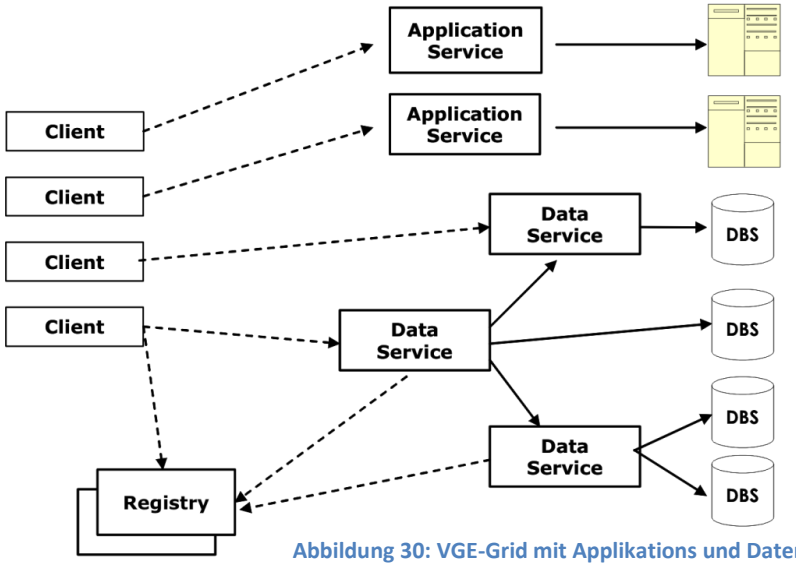



**Abbildung 30: VGE-Grid mit Applikations und Daten Services.**
**Quelle** [Ben08]

### *Registry Service*

In einer VGE Umgebung gibt es i.A. eine oder mehrere Registries. Services werden in Registries registriert und ermöglichen dem Client, ein für seine Anforderungen notwendiges Service zu finden. VGE Registries sind ebenfalls Web Services und verbergen die eigentliche Implementierung (z.B. UDDI) vor dem Benutzer.

### *Certificate Authorities (CAs) & PKI*

Eine Public Key Infrastructure (PKI) basierend auf X.509 Zertifikaten stellt den Clients und Services einen digitalen Identitätsnachweis zur Verfügung. Diese Zertifikate werden von Zertifizierungsstellen (*trusted 3rd parties; CA*) ausgestellt und ermöglichen neben der eindeutigen Identifikation, eine Verschlüsselung auf Transport- und Nachrichtenebene. Auf Transportebene kann durch TSL – Transport layer security – mit https verschlüsselt werden. Auf Nachrichtenebene erfolgt dies durch WS-Security, beispielsweise mit WSS4J[31].

[31] WSS4J ist eine Implentierung der OASIS Web Services Security (WS-Security) Spezifikation, um SOAP Nachrichten zu verschlüsseln und verifizieren.

### *Mehrphasiger Zugriff*

Das VGE wurde entworfen, um einen flexiblen, mehrphasigen Zugriff auf Services zu unterstützen. Dieser Zugriff gliedert sich in folgende Phasen:

1. **Auswahlphase:** In der Auswahlphase erfolgt die Selektion eines entsprechenden Services. Ist kein Service bekannt, wird für die Suche eine Registry verwendet.
2. **Business Phase:** Diese dient zum Aushandeln bzw. Festlegen des Lizenz- und/oder Preismodelles. Das VGE stellt mehrere flexible Preismodelle (Flatrate, pay-per-use, etc.) zur Verfügung; es ist jedoch so konzipiert, dass andere Modelle unterstützt werden. Lizenzmodelle hängen üblicherweise von der Applikation ab.
3. **Quality of Service (QoS) Phase:** In dieser Phase werden diverse Qualitätsparameter ausgehandelt und mittels eines SLAs (Service Level Agreement) festgelegt. Qualitätsparameter sind z.B. die genaue Ausführungszeit und der Preis.
4. **Ausführungsphase:** Zuletzt kommt die Ausführungsphase, in der der Client die Eingabedaten für den Job an das Service sendet, den Job startet und nach dessen Ausführung die zurückgelieferten Ergebnisdaten herunterlädt und auswertet.

In weiterer Folge werden die Service bzw. Client Infrastruktur genauer erläutert.

## 3.3 Service Infrastruktur

Die VGE Service Infrastruktur unterstützt das automatisierte zur Verfügung stellen von Services für den Zugriff auf HPC Appliationen bzw. auf Datenressourcen.

### 3.3.1 VGE Service Komponentenmodell

Ein VGE-Service ist eine Komposition aus Funktionalitäten, welche durch Komponenten realisiert werden. Die Komponenten sind im VGE Komponentenmodell organisiert, welches konzeptionell dem WSRF-Modell folgt. Im VGE Komponentenmodell ist eine Komponente mit genau einer Ressource assoziert.

Das VGE-Komponentenmodell beinhaltet eine Reihe von Komponenten, die unterschiedliche Funktionalitäten zur Verfügung stellen. Folgende Funktionen werden durch VGE Service-Komponenten angeboten:

- Job Management
- Datenzugriff
- Datentransfer zwischen Clients und Services

- QoS Unterstützung
- Verschiedene Sicherheitskonzepte
- Monitoring
- Web-Portale

Das Komponentenmodell bzw. das Provisioning Sub-System der VGE Service Infrastruktur ermöglicht dem Service Provider zum Zeitpunkt des Deployments, die benötigten Service-Komponenten auszuwählen und zu konfigurieren, um ein Service individuell aufzusetzen. Nach außen besitzen alle VGE-Services ein einheitliches WSDL-Interface und folgen dem im Kapitel 3.2 beschriebenen Zugriffsmuster. Deshalb werden alle VGE-Services auch als generische Services bezeichnet.

Service-Komponenten stellen eine abgekapselte Funktionalität zur Verfügung; aufgrund ihrer Funktionalität besitzen die Service-Komponenten eine Schnittstelle nach außen und/oder greifen auf eine Ressource zu.

Im Allgemeinen wird durch die Nutzung generischer Web Service Schnittstellen die Komplexität von Web Service- und Grid-Technologie vor dem Benutzer verborgen.

Zum Anbieten der Services wird Tomcat als Web Server bzw. Servlet Container und das Web Service Framework Axis benutzt, für Datenservices wird zusätzlich intern OGSA-DAI verwendet.

### 3.3.2 Applikationsservices

Applikationsservices bzw. dessen Komponenten ermöglichen eine Virtualisierung von HPC Anwendungen als Grid-Services. Die auszuführende HPC Anwendung wird mit den konfigurierbaren Service-Komponenten assoziert und der Zugriff via VGE-Service ermöglicht. Die generischen VGE-Services gewährleisten ein einheitliches Zugriffsmuster für alle Benutzer, obwohl intern die Anwendungen sehr heterogen sind.

Folgende Schritte sind erforderlich, um Anwendungen als Services anzubieten und zu nutzen:

1) Zuerst erfolgt eine Zusammenstellung bzw. Auswahl der notwendigen Komponenten und deren Konfiguration, damit das Service die gewünschte Funktionalität bietet.
2) Der Service Provider stellt das Service mit den gewünschten Komponenten automatisch zusammen.
3) Das zusammengestellte und installierbare Service wird in der Hosting-Umgebung – einen Web Server bzw. Servlet-Container – deployed.
4) Der Zugriff auf das Service erfolgt durch Standard-Web Service-Technologie analog zum vorgestellten Zugriffsmuster über die generischen VGE-Service Interfaces.

Um bestehende HPC Anwendung als VGE-Service anbieten zu können, müssen folgende Voraussetzungen erfüllt sein:

- Das Starten und Abbrechen der Anwendungen muss durch Skripte möglich sein.
- In der Anwendung dürfen keine absoluten Pfadangaben für Ein-/Ausgabeoperationen sein. Sämtliche Aktivitäten müssen zur relativen Position der Anwendung im Dateisystem erfolgen.

Abbildung 31 zeigt die (möglichen) Service-Komponenten eines Applikationsservices. Durch die Auswahl der entsprechenden Servicekomponenten sind die gewünschten Funktionalitäten nach dem Deployment verfügbar. Optional kann ein Service z.B. QoS Unterstützung anbieten oder ein Web-Portal für dessen Administration beinhalten.

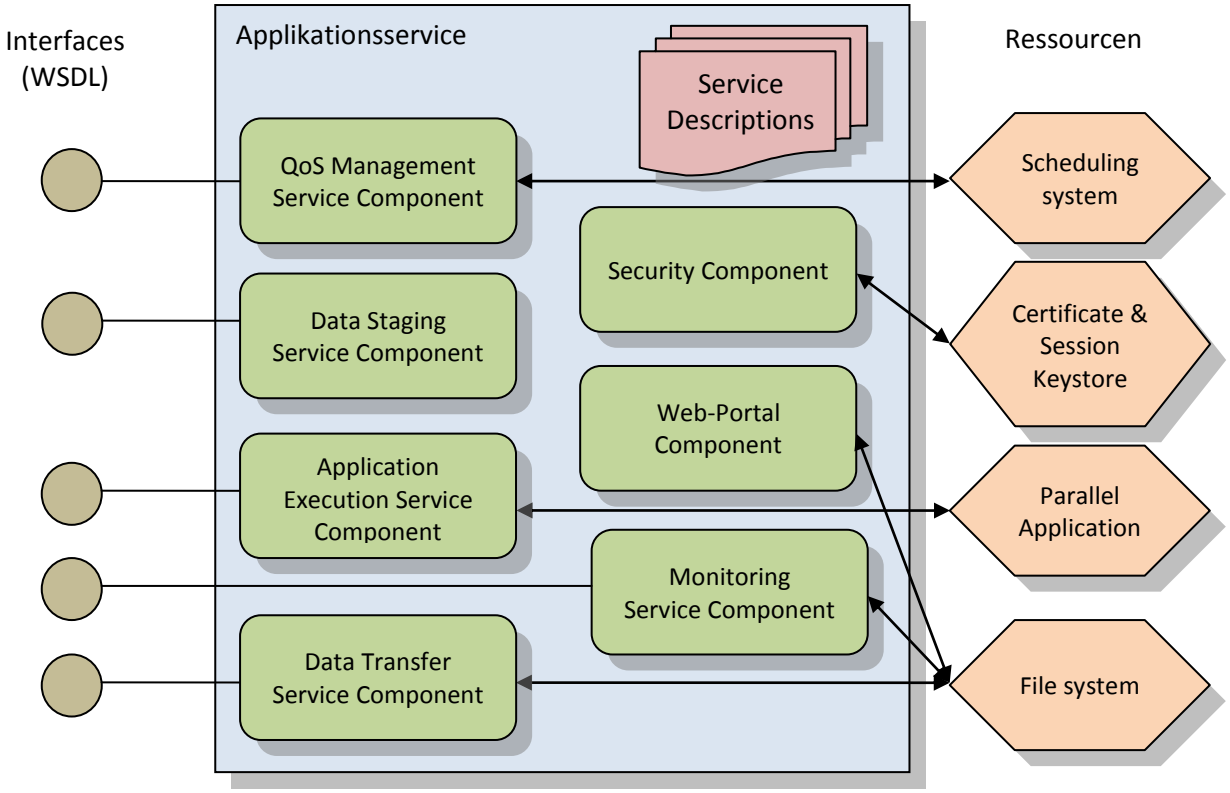



Abbildung 31: VGE Application Service

### *Application Execution Service-Komponente*

Die *Application Execution Service*-Komponente wird für das Ausführen der HPC Anwendung verwendet und ermöglicht das Starten eines Jobs mit der HPC Applikation, dessen Abbrechen bzw. die Statusabfrage laufender oder bereits beendeter Jobs.

Service-seitig werden für diese Operationen Skripts vom Serviceanbieter bzw. der Administration zur Verfügung gestellt und beim Aufruf der entsprechenden Operation ausgeführt. Skripte für die Statusabfrage und den Abbruch eines Jobs sind optional und hängen von der Art der Anwendung ab.

Für jedes Service muss ein eigenes Arbeitsverzeichnis angegeben werden. In einem Referenzverzeichnis werden die entsprechenden Skripte gespeichert. Wird nun ein neuer Job initiiert, so wird innerhalb dieses Arbeitsverzeichnisses ein neues Verzeichnis mit einem eindeutigen Namen für diesen Job angelegt; die notwendigen Skripte werden vom Referenzverzeichnis kopiert und im Arbeitsverzeichnis ausgeführt.

### *Data Transfer Service-Komponente*

Die *Data Transfer Service*-Komponente ermöglicht das Übertragen von Eingabe- bzw. Ergebnisdaten vor bzw. nach dem Ausführen der Anwendung zwischen Client und Service durch die entsprechenden Operationen *upload* und *download*.

Normalerweise werden die Eingabe- bzw. die Ergebnisdaten in Form eines ZIP-Archives übertragen. Eine Übertragung erfolgt ins bzw. vom jeweiligen Jobverzeichnis; die Clients haben somit keinen Zugriff auf andere Bereiche des Dateisystems.

### *QoS Management Service-Komponente*

Das VGE-QoS-Modell folgt einem Business Grid-Ansatz, wobei Clients und Service Provider QoS-Bedingungen in Bezug auf die genaue Ausführungszeit bzw. den Preis aushandeln können und ein entsprechendes Abkommen (SLA) treffen. Clients sind i.A. bereit, sofern ihre QoS Bedingungen eingehalten werden, einen Preis zu bezahlen. Service Provider sind bereit für Ressourcenverfügbarkeit zu garantieren, sofern sie Geld dafür erhalten. Die QoS Management Komponente bietet Operationen, welche ein entsprechendes QoS-Verhandlungsprotokoll realisieren.

### *Data Staging Service-Komponente*

Daten werden nicht nur zwischen Client und Service übertragen, sondern auch zwischen Services (z.B. im Falle der Benutzung eines Services durch ein anderes). Der wesentliche Unterschied zur Data Transfer-Komponente besteht darin, dass einem Service A das Recht für den Datenzugriff auf ein Service B im Namen bzw. Auftrag eines Clients erteilt wird. Dieses Übertragen von Rechten wird auch als Delegation bezeichnet.

Der Datentransfer zwischen Services wird durch die Operationen *push* und *pull* realisiert. Intern wird die *Data Transfer*-Komponente verwendet; somit ist nur der eingeschränkte Zugriff auf die jeweiligen Jobverzeichnisse möglich.

Alle weiteren Service-Komponenten werden in diesem Zusammenhang nicht näher erläutert.

### 3.3.3 Datenservices

Datenservices ermöglichen die Virtualisierung von Datenressourcen als Grid-Services. Eine Datenressource ist i.A. ein Sammelbegriff, welcher von einzelnen Dateien bzw. einem Verzeichnis über relationale und XML-Datenbanken bis zu einer „Virtual DB“[32] beinhalten kann. Datenservices ermöglichen somit die Nutzung von heterogenen (verteilten) Datenressourcen über Grid Services.

Der Zugriff erfolgt wie bei den Applikationsservices über generische Service Schnittstellen und ermöglicht dadurch ein einheitliches Zugriffsmuster sowohl für Daten- als auch Applikationsservices.

Intern wird der Zugriff auf die Datenbestände durch OGSA-DAI, OGSA-DQP[33] und GDMS (siehe „2.3.3 OGSA-DAI Software“) realisiert.

In Abbildung 32 ist der Komponentenaufbau für Datenservices ersichtlich:

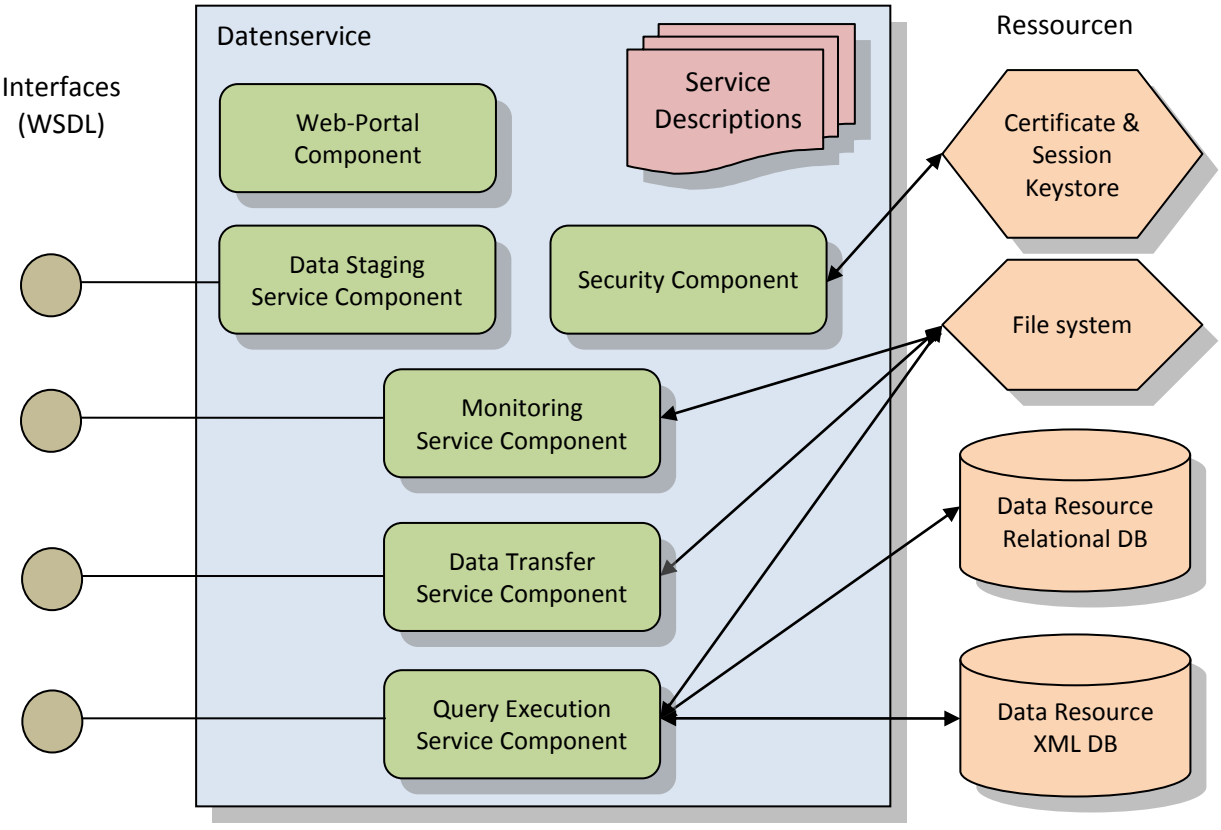



Abbildung 32: VGE Data Service

[32] Eine virtuelle Datenbank ist ein Zusammenschluß mehrerer Datenbank, realisiert durch ein globales Schema kombiniert aus den Schemata der einzelnen Datenbanken.
[33] OGSA Distributed Query Processing (OGSA-DQP) ist eine Erweiterung zu OGSA-DAI.

### *Query Execution Service-Komponente*

Die *Query Execution Service*-Komponente ist das Äquivalent zur *Application Execution Service*-Komponente und dient zur Ausführung von Aktionen auf Datenressourcen.

Die Eingabedaten eines Datenservice entsprechen einem OGSA-DAI Perform-Dokument. Das Ergebnis eines Datenservices ist ein OGSA-DAI Response-Dokument. Das Perform-Dokument beinhaltet die gewünschte SQL-Aktion wie z.B. die Datenabfrage mit dem *Select*-Kommando oder das Einfügen von Daten mit dem *Insert*-Kommando. Zum Erstellen des Perform-Dokumentes bzw. zur Auswertung des Response-Dokumentes wird von OGSA-DAI ein Client Framework zur Verfügung gestellt, das vom VGE Client benutzt werden kann. Intern wird OGSA-DAI, OGSA-DQP bzw. GDMS für den Datenzugriff verwendet.

### *Data Transfer Service-Komponente*

Die *Data Transfer Service*-Komponente ermöglicht, wie auch bei den Applikationsservices, das Übertragen von Daten zwischen Client und Service. Dies kann z.B. ein Perform-Dokument zur Ausführung einer Datenressource-Aktion bzw. ein Response-Dokument mit den Ergebnissen sein. Es können aber auch binäre Daten in Form von Dateien übertragen werden, die z.B. als Blobs in der Datenbank gespeichert werden.

Andere Komponenten wie z.B. die *Data Staging* Service-Komponente bieten die gleiche Funktionalität wie in Applikationsservices bzw. werden nicht genauer behandelt.

## 3.4 Client Infrastruktur

Die Client-Infrastruktur dient als Werkzeug, um auf einfache Art und Weise Client-Applikationen zu realisieren, die verschiedene Grid-Services verwenden können. Unterstützt wird sowohl die Entwicklung komplexer Applikationen, als auch die Integration in bestehende Applikationen durch Client-Realisierungen in vielen Programmiersprachen.

Die Client-Infrastruktur besteht aus folgenden Bestandteilen:

- Java-Bibliotheken
- Dokumentation
- Beispiele
- Execution- bzw. Beispiel-Entwicklungsumgebung

Die Client-Infrastruktur bietet folgende Funktionalitäten bzw. APIs:

- Auffinden bzw. Auswählen von Services
- Job Management (Starten von Jobs, Unterbrechung der Nutzung, Abbruch, Status- bzw. Ergebnisabfrage)
- QoS: Bestimmung und Service Auswahl auf Grund diverser Qualitätskriterien
- Sicherheit

Client-Komponenten:

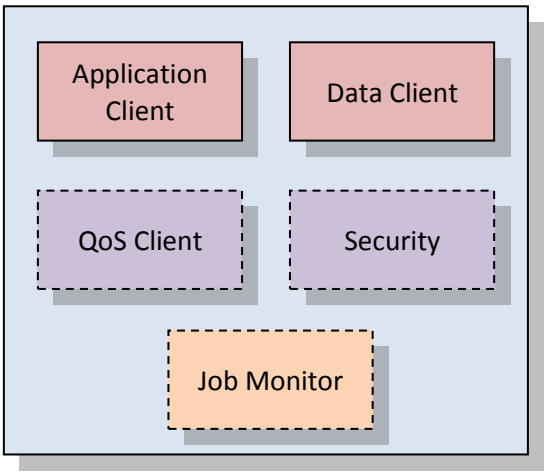



Abbildung 33: Client-Komponenten

Ein VGE-Client ist analog zu den Services aus Komponenten aufgebaut und kann aufgrund der Komponentenbauweise individuell zusammengestellt werden. In Abbildung 33 sind die verschieden Client-Komponenten dargestellt.

Eine Client-Komponente ist die Gruppierung und Zusammenfasssung von Funktionalität analog einer Service-Komponente. Dadurch ist die Kommunikation mit der Service-Komponente bzw. die Nutzung der Funktionalität der Service-Komponente möglich.

Um die Funktion einer Service-Komponente zu nutzten, umfasst eine Client-Komponente:

- sämtliche Klassen & Bibliotheken des entsprechenden Client-APIs, die für die Nutzung einer Service-Komponente notwendig sind
- Konfigurationsdaten/–dateien, die dafür notwendig sind

Für die Entwicklung von Grid-Client-Applikationen wird in der VGE Client Infrastruktur ein high-level API in Java sowie in anderen Plattformen wie •NET, C# oder C zur Verfügung gestellt. Weiters gibt es eine Client Unterstützung für Shell Scripts, sowie einen Web-Client, der auf einfache Weise das Starten & Abbrechen von Jobs, Statusabfragen und den Transfer von Eingabe- und Ergebnisdateien ermöglicht.

Die abstrakten Schichten des Clients APIs werden in Abbildung 34 dargestellt. Höhere Schichten nutzen Methoden niedrigerer Schichten um Funktionalitäten zu realisieren. Die Komplexität der darunterliegenden Schichten wird dadurch verborgen. Höhere Schichten bringen ein höheres Abstraktionsniveau, haben aber auch weniger Einfluß auf das Verhalten. Listing 5 enthält ein Beispiel, in dem ein Applikationsservice durch die Verwendung des *ApplicationProxy* benutzt wird.

| |
|---|
| **Agent Layer**<br>Application-Agent, Data-Agent, QoS-Agent |
| **ServiceProxy Layer**<br>Application-Proxy, QoS-Proxy, Streaming-Proxy<br>ermöglichen<br>Application/Query Exection, QoS, Data Transfer, Data Staging, Recovery, Monitoring |
| **Stub Layer**<br>Application-Stub, QoS-Stub, Streaming-Stub |
| Message Layer + WS-Security |
| Transport Layer + TLS |

**Abbildung 34: Abstrakte Schichten des Client APIs. Quelle** [Ben08]

Komponenten haben APIs auf verschiedenen Schichten. Die QoS Client-Komponente ermöglicht QoS Unterstützung auf ServiceProxy-Schicht durch den QoS-Proxy, als auch auf Stub-Schicht durch den QoS-Stub. Alle Stubs der Stub-Schichten sind wiederum im entsprechenden API gebündelt. Das Stub-API beinhaltet einen Application-Stub, QoS-Stub, etc. Tabelle 3 enthält eine Auflistung der Client-APIs.

| | |
|---|---|
| AGENT | High-Level Agent API für Application und QoS Clients |
| DISCOVERY | Discovery-API für das Auffinden von geeigneten Services |
| MONITORING | Client-seitiges Monitoring von Ereignissen |
| PROXY | Proxy API beinhaltet einen Application, QoS und Streaming[34] Proxy |
| SECURITY | Security API |
| STUB | Low Level Stub API zur Kommunikation mit den Service-Komponenten |

Weitere Pakete:

| | |
|---|---|
| CMD | Command Line Client (Interpreter in Java) |
| SERVLET | Web-Client Klassen; beinhaltet einen Application, Data & QoS Client |

**Tabelle 3: Client-APIs**

Listing 5 beinhaltet ein Beispiel für einen VGE Client Code in Java, der ein Applikationsservice nutzt. In diesem Beispiel wird ein *AppProxy* initialisiert, anschließend wird durch diesen die Eingabedatei „upload.dat“ zum Service hochgeladen. Danach wird die Applikation gestartet und auf das Ergebnis gewartet. Zuletzt wird die Ergebnisdatei heruntergeladen.

[34] Als Streaming bezeichnet man das kontinuierliche Übertragen von Daten, auch benannt als Übertragung von Datenströmen. Die Datenmenge bzw. deren Größe ist am Beginn der Übertragung nicht bekannt.

Durch die Nutzung des *ApplicationProxy* wird die Komplexität der darunterliegenden Schichten verborgen. Der *ApplicationProxy* ermöglicht es auf einfache Weise, Eingabe- und Ergebnisdaten zu transferieren, Jobs zu starten und zu überwachen, etc.

```
// creates new proxy based on properties
Properties props = new Properties();
props.setProperty("client.job.uri","http://pia.par.univie.ac.at:9090/HelloWorld/scs/");
props.setProperty("client.base.proxy.exception", "true");
AppProxy someProxy = new AppProxyImpl(props);

// obtain statusinformation on the proxy object itself
System.out.println("----->  ProxyStatus: " + someProxy.getProxyStatus());

// upload a local file to the remote native application using the proxy object
someProxy.upload(new File("upload.dat"), "inputABC");

// start the remotely deployed and configured, native, application
someProxy.start();

AppProxyState proxyStatus = AppProxyState.ERROR;
String appStatus = null;
while (proxyStatus != AppProxyState.FINISHED) {
   // obtain statusinformation on the execution of the started job
   appStatus = someProxy.getApplicationStatus();
   System.out.println("----->  Statusinformation: " + appStatus);

   // print out proxy object status
   proxyStatus = someProxy.getProxyStatus();
   System.out.println("----->  ProxyStatus: " + proxyStatus);
}

// create filehandler for downloading the result of the executed application
DataHandler data = (DataHandler) someProxy.download("output.dat");
File outFile = new File("output.dat");
FileOutputStream out = new FileOutputStream(outFile);

data.writeTo(out);
```

**Listing 5: VGE Client Code Beispiel für die Nutzung eines Applikationsservices**

Die Implementierung eines Dataenservice-Clients ist analog der eines Clients für Applikationsservices in Listing 5.

Services und Web-Clients werden durch den, im nächsten Kapitel beschriebenen, Provisioning-Prozess zur Verfügung gestellt.

## 3.5 Provisioning

Das Provisioning wird im Kontext von Grid-Computing als Prozess des zur Verfügungstellens eines Grid-Services verstanden. Nach dem Provisioning ist ein Service zur Interaktion mit den Benutzern bereit.

Grid-Services sind komplex, weil sie aus verschiedenen Komponenten zusammengesetzt sind. Der Provisioning-Prozess verbirgt diese Komplexität vor dem Service-Provider. Die Funktionalität eines Services oder Clients ist durch dessen Komponenten bestimmt.

### *Client & Service Komponenten*

Abbildung 35 verdeutlicht den Komponentenaufbau und zeigt Service- und Client-Komponenten, die eine individuelle Kombination ermöglichen bzw. die aufgrund ihrer Auswahl ein Service oder einen Client mit der gewünschten Funktionalität anbieten.

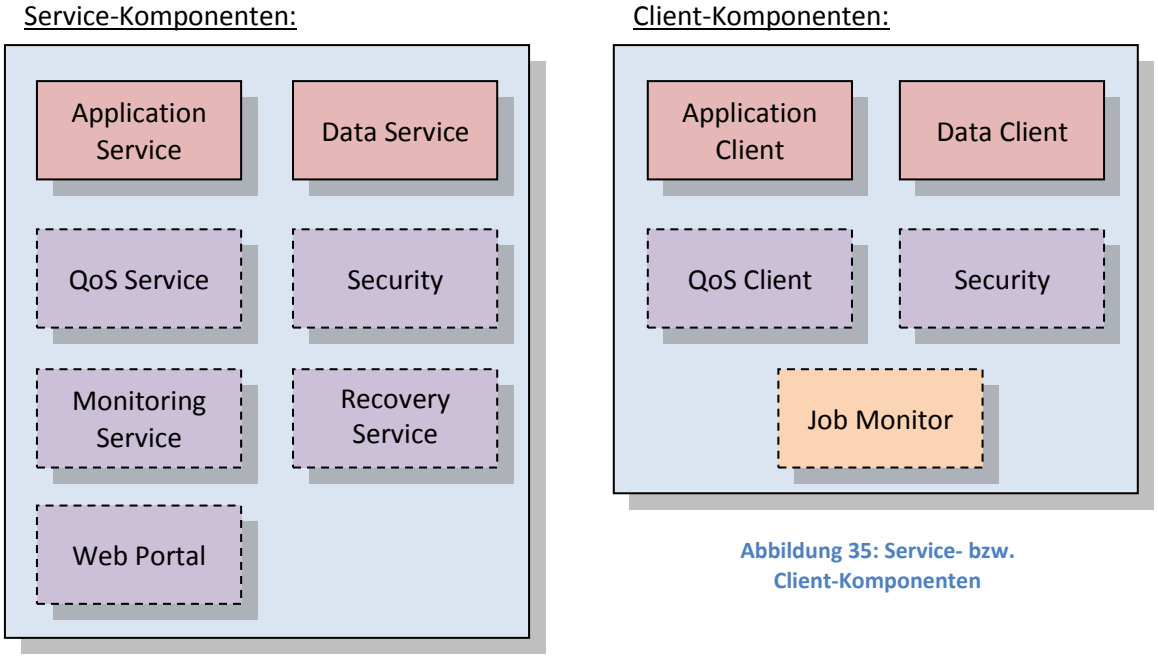



**Abbildung 35: Service- bzw. Client-Komponenten**

Eine Service-Komponente umfasst sämtliche Daten und Dateien, um eine bestimmte Funktion zu ermöglichen und könnte z.B. folgendes umfassen:

- sämtliche Klassen, Bibliotheken und Dateien, die für die Funktionsbereitstellung einer Service-Komponente notwendig sind
- Konfigurationsdaten/–dateien, die für diese Service-Komponente notwendig sind

Eine Client-Komponente hingegen umfasst sämtliche Daten und Dateien, um die Funktion einer Service-Komponente zu nutzen:

- Sämtliche Klassen, Bibliotheken und Dateien, die für die Nutzung einer Service-Komponente notwendig sind.
- Konfigurationsdaten/–dateien, die dafür notwendig sind

Eine Service-Komponente kann auch ein Interface in Form eines Web-Portals sein und enthält somit sämtliche HTML-Seiten, Grafiken, etc.

### *Client-Service-Interaktion*

Service- und Client-Komponenten können modular verwendet werden, d.h. nicht immer müssen alle Komponenten verwendet werden. Um eine bestimmte Funktionalität z.B. QoS zu nutzen, muss die entsprechende Komponente Service- und Client-seitig vorhanden sein. Es kann aber auch ein Client ohne QoS ein Service mit QoS-Komponente verwenden, allerdings kann die QoS Unterstützung dann nicht genutzt werden.

Abbildung 36 zeigt verschiedene Clients und Services mit jeweils unterschiedlichen Komponenten. Service A bietet Security und ein Web-Portal, Service B bietet QoS und ebenfalls Security. Für Client 1 ist beispielsweise QoS wichtig und wird bei der Nutzung von Service B unterstützt. Bei Client 2 soll die Sicherheit gewährleistet sein, QoS ist hingegen nicht von Bedeutung.

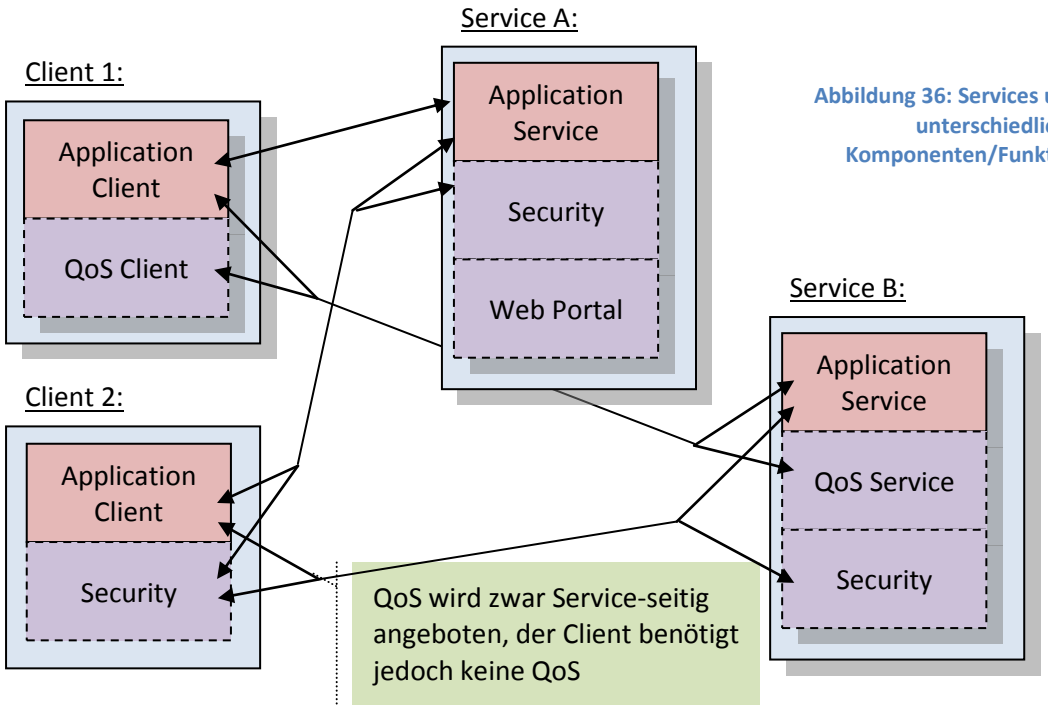



**Abbildung 36: Services und Clients mit unterschiedlichen Komponenten/Funktionalitäten**

### *Service-Provisioning*

Das Service-Provisioning ist der Prozess von der Auswahl und Konfiguration der Service-Komponenten bis zum laufenden Web Service. Client-seitig wird nur das Provisioning des Web-Clients unterstützt, daher bezieht sich dieses Kapitel mehrheitlich auf das Bereitstellen von Services.

Einen zentralen Überblick über den Provisioning Prozess bietet die Abbildung 37, welche im ganzen Kapitel immer wieder referenziert wird.

Der Provisioning Prozess teilt sich in folgende Abschnitte:

1) Individuelle Service-Konfiguration:
   a) Service-Komponenten-Auswahl und deren Konfiguration:
   b) Service-Hosting & zusätzliche Konfigurationen
2) Automatisches Deployment:
   a) Provisioning zur Erzeugung des Web Application Archives (WAR-Datei)
   b) Deployment des Services & Initialisierung

Der Service Provider wird während des gesamten Provisioning-Prozesses – von der Auswahl der Service-Komponenten bis zum laufenden Service - durch das grafische Deployment-Tool geleitet bzw. unterstützt, welches Teil des Provisioning-Environments ist. Das Provisioning-Environment ermöglicht eine automatisierte Zusammenstellung des Services bis zum Deployment des Services im Hosting-Environment.

Im ersten Teil des Service-Provisioning-Prozess - der individuelllen Service-Konfiguration - werden entsprechende Service-Komponenten aufgrund ihrer Funktionalität ausgewählt und die dafür notwendigen Konfigurationsparameter zur Individualisierung der Service-Komponenten eingegeben.

Der nächste Schritt betrifft das Service-Hosting, in dem Parameter der Hosting-Umgebung des Services eingegeben werden. Das Deployment-Tool generiert aus den Daten beider Schritte die Service Description. Eine Service Description besteht aus mehreren Dateien: einen Application bzw. Data Descriptor sowie ein oder mehrere Service Properties-Dateien.

Im zweiten Teil des Service-Provisioning-Prozesses – dem automatisierten Deployment – findet zuerst das eigentliche Provisioning statt. Es wird ein installierbares Web Application Archive (WAR-Datei) erzeugt. Dazu wird eine der Servlet-Spezifikation entsprechende Verzeichnisstruktur angelegt, Konfigurationsdateien wie z.B. der Web-Application-Deployment-Deskriptor generiert, interne und externe Bibliotheken in die Verzeichnisstruktur kopiert, etc. Dieser Prozess ist automatisiert durch ANT-Skripte und erstellt das installierbare Web Application Archive (WAR-Datei). Als nächstes erfolgt das Deployment des Web Services im Servlet-Container mit der erzeugten WAR-Datei, ebenfalls automatisiert durch ANT-Skripte. Zuletzt erfolgt eine Initialisierung des Services. Der Service-Provisioning-Prozess ist abgeschlossen und resultiert in einem laufenden, initialisierten VGE-Service.

Die Abbildung 37 illustriert den zuvor beschriebenen Service-Provisioning-Prozess.

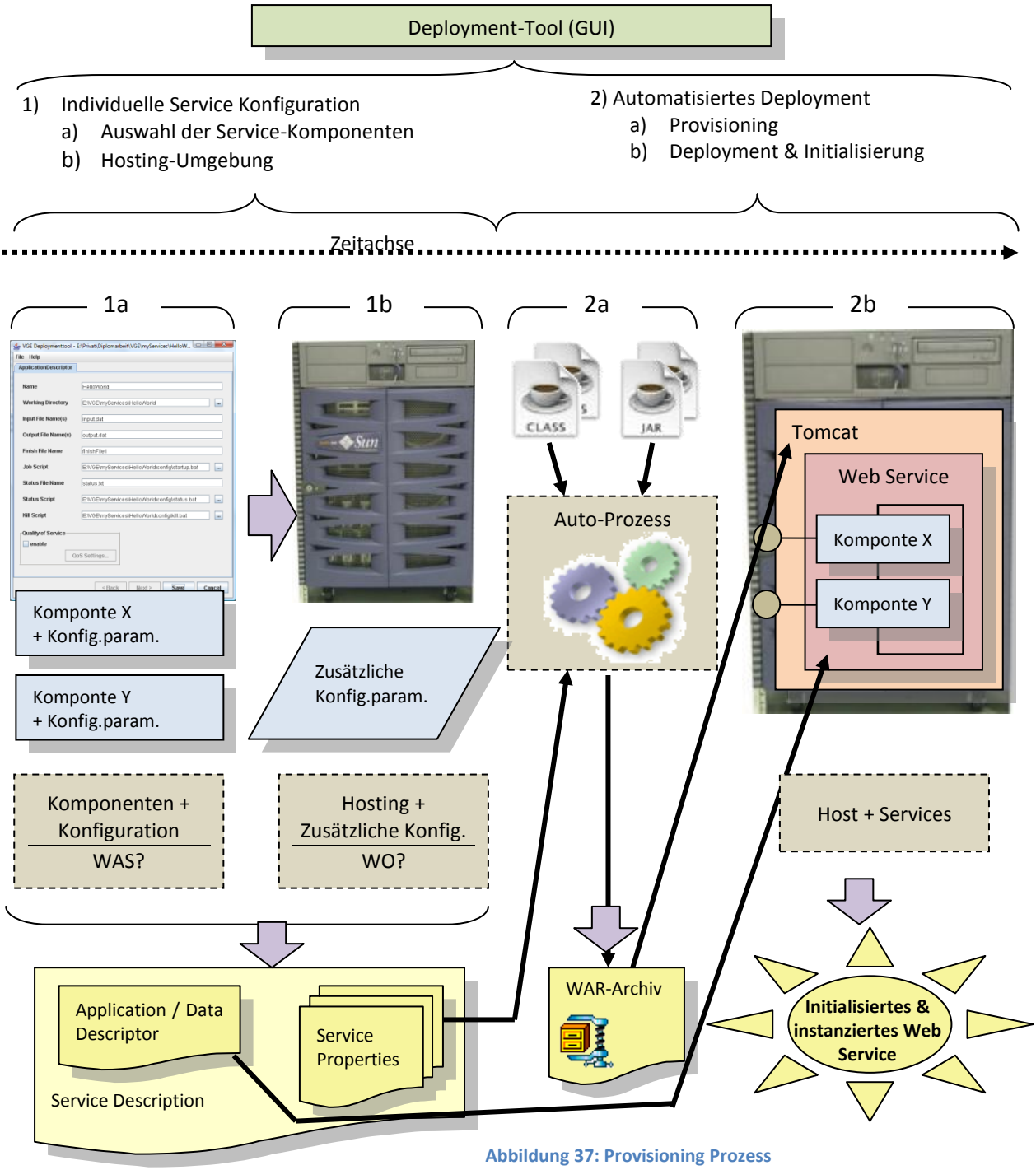



Abbildung 37: Provisioning Prozess

Im nächsten Kapitel wird die individuelle Service-Konfiguration – der linke Abschnitt in der Abbildung 37 – beschrieben.

### 3.5.1 Individuelle Service-Konfiguration

Die individuelle Service-Konfiguration ist im linken Teil der Abbildung 37 dargestellt und teilt sich in die Service-Komponentenauswahl und deren Konfiguration bzw. in das Service-Hosting und der Eingabe zusätzlicher Konfigurationsparameter.

Das Ergebnis der individuellen Service-Konfiguration ist eine vollständige Beschreibung (Description), welche aus Application- bzw. Data-Descriptor und Service Properties besteht.

Die minimale Ausprägung eines Service ist durch folgende Konstellation gegeben:

| | | | | |
|---|---|---|---|---|
| *Application Execution*-Service Komponente<br>*Query Excecution*-Service Komponente | + | *Data Transfer*-<br>Service Komponente | =<br>= | Applikationsservice<br>Datenservice |

Da ein Service zumindest diese Komponenten beinhaltet, ermöglicht die Service-Komponentenauswahl die Selektion zusätzlicher, optionaler Komponenten.

### *Application Service*

Sämtliche Daten für die Parametrisierung der generischen Komponenten müssen festgelegt werden, um eine HPC-Applikation als VGE-Applikationsservice anbieten zu können. Mit dem grafischen Deployment-Tool für ein Applikationsservice können alle notwendigen Informationen bequem eingegeben werden. Folgende Daten sind erforderlich:

- Service-Name
- Service-Konfiguration bzw. Parameter für die *Application Service*-Komponente
  - Arbeitsverzeichnis
  - Skripte:
    - zum Starten der Applikation
    - optional ein Skript zur Statusabfrage
    - optional ein Skript zum Abbrechen der Applikation
  - Name oder Pattern der Eingabedatei(en)
  - Name oder Pattern der Ausgabedatei

→ Application Descriptor

- Service-Hosting & zusätzliche Service-Komponenten
  - Optionale, zusätzliche Service-Komponenten (QoS, Security, Streaming, Monitoring, Web-Portal)
  - Angaben zur Hosting-Umgebung in der das Service installiert werden soll, wie
    - Service-URI des Web Servers
    - Account-Daten für die Benutzung des Application Managers von Tomcat

→ Service Properties

Das Ergebnis der Eingabe aller Daten im Deployment-Tool ist eine vollständige Service Beschreibung (Description) in Form eines XML Application Descriptors sowie der Service Properties. Der Application Descriptor wird im Wesentlichen zur Laufzeit benötigt und enthält Informationen zur Applikation. Die Service Properties werden hauptsächlich für den ANT-basierten Deployment-Prozess benötigt.

Abbildung 38 und Abbildung 39 zeigen das Deployment-Tool, Listing 6 beinhaltet den daraus erzeugten Service-Deskriptor und Listing 7 die Service Properties-Datei.

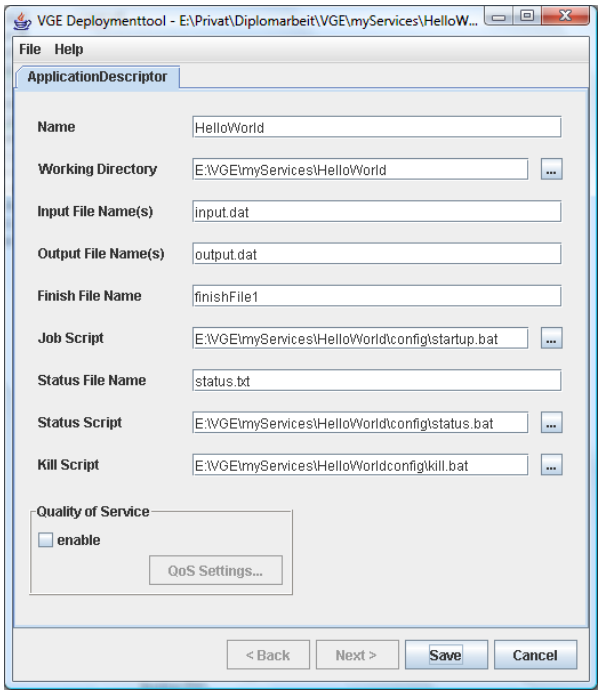



Abbildung 38: Service-Konfiguration

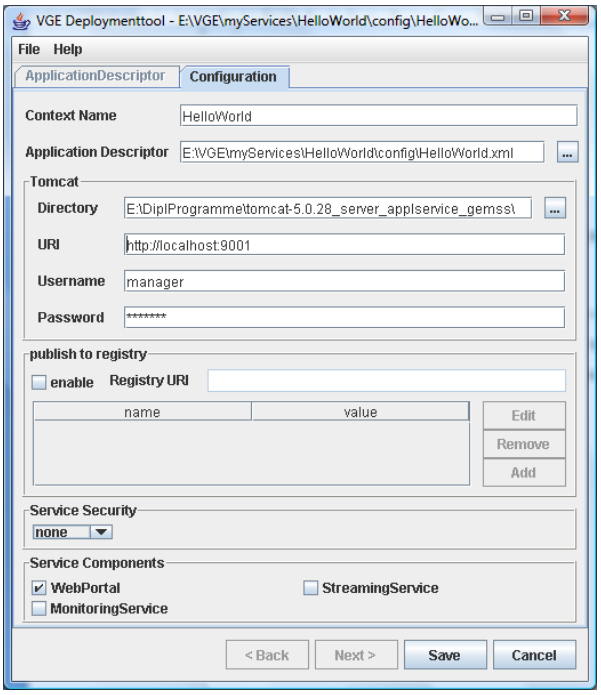



Abbildung 39: Hosting-Umgebung und zusätzliche Service-Komponenten

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<application>
  <info>
    <name>HelloWorld</name>
  </info>
  <configuration>
    <working-directory>
      <path>E:\\VGE\\myServices\\HelloWorldWindows</path>
    </working-directory>
    <input-files>
      <file>
        <name>input.dat</name>
      </file>
    </input-files>
    <output-files>
      <file>
        <name>output.dat</name>
      </file>
    </output-files>
    <job-script-parameters>
      <job-script-parameter>
        <name>debug</name>
      </job-script-parameter>
    </job-script-parameters>
    <job-script>
      <path>E:\\VGE\\myServices\\HelloWorld\\config\\startup.bat</path>
    </job-script>
    <finish-file>
      <name>finishFile1</name>
    </finish-file>
    <status-info>
      <status-script>
        <path>E:\\VGE\\myServices\\HelloWorld\\config\\status.bat</path>
      </status-script>
      <status-file>
        <name>status.txt</name>
      </status-file>
    </status-info>
    <kill-script>
      <path>E:\\VGE\\myServices\\HelloWorld\\config\\kill.bat</path>
    </kill-script>
  </configuration>
</application>
```

Listing 6: Service-Deskriptor für ein Applikationsservice

```
1  #example service
2  service.name=HelloWorld
3  service.application.descriptor=E:\\VGE\\myServices\\HelloWorld\\config\\HelloWorld.xml
   service.config.dir=E:/VGE/myServices/HelloWorld/config
4  service.tomcat.dir=E:\\DiplProgramme\\tomcat-5.0.28_server_applservice_gemss
5  service.protocol=http
6  service.host=localhost
7  service.port=9001
8  service.manager.user=manager
9  service.manager.pwd=manager
10 service.remote.dist=
11 service.components=ApplicationService,WebPortal
12
```

**Listing 7: Service Properties**

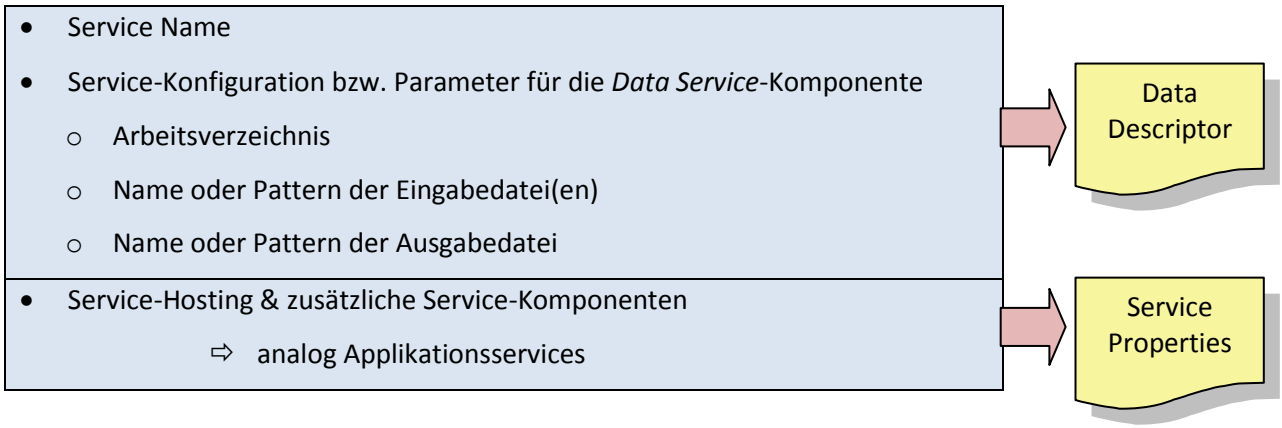
Im nächsten Abschnitt wird die individuelle Service-Konfiguration für Datenservices bzw. die Unterschiede in Bezug auf Applikationsservices beschrieben.

### *Data Service*

Im Zuge der individellen Konfiguration eines Datenservices werden folgende Angaben mit dem Deployment-Tool erhoben, um eine Datenressource als Service anzubieten:

a) Selektion und Konfiguration der gewünschten Service-Komponenten (analog zu einem Applikationsservice)
b) Parameter der Hosting-Umgebung (analog zu einem Applikationsservice)
c) Angaben zu den verwendeten Datenressourcen

Folgende Daten sind zur Parametrisierung der Komponenten erforderlich:



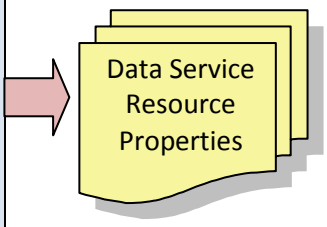
Für jede verwendete Datenressource müssen vom Serviceanbieter folgende spezifischen Zugriffsdaten angegeben werden:

- Datenressource-Name
- Datenressource-Produkt:
  - entweder eine relationale Datenbank wie MySQL, Oracle, etc.
  - oder eine XML-Datenbank wie eXist, etc.
  - oder der Zugriff auf Dateien und Verzeichnisse

  Das auszuwählende Datenressource-Produkt entspricht den von OGSA-DAI unterstützten Datenressourcen (aufgelistet im Kapitel „OGSA-DAI Data Resources“).
- Datenressource-URI

  Wird auf eine MySQL-Datenbank zugegriffen, dann wäre die URI z.B.

  ```
  jdbc:mysql://localhost:3306/vgds_jobmanager
  ```
- Zugriffsdetails (Access-Credentials)
- Weitere Angaben wie Hersteller des Produktes, Versionsnummer



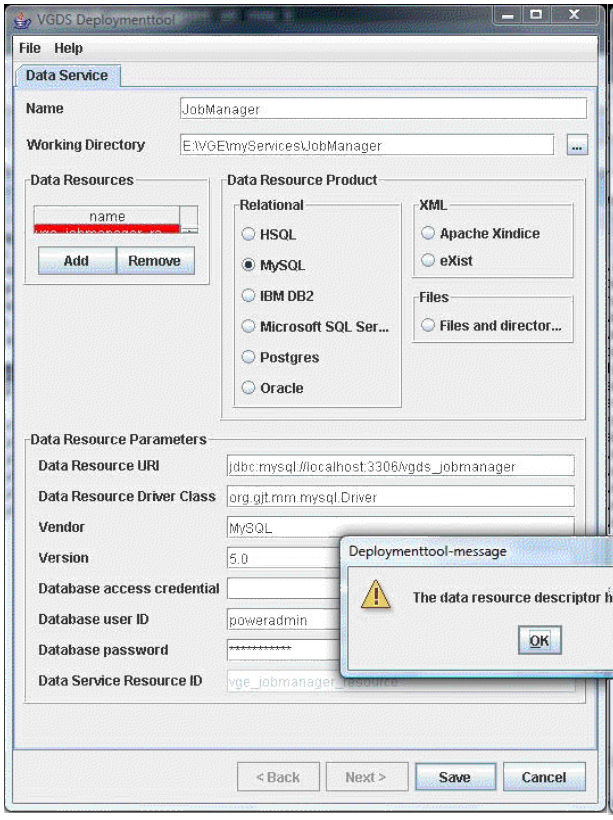




Abbildung 40: Konfigurationsdaten der Datenressourcen

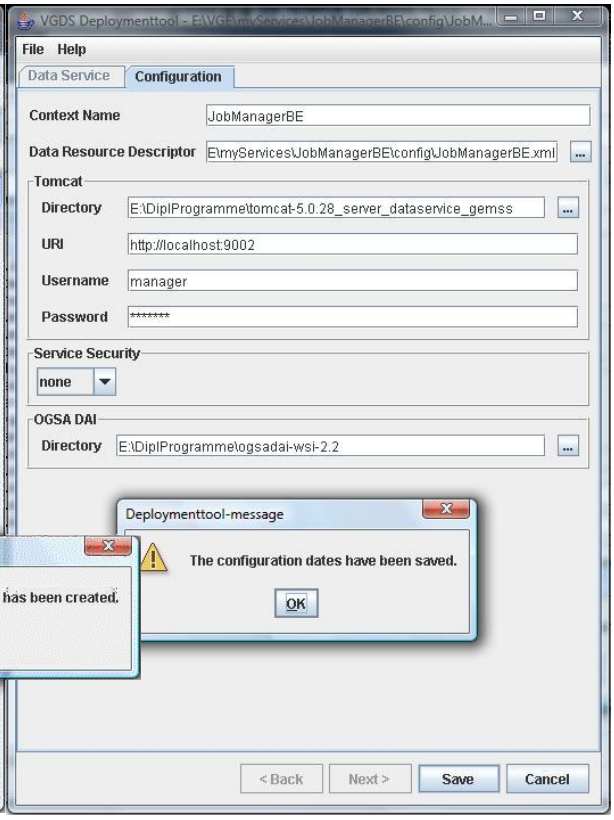



Abbildung 41: Hosting-Umgebung und zusätzliche Service-Komponenten

Abbildung 40 und Abbildung 41 zeigen das Deployment Tool mit sämtlichen Angaben zu einer MySQL-Datenbank als Datenressource. Das Ergebnis des Deployment-Tools ist ein Data Service-Deskriptor und eine Service Properties-Datei analog zu einem Applikationsservice.

Zusätzlich wird für jede Datenressource eine *Data Service Resource*-Datei mit den eingegebenen Daten erzeugt. Das Resultat, der in Abbildung 40 eingegeben Datenressource, ist in Listing 8 abgebildet. Diese *Data Service Resource*-Dateien werden im Zuge des OGSA-DAI Deployments benötigt, da für jede benötigte Datenressource der Zugriff mittels OGSA-DAI eingerichtet wird.

```
1  #data service resource
2  dai.resource.id=vge_jobmanager_resource
3  dai.data.resource.type=Relational
4  dai.product.name=MySQL
5  dai.product.vendor=MySQL
6  dai.product.version=5.0
7  dai.data.resource.uri=jdbc:mysql://localhost:3306/vgds_jobmanager
8  dai.driver.class=org.gjt.mm.mysql.Driver
9  dai.credential=
10 dai.user.name=poweradmin
11 dai.password=poweradmin!
12
13 dai.driver.jars=
14 dai.data.service.resource.uri.one=
15 dai.data.service.resource.id.one=
16 dai.data.service.resource.description.one=
17 dai.data.service.resource.uri.two=
18 dai.data.service.resource.id.two=
19 dai.data.service.resource.description.two=
```

Listing 8: Data Service Resource-Datei

Wenn sämtliche Daten mit dem Deployment-Tool eingegeben wurden und somit die individuelle Service-Konfiguration abgeschlossen ist, kann der automatische Deployment-Prozess mit der Generierung der WAR-Datei beginnen.

### 3.5.2 Automatischer Deployment-Prozess

Der automatische Deployment-Prozess ist im rechten Teil der Abbildung 37 dargestellt und benötigt die, durch die individuelle Service Konfiguration, erhobenen Daten.

Der automatische Deployment-Prozess besteht aus zwei Phasen:

1. Provisioning zur Erzeugung des Web Application Archives (WAR-Datei)
2. Deployment des Services & Initialisierung

Das Provisioning eines Services erfolgt automatisiert durch ANT-Skripte. Die dazu benötigten Eingabedaten werden über entsprechende Properties-Dateien bezogen, welche vom Deployment-Tool am Ende der individuellen Konfiguration (in Kapitel 3.5.1) erzeugt wurden.

Das Ergebnis des Provisioning-Prozesses ist eine WAR-Datei, die das Service als vollständige Webanwendung mit sämtlichen benötigten Dateien und Bibliotheken nach der Java-Servlet-Spezifikation **[Mor08]** enthält.

### *Applikationsservice*

Der automatische Deployment-Prozes eines Applikationsservices besteht aus folgenden Schritten:

1. Erzeugen des *Web-Application-Deployment*-Deskriptors *(/WEB-INF/web.xml*)
2. Erzeugen des Deployment Deskriptors für Apache Axis (*client*- bzw. *server-config.wsdd*)
3. Erzeugen der WSDL-Dateien. Üblicherweise wird für jede Komponente eine entsprechende WSDL-Datei erzeugt.[35]
4. Erzeugen interner Properties-Dateien, die während der Laufzeit benötigt werden und Parameter der Service-Umgebung beinhalten
5. Laden sämtlicher benötigter Bibliotheken. Dies sind Bibliotheken der VGE Software, wie auch Bibliotheken von externer Software.
6. Archivierung sämtlicher Dateien zu einem *Web Application Archive* (WAR-Datei) gemäß der Java-Servlet-Spezifikation. Dieses Archiv beinhaltet die *web.xml*-Datei, *config.wsdd*-Datei, wsdl-Dateien, Properties-Dateien, VGE-Bibliotheken, externe Bibliotheken, usw.
7. Installation des Web Service auf dem Tomcat-Server mit Hilfe der Tomcat Manager Applikation – aus Web Server-Sicht ist dies der „eigentliche“ Deployment-Prozess.
8. Initialisierung/Instanzierung des Services[36]

### *Datenservice*

Der automatische Deployment-Prozess im Zuge des Einrichtens eines Datenservices ist ähnlich dem eines Applikationsservices. Es wird ein Web Application Archive erzeugt und deployed – dies entspricht den, beim Deployment-Prozess für Applikationsservice angeführten, acht Schritten.

Zusätzlich zum Deployment des Services müssen noch die verwendeten Datenressourcen eingerichtet werden. Dies umfasst folgende zusätzliche, für Datenservices spezifischen Deployment-Schritte:

9. Installation von OGSA-DAI auf dem Tomcat-Server

   Zusätzlich zum VGE-Service wird das OGSA-DAI WSI Service deployed.
10. Einrichten eines *OGSA-DAI Data Service*

    Ein *OGSA-DAI Data Service* wird eingerichtet, dass den Zugriff auf sämtliche Datenressourcen ermöglichen soll.

[35] Von Axis erzeugte WSDL-Dateien sind nicht WS-I konform.
[36] Servlets basierend auf dem AxisServlet werden erst beim ersten Zugriff durch den Client initialisiert. VGE-Services werden hingegen bereits am Ende des Deployment-Prozesses initialisiert. Dadurch wird einerseits ein erfolgreiches Deployment garantiert (sämtliche benötigten Dateien und Bibliotheken sind vorhanden), anderseits kommt es zu keinen Wartezeiten durch die Initialisierung beim ersten Zugriff.

Schritt 11 und 12 werden für jede durch das VGE Datenservice benutzbare Datenressource durchgeführt:

11. Einrichten der *Data Service Resources*

    Die *Data Service Resource* wird eingerichtet und im *OGSA-DAI Data Service* eingetragen. Hierfür sind die mit dem VGE Deployment-Tool erzeugten *Data Service Resource* -Dateien notwendig (siehe „Listing 8: Data Service Resource-Datei”).

12. Freigabe zur Nutzung der *Data Service Resource* durch das *Data Service*

    Die *Data Resource* ist sofort ohne Neustart des Service Containers verfügbar und kann vom OGSA-DAI *Data Service* bzw. dem VGE-Datenservice genutzt werden.

Der OGSA-DAI Deployment-Prozess ist ebenfalls ANT-basiert. Der VGE-Deployment-Prozess ruft zur Durchführung der Schritte 9 – 12 die entsprechenden ANT-Prozeduren des OGSA-DAI Provisioning-Environments auf.

### 3.5.3 Client-Provisioning

Das Client-Provisioning bezeichet das Einrichten des Clients bzw. dessen Umgebung, um die Kommunikation mit bzw. die Nutzung von Services zu ermöglichen. Client-seitiges Provisioning wird für den Web-Client unterstützt. Sämtliche Daten werden über eine Client-Properties-Datei zur Verfügung gestellt. Diese Daten sind:

- Client-Kontextname für die Hosting-Umgebung
- Angaben zur Hosting-Umgebung, in der der Web-Client installiert werden soll, wie
  - URI des Web Servers
  - Account-Daten für die Benutzung des Application Managers von Tomcat

  Diese Parameter sind analog den Hosting-Daten bei der individuellen Service-Konfiguration.

- Optional die Angabe von Konfigurationsparameter für die Security-Komponente
  - z.B. Name der Datei mit X.509 Zertifikaten

Ein Beispiel für eine Client-Properties-Datei ist in Listing 9 angeführt.

Der Deployment-Prozess des Web-Clients ist ant-basiert und umfasst folgende vier Schritte:

1. Erzeugen des *Web-Application-Deployment*-Deskriptors (/WEB-INF/web.xml)
2. Erzeugen interner Properties-Dateien, die während der Laufzeit benötigt werden und z.B. Daten der Client-Web-Anwendungsumgebung beinhalten

3. Archivierung sämtlicher Dateien zu einem *Web Application Archive* (WAR-Datei) gemäß der Java-Servlet-Spezifikation. Dieses Archiv beinhaltet die *web.xml*-Datei, VGE-Bibliotheken, externe Bibliotheken, usw.
4. Installation des Web Service auf dem Tomcat-Server mit Hilfe der Tomcat Manager Applikation – aus Web Server-Sicht ist dies der „eigentliche“ Deployment-Prozess.

```
# Example client-specific properties file.
#
# NOTE: If you edit this file for your environment, please do not
#       commit it to the repository (SVN).

# The client.name property indicates the client context name to be
# used for the deployment of a client in the hosting environment.
# E.g. if there is no client.context.prefix property defined and the
# client.name is "gsclient" your client will be installed at:
# http://localhost:9090/gsclient

client.name=gsclient


# Prefix with ending slash (except if it is empty)!

client.context.prefix=


# The client.protocol + client.host + client.port propterties
# specifies an URL (without ending slash!) to the user/running Tomcat.

client.protocol=http
client.host=localhost
client.port=9090


# The client.manager.user and client.manager.pwd specify the username
# and password required for the manager servlet of Tomcat to execute
# manager tasks (for further details see
# <tomcat-dir>/conf/tomcat-users.xml configuration file).

client.manager.user=manager
client.manager.pwd=manager


# The (optional) client.remote.dist property specifies (if not empty!)
# a fully qualified pathname to the distribution directory (contains
# the war-files) on the host where the destination Tomcat installation
# runs. If you deploy your clients on the localhost you do not need
# to specify this property (just leave it emty).

client.remote.dist=


# The client.components property indicates which client-components
# will be installed/deployed. Currently the following client-components
# are supported:
# E2ESecurity|WSS4J, ApplicationClient, QoSClient, DataClient
# Note 1: You must specify at least the ApplicationClient client-
#         component
# Note 2: The order of the components indicates the order of
#         installation/deployment and hence may have side-effects on
#         not yet deployed client-components.

client.components=ApplicationClient



client.upload.path=/home/gerry/Projects/Eclipse-Workspace/VGE-GCPR/upload
client.services.file=/home/gerry/Projects/Eclipse-Workspace/VGE-GCPR/upload/services.dat
client.keystore.file=

client.security.keystore.file = client.jks
client.security.keystore.pwd = changeit
client.security.keystore.alias = myalias
client.security.policy.file = client.policy
client.security.crl.file = client.crl
```

**Listing 9: Web-Client Properities-Datei**

# 4 Erweiterung des VGE

Die Erweiterung des VGE umfasst im Wesentlichen sowohl ein zentrales Job Management als auch eine Benutzer-Verwaltung in einem VGE-Grid.

## 4.1 Ziele des Job Managers

Das Ziel des Job Managers ist die Verwaltung orts- und client-unabhängig laufender sowie abgeschlossener Jobs eines eindeutig identifizierbaren Benutzers. Dadurch ist

- die Statusabfrage von auf verschiedenen Clients gestarteten Jobs,
- die Abfrage vom Ergebnis eines beendeten Jobs und
- der Abbruch laufender Jobs

möglich.

Neben dem eigentlichen Ziel des Job Managements soll als weiteres Ziel eine persistente Speicherung der Job-Daten erfolgen, um eine Archivierung bzw. Auswertung diverser Nutzungsdaten zu ermöglichen. Dem Benutzer kann, aufgrund der gespeicherten Informationen, eine chronologische Auflistung der benutzten Services für eine spätere Wiederverwendung angezeigt werden.

Der Job Manager ist als zusätzliche Client-Komponente realisiert und kann optional mit bestehenden Clients verwendet werden.

### *Erweiterungsszenarien*

Das Ziel der Erweiterung des VGE-Systems durch den Job Manager wird durch folgende Szenarien verdeutlicht:

I. <u>Statusabfragen von Jobs auf verschiedenen Clients</u>

1. Ein Benutzer startet auf Client 1 einen Job A.
2. Danach startet der Benutzer auf Client 2 einen Job B.

⇨ Um den aktuellen Status des Jobs A abzufragen, muss der Benutzer den Client 1 benutzen. Der Status des Jobs B ist nur von Client 2 ersichtlich.
Beispiele für dieses Szenario sind sowohl die Benutzung von 2 verschiedenen Web-Clients, als auch, dass Client 1 ein Web-Client und Client 2 ein Kommandozeilen-Client ist.

❖ Lösung: Der Benutzer kann sämtliche gestarteten Jobs mit einem Client verwalten, unabhängig davon, auf welchem Client die Jobs gestartet wurden.

II. Statusabfragen von Jobs verschiedener Benutzer (eventuell auf verschiedenen Clients gestartet)

VGE-Services werden von mehreren Personen bzw. Instituten genutzt. Laufende Jobs sollen von einer Person bzw. einem Institut verwaltet werden. Ein weiterer Anwendungsfall wäre, dass für die Bereitstellung von Eingabedaten und das Starten der Jobs eine Person bzw. Personengruppe zuständig ist. Die Auswertung der Ergebnisse erfolgt von einer anderen Person bzw. Personengruppe.

⇨ Jobs sind im Falle des Web-Clients nur für den Initiator sichtbar. Für die Job-Administration mehrerer Benutzer ist eine Authentifizierung mit den jeweiligen Zugriffsdaten erforderlich. Jobs, durch den Kommandozeilen-Client initiiert, sind keinem Benutzer zugeordnet und nur der ausführende Benutzer kennt die Job-Daten.

❖ Lösung: Nach einer erfolgreichen Authentifizierung kann der Benutzer sämtliche Jobs von anderen Personen verwalten, für die der Benutzer zur Administration berechtigt ist.

III. Archivierung

⇨ Web-Client:

Wird ein Job nach dem Herunterladen der Ergebnisse gelöscht, so ist der Job bzw. seine Charakteristika (aufgerufenes Service, Upload-, Start- und Download-Zeitpunkt, Status) nicht mehr verfügbar. Dasselbe gilt für den Fall eines Restarts des Clients.

⇨ Kommandozeilen-Client:

Job-Charakteristika sind nach der Ausführung mit dem Kommandozeilen-Client nicht mehr verfügbar.

❖ Lösung: Eine permanente Speicherung der Job-Daten ermöglicht jederzeit einen Zugriff auf die Informationen, auch wenn der Job am Client nicht mehr verfügbar ist.

IV. Überblick

- Benutzte Clients
- Aufgerufene Services
- Benutzerverhalten (Benutzer A hat Services X, Y und Z benutzt. Diese Informationen sind z.B. für eine spätere Wiederverwendung sinnvoll.)

⇨ Diese Daten sind derzeit sowohl für den Benutzer als auch für den Service Provider zur Gewinnung eines Überblickes nicht vorhanden.

❖ Lösung: Ein Protokoll der Job-Aktivitäten im VGE-System stellt die benötigten Daten für einen Überblick zur Verfügung.

Aus diesen Szenarien ergeben sich gewisse Anforderungen an den Job Manager.

## 4.2 Anforderungen

Folgende Anforderungen ergeben sich aufgrund der Szenarien im vorigen Kapitel bzw. aus der Aufgabenstellung der Diplomarbeit an die VGE-Erweiterung:

- Jeder Job ist im VGE-System eindeutig identifizierbar.
- Aktivitäten von Clients (Initialisierung eines Jobs bzw. Übertragung der Eingabedatei(en), Starten von Jobs, Statusabfragen, etc.) – in Folge als Job-Aktivitäten bezeichnet - werden überwacht und relevante Job-Daten werden protokolliert.
- Diese Daten sind zentral verfügbar.
- Durch diese Daten ist der letzte, dem Client bekannte Job-Status der Client-Service-Interaktionen feststellbar.
- Diese Daten werden permanent gespeichert und sind nicht mehr vom Client abhängig. Auch nach dem Ende eines Jobs sind diese Daten verfügbar.
- Durch die protokollierten Daten ist eine zentrale Administration der Jobs möglich. Job-Aktivitäten (Start, Abbruch, Statusabfrage, Herunterladen der Ergebnisse) können unabhängig vom dem Client, der den Job initiierte, ausgeführt bzw. fortgesetzt werden.
- Diese Daten beinhalten einerseits Job-Daten als auch Benutzer-Daten, die einer Person zugeordnet werden können. Werden VGE-Services mit einem Web-Client benutzt, so sind dies die Login-Daten des Benutzers. Werden VGE-Services automatisch durch Skripte, Programme etc. benutzt, so wurde diese automatische Benutzung von einer Person eingerichtet bzw. ist eine Person dafür verantwortlich. Daher können Jobs immer einer Person zugeordnet werden und eine Job-Person-Verbindung ist vorhanden.
- Für eine zentrale Administration der Jobs ist eine Authentifizierung notwendig, damit eine Person nur die eigenen Jobs administrieren kann bzw. die Jobs jener Personen, für die ihr die Berechtigung erteilt wurde.
- Eine Benutzer-Verwaltung ist aufgrund der Authentifizierung notwendig bzw. um die Administration von Jobs durch Dritte zu ermöglichen.
- Der Job Manager ist eine funktionale Erweiterung des VGE-Systems. Die Architektur des VGEs wird somit nicht verändert.
- Service-seitig dürfen keine Änderungen vorgenommen werden, damit die Nutzbarkeit von Services nicht unterbrochen wird (laufende Jobs sind von der Integration bzw. Inbetriebnahme des Job Managers nicht betroffen).
- Eine Administration der Jobs ist auch mit mobilen Endgeräten möglich.

Aus den zuvor beschriebenen Szenarien kristallisieren sich zwei Entitäten im Kontext des Job Managers heraus: der Benutzer und seine Jobs.

## 4.3 Die Entititäten User und Job

In den Anwendungsszenarien sind zwei Entitäten feststellbar: Benutzer (User) und deren Jobs. Dieser Abschnitt beschreibt die beiden Entitäten User und Job mit ihren Charakteristika und deren Beziehung zueinander, die sich aus den Anwendungsszenarien ergeben. Die Beziehung zwischen User und Job wird in einem Entity-Relationship-Diagramm (ER-Diagramm) dargestellt.

### 4.3.1 User

Ein Benutzer (User) verwendet VGE-Services. Der User startet von einem Client Jobs, kontrolliert deren Status und lädt nach erfolgreicher Beendigung des Jobs dessen Ergebnisse herunter oder bricht die Ausführung eines Jobs ab.

Ein Benutzer ist durch folgende Attribute charakterisiert:

- eine eindeutige Benutzerkennung und
- ein Passwort zur Authentifizierung

### 4.3.2 Job

Das Benützen eines Services durch einen User resultiert in der Ausführung eines Jobs. Ein Job hat folgende Eigenschaften:

- den Initiator (User) des Jobs
- eine im VGE-System eindeutige Job-ID
- den Client bzw. dessen Netzwerkadresse, auf dem der Job gestartet wurde
- das aufgerufene Service bzw. dessen Netzwerkadresse
- den aktuellen Status des Jobs

### 4.3.3 User-Job-Diagramm

Das ER-Diagramm in Abbildung 42 symbolisiert die Beziehung zwischen User und Jobs.

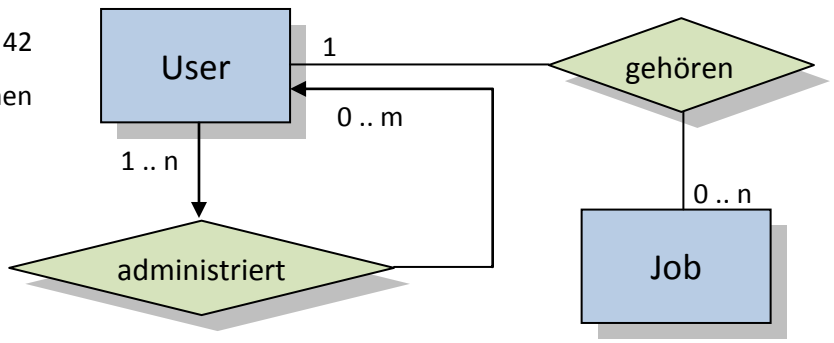



Abbildung 42: User-Job-Diagramm

Die im Job Manger involvierten Entitäten sind beschrieben, somit können die Architektur und die Bestandteile des Job Managers beschrieben werden.

## 4.4 Lösungsansatz und Architektur

Aufgrund der Szenarien bzw. Anforderungen wurde folgender Lösungsansatz gewählt:

- Job-Aktivitäten werden client-seitig mit einer zusätzlichen Komponente überwacht.
  - ⇨ Job Monitoring Komponente
- Das VGE selbst wird zur Realisierung der Datenspeicherung und des Datenzugriffes benutzt, indem die protokollierten Daten in einem Datenservice gespeichert werden.
  - ⇨ Job Manager Data Service
- Die Administration von Jobs und Benutzern wird durch zwei Interfaces ermöglicht:
  - ⇨ Job Manager Client
  - ⇨ Mobile Job Manager Client

### 4.4.1 Architektur

Im Mittelpunkt der Architektur (Abbildung 43) steht das Job Manager Data Service als Datenspeicher, die Job Monitoring Komponente fungiert als Datenlieferant und die beiden Clients dienen als Interfaces und ermöglichen die Administration von Benutzern und Jobs.

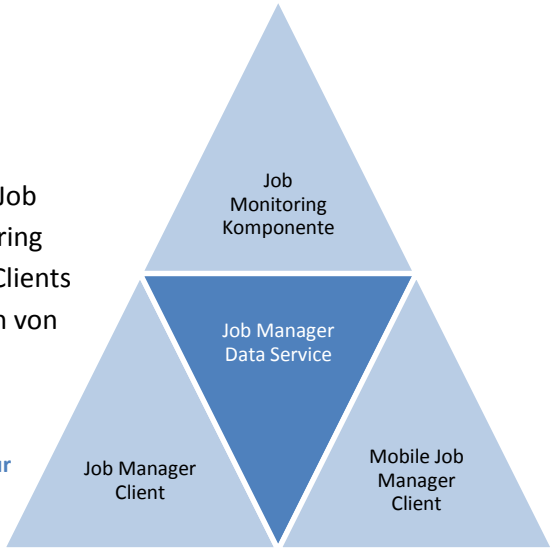



**Abbildung 43:**
**Job Manager Architektur**

### 4.4.2 Job Monitoring Komponente

Wird ein VGE-Client mit der Komponente *„JobMonitor“* installiert, so werden die Aktionen des Clients überwacht. Bei gewissen Aktivitäten (*„Upload“*, *„Start“*, *„Download“*, *„Kill“*) werden relevante Job-Daten mit dem Job Manager Data Service gespeichert oder aktualisiert.

### 4.4.3 Job Manager Data Service

Das Job Manager Data Service ist ein VGE Datenservice und dient zur Speicherung der Job-Daten in einer relationalen Datenbank. Die Datenbank wird mit einem definierten Datenbankschema zur Speicherung der Job-Daten eingerichtet.

### 4.4.4 Job Manager Client

Der Job Manager Client dient dem Zugriff auf die gespeicherten Job-Daten und ermöglicht das Starten von Jobs, die Abfrage des aktuellen Status und den Abbruch laufender Jobs bzw. das Herunterladen von Ergebnissen abgeschlossener Jobs. Außerdem erfolgt mit diesem die Benutzerverwaltung, da nur Job-Daten gespeicherter Benutzer erfasst werden. Hat ein Benutzer Administrator-Rechte, dann kann diese Person auch Benutzerdaten und Jobs von Personen verwalten, für die sie zur Administration berechtigt ist.

### 4.4.5 Mobile Job Manager Client

Der Mobile Job Manager Client ermöglicht den Zugriff auf die, einem Benutzer zugeordneten, laufenden Jobs und kann deshalb auch als „Personal Job Manager“ bezeichnet werden. Durch den Mobile Job Manager Client kann der aktuelle Status eines Jobs ermittelt bzw. ein Job abgebrochen werden.

### 4.4.6 VGE-Service Architektur

An bestehenden Applikations- und Datenservices müssen keine Änderungen durchgeführt werden, um den Job Manager zu nutzen.

### 4.4.7 VGE-Client Architektur

Ein VGE-Client kann mit der optionalen Komponente *JobMonitor* installiert werden, um dessen Aktivitäten für den Job Manager mitzuprotokollieren. VGE-Clients mit hinzugefügter *JobMonitor*-Komponente reagieren auf gewisse Ereignisse des Clients. Werden mit diesen Clients die Aktionen „*Upload*“, „*Start*“, „*Download*“ oder „*Kill*“ eines Jobs durchgeführt, hat dies zur Folge, dass relevante Job-Daten wie Job-ID, Service-URI, etc. im Job Manager Data Service gespeichert oder aktualisiert werden.

### 4.4.8 Konzept des Job Managers

Abbildung 44 stellt das Konzept des Job Managers dar und zeigt in der Mitte zwei Applikationsservices (1 & 2) und ein Datenservice (A). Client 1 und Client 2 sind VGE-Clients mit hinzugefügtem Job Monitoring. Zu dem bestehenden VGE-System sind das Job Manager Data Service, der Job Manager Client und der Mobile Job Manager Client (abgesehen von der hinzugefügten Komponente bei den Clients 1 & 2) neu hinzugekommen.

Die grünen, punktierten Pfeile entsprechen z.B. Status Abfragen von Jobs oder dem Abbruch eines Jobs durch den Job Manager Client bzw. Mobile Job Manager Client. Die blauen, strichlierten Pfeile symbolisieren die Speicherung oder Aktualisierung von Job-Daten im Job Manager Data Service.

Die schwarzen Pfeile entsprechen der herkömmlichen VGE-Client-Service-Kommunikation.

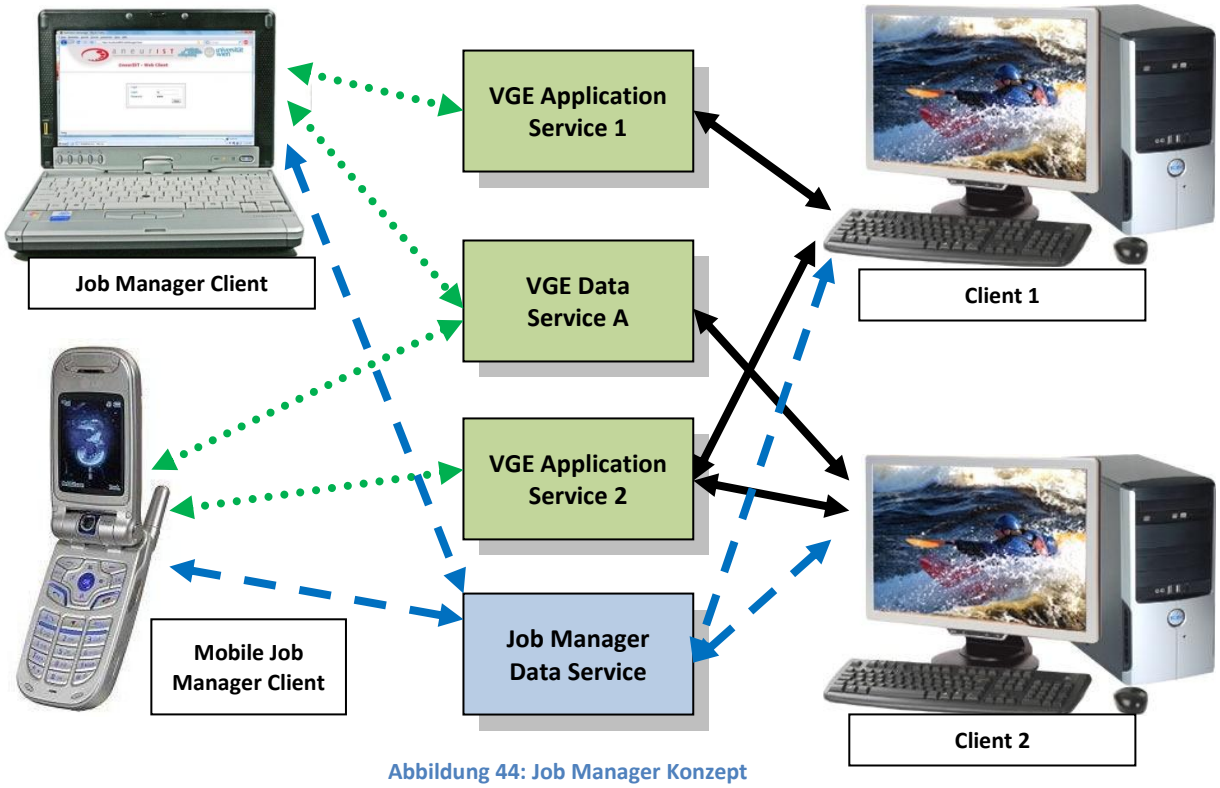



Abbildung 44: Job Manager Konzept

Aufgrund des Job Manager Konzeptes ergeben sich zusätzliche, technische Anforderungen an das VGE-System.

## 4.5 Technische Anforderungen

Folgende Anforderungen sind an das VGE-Sytem gegeben, um den Job Manager zu nutzen:

- Das Job Manager Data Service bzw. dessen relationale Datenbank muss mit einem definierten Datenbankschema eingerichtet sein.
- Für die Benutzung des Job Managers ist es Voraussetzung, dass die Benutzer (deren Profil inklusive Benutzerkennung) in der Datenbank des Job Manager Data Services gespeichert sind. Die Job Monitoring Komponente speichert nur Jobs, deren Benutzer bzw. Initiator in der Datenbank vorhanden ist. Eine Benutzung des Web-Interfaces „Job Manager Client“ bzw. des MIDlets „Mobile Job Manager Client“ setzt eine erfolgreiche Authentifizierung des Benutzers voraus.

- Das VGE wird durch einen neuen Job-Status – Status „*Killed*“ – erweitert, um den Abbruch eines Jobs durch den Benutzer zu speichern.
- VGE-Clients (Web-Client, Kommandozeilen-Client, …) mit installierter *Job Monitoring*-Komponente reagieren auf gewisse Ereignisse (*EventListener*) und führen entsprechende Job Manager Aktivitäten aus.

Das Kapitel Interaktionen verdeutlicht die zeitliche Abfolge der Job Manager Aktivitäten von Clients und Services.

## 4.6 Interaktionen

Abbildung 45 verdeutlicht die Interaktionen des Job Managers:

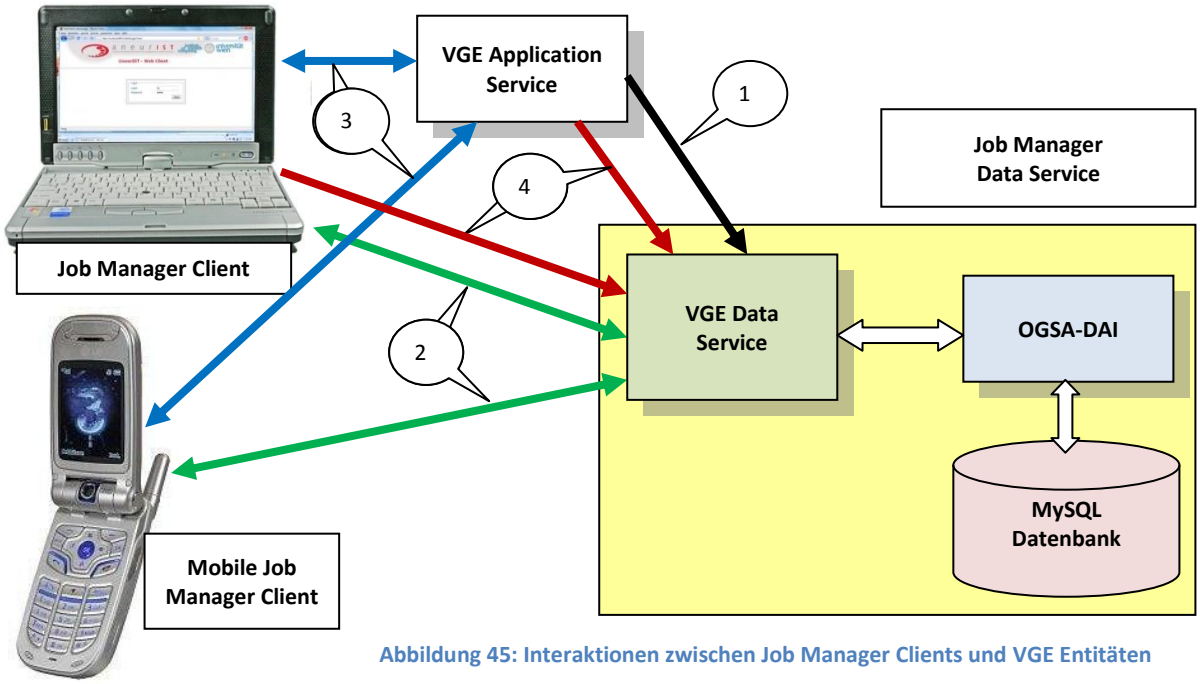



Abbildung 45: Interaktionen zwischen Job Manager Clients und VGE Entitäten

1) Ein VGE Client, der Job Monitoring benutzt, reagiert auf gewisse Aktionen eines Clients (bzw. eines *AppProxies*). Folgende Aktionen werden überwacht:
   - Aktion „*Upload*“: Mit dem Job Manager Data Service werden relevante Daten des Jobs (z.B. URI des aufgerufenen Applikationsservices; URI des Clients, von dem der Job initialisiert wurde) gespeichert. Der Job-Status wird auf „*Uploaded*“ gesetzt und der aktuelle Zeitpunkt gespeichert.

     Damit die Job-Daten gespeichert werden können, muss zuvor ein Anlegen des Benutzers mit dem Job Manager Client erfolgt sein.

- Aktion „*Start*“: Der Job-Status wird auf „*Started*“ gesetzt bzw. im Job Manager Data Service aktualisiert.

Diese Kommunikation wird durch den schwarzen Pfeil mit der Nummer 1 verdeutlicht.

2) Job Manager Client und Mobile Job Manager Client greifen auf das Job Manager Data Service zu, um Daten gespeicherter Jobs zu lesen – dies ist durch grüne Pfeile mit der Nummer 2 in der Grafik symbolisiert.
3) Job Manager Client bzw. Mobile Job Manager Client rufen den aktuellen Status laufender Jobs ab (Ergebnis der zuvor erfolgten Abfrage in Interaktion 2) – diese Aktion entspricht den blauen Pfeilen mit der Nummer 3.
4) Mit dem Job Manager Client oder dem VGE-Client wird der Download des Ergebnisses durchgeführt. Diese Aktion wird ebenfalls durch das Job Monitoring im Falle der Benutzung des VGE-Clients überwacht.
    - Aktion „*Download*“: Der Status wird auf „*Downloaded*“ gesetzt. Die Interaktion mit dem Job ist somit abgeschlossen.

Die roten Pfeile mit der Nummer 4 stellen diese Kommunikation dar.

# 5 Implementierung

In diesem Kapitel wird die Realisierung des Job Manager Data Services, der Job Monitoring Komponente und der beiden Clients, Job Manager Client und Mobile Job Manager Client, beschrieben.

## 5.1 Job Manager Data Service

Job-Daten sind in einem Datenservice – dem Job Manager Data Service - gespeichert. Im Konzept wurden die beiden Entitäten User, Job und deren Beziehung zueinander bereits beschrieben. Eine detailierte Auflistung (mit zusätzlichen Attributen zu den im Konzept bereits aufgelisteten) bzw. deren Abbildung in der Datenbank folgt in den nächsten Abschnitten.

### 5.1.1 User

Ein Benutzer weist folgende Attribute auf:

- Vor- und Nachname
- eine eindeutige Benutzerkennung und ein Passwort zur Authentifizierung
- Eigenschaften, um eine Benutzeradministration zu ermöglichen:
    - Benutzer muß das Passwort ändern
    - Benutzer ist gesperrt
    - Benutzer ist ein Administrator und hat das Recht, andere Benutzer und deren Jobs zu verwalten

Weitere, jedoch nicht zwingende Attribute sind:

- eine Email-Adresse für Benachrichtungen durch den Job Manager
- die IMEI[37] eines mobilen Endgerätes (deren Bedeutung wird beim Mobile Job Manager Client erläutert)

### 5.1.2 Job

Ein Job hat folgende Attribute:

- eine im VGE-System eindeutige Job-ID

[37] Die International Mobile Equipment Identity (IMEI) ist eine 15-stellige Seriennummer zur eindeutigen Identifikation eines GSM- oder UMTS-Endgerätes.

- den Initiator (User) des Jobs
- den Client bzw. dessen Netzwerkadresse, auf dem der Job gestartet wurde
- das aufgerufene Service bzw. dessen Netzwerkadresse
- den aktuellen Status des Jobs
- Pattern der Ergebnisdatei (aufgrund der Nutzung eines VGE-Services)
- verschiedene Zeitstempel:
    - Zeitpunkt des Hochladens der Eingabedaten
    - Startzeitpunkt des Jobs
    - Zeitpunkt des Herunterladens der Ergebnisse
    - Zeitpunkt des Abbruchs des Jobs durch den Benutzer

Das Proxy-API wurde im „Kapitel 3.4 Client Infrastruktur“ beschrieben bzw. es ist ein Beispiel für dessen Verwendung in „Listing 5: VGE Client Code Beispiel für die Nutzung eines Applikationsservices“ angeführt. Die Job-ID ist die ID des Proxies, der für die Kommunikation mit dem Service benutzt wird. Der Status eines Jobs entspricht dem Status des Proxies.

### 5.1.3 Datenbankmodell

Aus den zuvor beschriebenen Entitäten mit ihren Attributen und deren Beziehung zueinander resultiert das Datenbankmodell des Job Manager Data Service, welches in Abbildung 46 dargestellt ist.

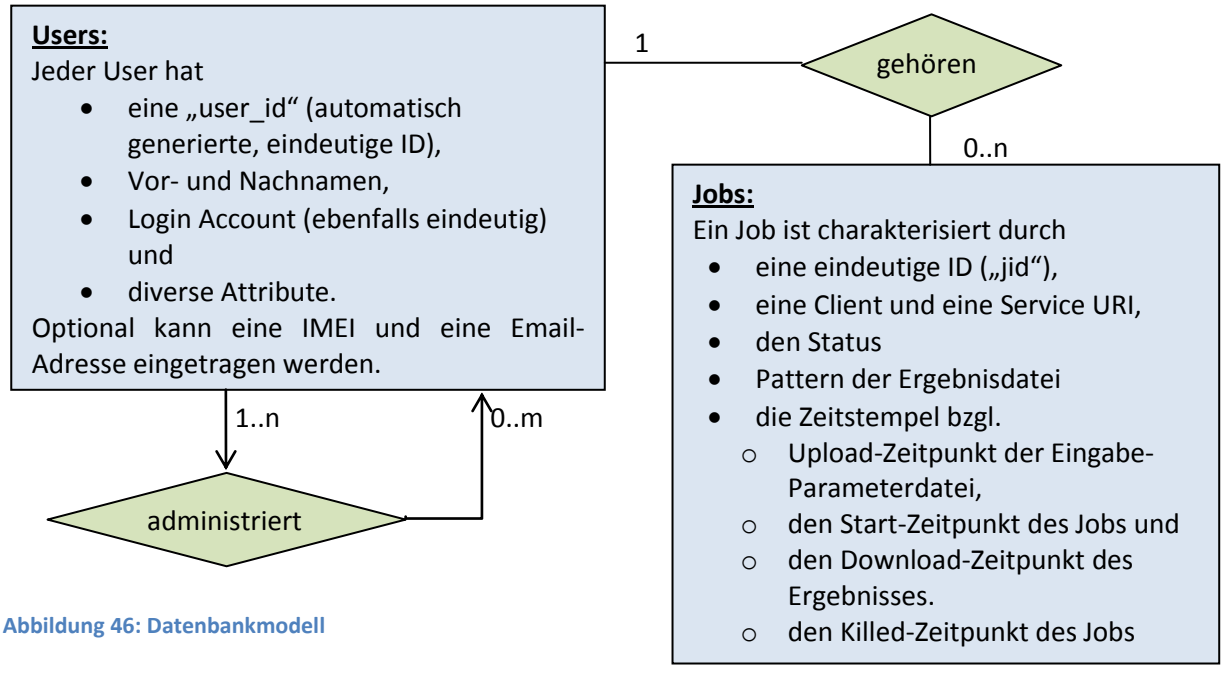



Abbildung 46: Datenbankmodell

Die Beziehung „*administriert*“ wird durch eine Tabelle „*administrate_user*“ ausgedrückt. Jeder User, dessen Attribut „*isAdmin*“ den Wert TRUE hat, kann andere User administrieren.

### 5.1.4 Data Resource - Realisierung mittels MySQL

Das Job Manager Data Service wurde mit einer MySql-Datenbank umgesetzt, wobei auch jede andere von OGSA-DAI unterstützte relationale Datenbank (siehe Kapitel „OGSA-DAI Data Resources“) verwendet werden kann. Die Unabhängigkeit des Datenbanksystems wird durch OGSA-DAI gewährleistet.

Als Entwicklungs- und Testumgebung wurde ein MySQL Community Server mit der Version 5.0.51a verwendet.

Zur Umsetzung des Datenbankmodelles wurde ein Schema „*vgds_jobmanager*“ mit den Tabellen „*users*“, „*jobs*“, „*administrate_user*“ angelegt:

| Tabelle „**users**“: | | | |
|---|---|---|---|
| Name | Datentyp | Muss-feld | |
| user_id | INTEGER | ✓ | Primärschlüssel;<br>wird von MySQL automatisch generiert |
| login | VARCHAR(15) | ✓ | Sekundärschlüssel (eindeutig) |
| first_name | VARCHAR(20) | ✓ | |
| last_name | VARCHAR(30) | ✓ | |
| password | CHAR(32) | ✓ | |
| change_pw | BIT(1) | ✓ | Standardwert: TRUE (1) |
| locked | BIT(1) | ✓ | Standardwert: FALSE (0) |
| isAdmin | BIT(1) | ✓ | Standardwert: FALSE (0) |
| imei | BIGINT | | |
| email | VARCHAR(60) | | |

`User_id` ist eine von MySql automatisch generierte, eindeutige Zahl. Passwörter werden in der Datenbank MD5-kodiert abgespeichert. MD5 (Message-Digest Algorithm 5, **[Riv92]** ) ist eine weit verbreitete kryptographische Hash-Funktion, wird von vielen Applikationen und Systemen unterstützt und erzeugt eine 32-stellige Hexdezimalzahl. In MySql wird das Generieren von MD5-Werten hiermit unterstützt: „`select md5('my password');`“.

`BIT(1)` wird zur Speicherung von *boolean*-Werten für die Attribute bzgl. der Benutzeradministration verwendet.

Eine IMEI ist eine 15-stellige Nummer und wird durch den Datentyp `BIGINT` abgebildet.

| Tabelle „**jobs**“: | | | |
|---|---|---|---|
| Name | Datentyp | Muss-feld | |
| jid | VARCHAR(30) | ✓ | Primärschlüssel |
| user_id | INTEGER | ✓ | Sekundärschlüssel |
| client_uri | VARCHAR(100) | ✓ | |
| service_uri | VARCHAR(100) | ✓ | |
| state | TINYINT | ✓ | Standardwert: Uninitialized (-2) |
| output_file_pattern | VARCHAR(30) | | |
| date_uploaded | DATETIME | | |
| date_started | DATETIME | | |
| date_downloaded | DATETIME | | |
| date_killed | DATETIME | | |
| Einschränkung:<br>Feld „`user_id`“ verweist auf die Tabelle „`users`“, Feld „`user_id`“.<br>Eine Lösch- oder Aktualisierungsaktion wird nicht durchgeführt, sollte es dadurch zu einer „Verletzung“ in der Sekundärtabelle kommen. | | | |

Der Datentyp `DATETIME` wurde zur Abspeicherung der diversen Zeitstempel eines Jobs gewählt.

Die Tabelle „`administrate_user`“ bildet die Beziehung ab, dass ein Benutzer andere Personen und auch deren Jobs administrieren darf.

| Tabelle „**administrate_user**“: | | | |
|---|---|---|---|
| Name | Datentyp | Muss-feld | |
| admin_id | INTEGER | ✓ | |
| user_id | INTEGER | ✓ | |
| Einschränkung:<br>Felder „`admin_id`“ und „`user_id`“ verweisen auf die Tabelle „`users`“, Feld „`user_id`“.<br>Eine Lösch- oder Aktualisierungsaktion wird nicht durchgeführt, sollte es dadurch zu einer „Verletzung“ in der Sekundärtabelle kommen. | | | |

Im nächsten Kapitel wird das Client-API des Job Managers beschrieben.

## 5.2 Client-API

Eine abstrakte Darstellung des Workflows bzgl. der Nutzung eines Datenservices (analog dem Beispiel „Listing 5: VGE Client Code Beispiel für die Nutzung eines Applikationsservices“) ist in Abbildung 47 dargestellt.

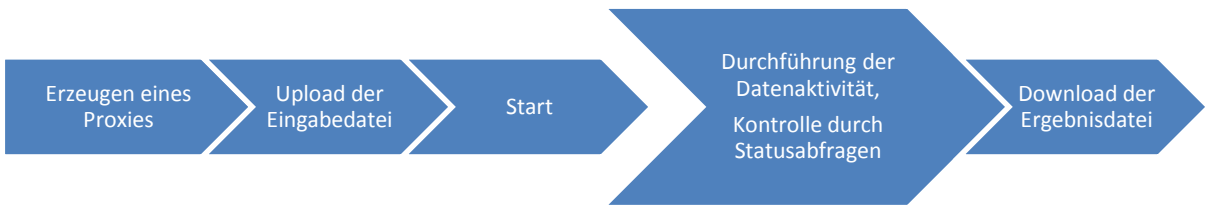



Abbildung 47: Workflow der Nutzung eines Datenservices

Dieser Workflow der Nutzung eines Datenservices wird nun in einem neuen API – dem *DatabaseAgent* – gekapselt. In Bezug auf die abstrakten Schichten des Client APIs in Abbildung 34 ist der *DatabaseAgent* Teil der Agent-Schicht und setzt auf die ServiceProxy-Schicht auf (siehe Abbildung 48).

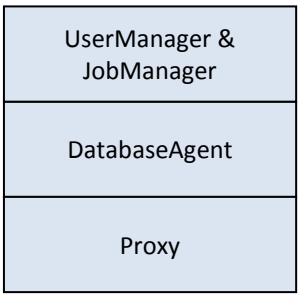



Abbildung 48: Agent- & Manager-Schicht

UserManager- und JobManager-API benutzen den DatabaseAgent für deren spezifische Funktionalität.

### 5.2.1 Low-level API Database Agent

**DatabaseAgent:** Der *DatabaseAgent* wird für jede Aktion, seitens des *UserManagers* als auch des *JobManagers*, in der Datenbank benutzt. Er ermöglicht das Durchführen von SQL-Abfragen (`SELECT`) als auch SQL-Aktivitäten wie `INSERT`, `UPDATE`, `DELETE`. Intern verwendet er das Proxy-API zur Kommunikation mit dem Job Manager Data Service.

### 5.2.2 High-level APIs JobManager und UserManager

**UserManger:** Der *UserManager* ermöglicht den Zugriff auf die gespeicherten User in der Datenbank und stellt Operationen der Benutzerverwaltung wie das Anlegen, Editieren und Löschen von Benutzern zur Verfügung.

**JobManager:** Der *JobManager* ermöglicht den Zugriff auf die gespeicherten Jobs bzw. Job-Daten und stellt Operationen zur Verfügung, wie Statusabfragen von Jobs und deren Änderung in der Datenbank, Löschen von Jobs aus der Datenbank, etc.

Die Verwendung des high-level-APIs wird im nachfolgenden Beispiel-Code (Listing 10) veranschaulicht:

```
1  // creating database agent, user & job manager
2  DatabaseAgent db = new DatabaseAgent( sURI, sWorkingDir, sPatternInputFile, sPatternOutputFile );
3  UserManager userManager = new UserManager(db);
4  JobManager jobManager = new JobManager(db);
5
6  // checks if user is stored in the database
7  User user = userManager.getUser( sAccount );
8  if ( user != null ) {
9     // user is available
10    if ( ! user.getLocked() ) {
11       // user is not locked
12       System.out.println("Hello, "+user.getDisplayName());
13
14       // getting jobs from database
15       Jobs[] jobs = jobManager.getJobs( user.getLogin() );
16       System.out.println("You have " + job.length + " jobs stored in the database.");
17       // getting actual job status
18       for (int i=0; i<jobs.length; i++) {
19          jobs[i].updateJobStatus();
20          System.out.println("Job " + job.getJID() + " has status " + job.getState());
21       }
22    }
23 }
```

Listing 10: Beispiel-Code für die Nutzung des User- & JobManager APIs

Listing 10 beinhaltet einen Beispiel-Code, bei dem das *User*- und das *JobManager*-API verwendet werden. In der Zeile 2 wird eine `DatabaseAgent`-Instanz erzeugt, in Zeile 3 & 4 jeweils eine Instanz des `UserManager` und des `JobManager`. Beide bekommen als Parameter die zuvor angelegte `DatabaseAgent`-Instanz mit. In Zeile 7 wird mit dem `UserManager` überprüft, ob ein Benutzer mit einer bestimmten Kennung vorhanden ist. Beim erstmaligen Aufruf von `getUser()` werden implizit alle Benutzer von der Datenbank geladen. Ist der `User` vorhanden und nicht gesperrt, so werden seine `Jobs` von der Datenbank geladen (in Zeile 15). Für seine `Jobs` wird der aktuelle Status ermittelt und ausgegeben - Zeile 19 & 20. Hat sich bei einem `Job` der Status geändert, so speichert `updateJobStatus()` implizit den neuen Status in der Datenbank.

### 5.2.3 Komponenten, APIs & Interfaces

Der Architektur des Job Managers wird in der Abbildung 49 dargestellt.

Die Job Monitoring-Komponente, der Job Manager Client und der Mobile Job Manager Client greifen mit den high-level-APIs (JobManager & UserManager) oder mit dem low-level-API (DatabaseAgent) auf das Job Manager Data Service zu.

Der Job Manager Client stellt ein Web-Interface in Form eines Web-Servlets zur Verfügung, der Mobile Job Manager Client ein MIDlet-Interface.

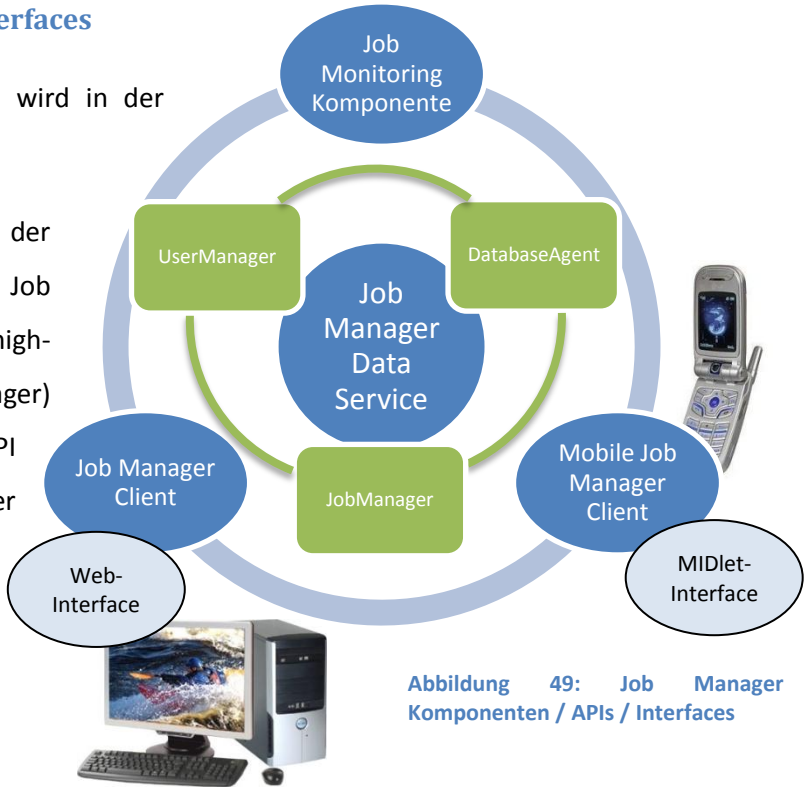



**Abbildung 49: Job Manager Komponenten / APIs / Interfaces**

## 5.3 Job Monitoring-Komponente

Die Funktionalität der Job Monitoring-Komponente wurde bereits im Kapitel „4.4.2 Job Monitoring Komponente“ beschrieben. Hier wird nun die technische Umsetzung erläutert.

Wird ein VGE Client (Web-Client, Kommandozeilen-Client, …) mit der Komponente *JobMonitor* installiert, so nutzt er den *GridEventListener* (abgeleitet vom Java EventListener), um auf die Aktionen „*Upload*“, „*Start*“, „*Download*“ und „*Kill*“ eines Clients (bzw. eines *AppProxy*) zu reagieren. Dem

*AppProxy* wird mit der Methode *proxy.addGridEventListener* die vom *GridEventListener* abgeleitete Klasse *JobManagerGridEventListener* hinzugefügt. Diese führt die entsprechenden Aktionen zur Speicherung oder Aktualisierung der Job-Daten im Job Manager Data Service durch.

Für die Aktion „*Kill*“ ist es notwendig, dass der Status eines Proxies (*AppProxyState*) um die Ausprägung „*Killed*“ erweitert wird.

Eine genaue Funktionalität des Job Manager Clients wird im nächsten Kapitel beschrieben.

## 5.4 Job Manager Client

Der Zugriff auf den Job Manager Client erfolgt über ein Web-Interface und dient zur Verwaltung von Jobs bzw. zur Administration der Benutzer. Damit der Job Manager Client benutzt werden kann, ist eine erfolgreiche Authentifizierung des Benutzers mit Kennung und Passwort Voraussetzung.

### 5.4.1 Job-Management

Der Benutzer hat zwei Möglichkeiten der Job-Abfrage:

- Die Jobs des eingeloggten Benutzers werden aufgelistet.
- Sämtliche Jobs von Benutzern, für die der eingeloggte Benutzer das Recht zum Administrieren hat, werden aufgelistet.

Eine spezifische Anzeige der aufgelisteten Jobs ist möglich. Jobs können aufgrund folgender Auswahlkriterien angezeigt werden:

- Jobs von bestimmten Benutzern (für den Fall, dass der Benutzer Jobs anderer Personen administrieren darf)
- Status eines Jobs
- Service URI

Die angezeigten Jobs können auf Grund folgender Attribute sortiert werden:

- Job ID
- Service URI
- Status
- Extended Status

Für jeden aufgelisteten Job kann, abhängig vom aktuellen Status,

- der Start durchgeführt werden,

- der aktuelle Status ermittelt werden,
- ein noch laufender Job abgebrochen werden,
- das Herunterladen der Ergebnisse durchgeführt werden oder
- eine Löschung der Job-Daten aus der Datenbank vollzogen werden.

Ein manuelles Abspeichern eines Jobs bzw. seiner Daten wird ebenfalls ermöglicht.

### 5.4.2 User-Administration

Der Benutzer kann sein Profil editieren. Er hat die Möglichkeit

- sein Passwort zu ändern,
- eine Änderung seiner Benutzerdaten wie Name, Email-Adresse, IMEI eines MIDs, etc. durchzuführen.

Hat der Benutzer das Recht zur Administration, so kann er folgende Aktivitäten durchführen:

- sämtliche im Job Manager Data Service gespeicherten Benutzer auflisten, bei denen er als Administrator eingetragen ist. Weiters kann der Benutzer diese Profile sowohl editieren als auch löschen.
- neue Benutzer anlegen

Der Mobile Job Manager Client ermöglicht ebenso wie der Job Manager Client die Statusabfrage und den Abbruch von Jobs.

## 5.5 Mobile Job Manager Client

Um den Mobile Job Manager Client zu nutzen, muss zuerst eine Anmeldung erfolgen. Für die Authentifizierung sind zwei Arten vorgesehen:

- Eingabe von Benutzerkennung und Passwort
- Authentifizierung durch die IMEI des MIDs

Nach erfolgreichem Einloggen werden die dem Benutzer zugeordneten, laufenden Jobs aufgelistet.

Für jeden aufgelisteten Job besteht nun die Möglichkeit, den aktuellen Status abzurufen oder einen Abbruch durchzuführen.

### 5.5.1 kSOAP

kSOAP ermöglicht den SOAP-Nachrichtenaustausch für Web Service Clients und wurde entwickelt, um in „eingeschränkten“ Java-Umgebungen, wie in Applets oder J2ME Applikationen (CLDC / CDC / MIDP), verwendet zu werden.

kSOAP ist OpenSource, die derzeit aktuellste Version (Version 2.1.2) wurde im März 2007 freigegeben. kSOAP wird gebündelt als Library zur Verfügung gestellt, und es stehen sämtliche Quellcodedateien zum Download bereit.

kSOAP wurde zur Kommunikation mit dem Job Manager Data Service verwendet.

Die Implementierung des Client-APIs ist in der folgenden Klassenübersicht dargestellt.

## 5.6 Klassenübersicht bzw. –beschreibung

Sämtliche Klassen und ihre Kollaborationen werden in diesem Kapitel vorgestellt. Folgende Aufstellung bietet einen Überblick über die Klassenbeschreibungen in den nächsten Abschnitten:

- Allgemeine Klassen & Interfaces, die sowohl vom Job Manager Client als auch dem Mobile Job Manager Client benutzt werden
- Job Monitoring-Komponente
- Job Manager Client
  - API für J2SE
  - Web-Servlet-Interface
- Mobile Job Manager Client
  - API für J2ME
  - MIDlet-Interface

### 5.6.1 Allgemeine Klassen & Interfaces

Allgemeine Klassen werden sowohl vom Job Manager Client als auch vom Mobile Job Manager Client benutzt.

Die Interfaces *DatabaseJobMapping*, *DatabaseUserMapping* und *DatabaseAdministrateUser-Mapping* (Abbildung 50) stellen die Zuordnung in den beiden Clients zum Job Manager Data Service bzw. zur Datenbank dar. Tabellenfelder der Datenbank-Tabellen *users*, *jobs*, *administrate_user* im Datenbankschema werden auf interne Namen abgebildet. Änderungen im Datenbank-Schema können zentral über diese Interfaces nachvollzogen werden.

*JobBase* und *UserBase* (Abbildung 51) bilden die Basis für die spezifischen Ableitungen von *Job*- und *User*-Klassen für den entsprechenden Client – Job Manager Client bzw. Mobile Job Manager Client.

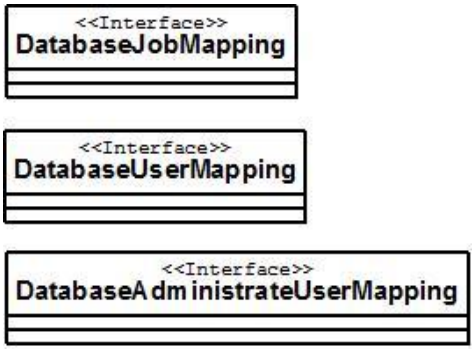



Abbildung 50: Database Mapping der Tabellen jobs, users & administrate_user

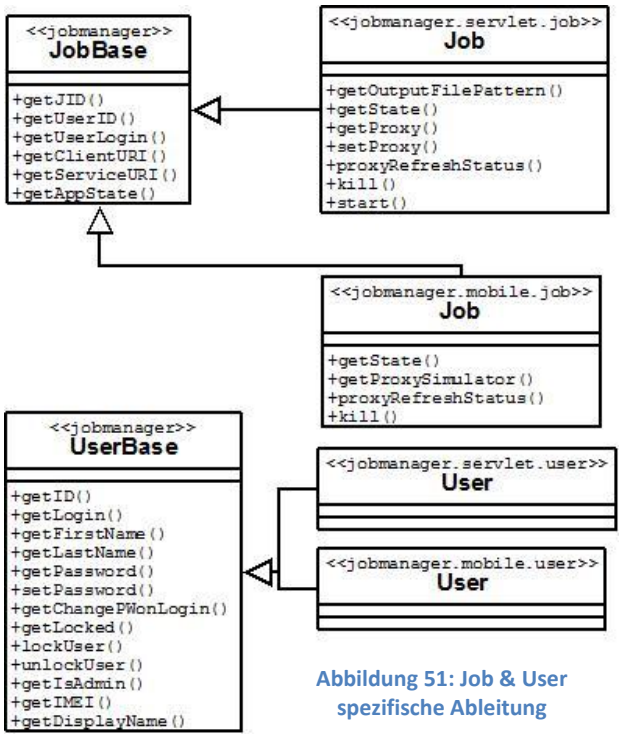



Abbildung 51: Job & User spezifische Ableitung

### 5.6.2 Job Monitoring-Komponente

Die Klasse *JobManagerGridEventListener* (Abbildung 52) implementiert das *GridEventListener*-Interface (siehe Kapitel 5.3 bzw. 3.4 Client Infrastruktur) des VGE-Client-Monitoring-APIs und reagiert auf die Ereignisse bzw. Aktionen des VGE-Clients „*Start*“, „*Upload*“, „*Download*“ und „*Kill*“. In Folge werden relevante Job-Daten mit dem Job Manager Data Service gespeichert oder aktualisiert.

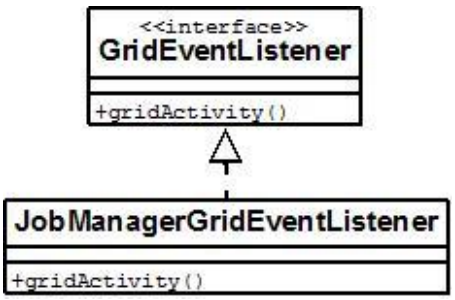



Abbildung 52: JobManagerGridEventListener der Job Monitoring-Komponente

### 5.6.3 Job Manager Client

Die Klassenbeschreibung für den Job Manager Client teilt sich in das API für J2SE und den Klassen für das Web-Servlet-Interface.

#### *API für J2SE*

Abbildung 53 zeigt die high-level-APIs *JobManager* und *UserManager* bzw. das low-level-API *DatabaseAgent* für den Job Manager Client. *JobManager* und *UserManager* benutzen den *DatabaseAgent* um Datenbank-Aktionen durchzuführen. Die Klassen *Job* und *User* entsprechen den bereits beschriebenen Entitäten im Kapitel „4.3.1 User“ und „4.3.2 Job“. Die Klasse *MD5* wird zur Kodierung des Benutzer-Passwortes mit dem MD5-Algorithmus verwendet. In der Datenbank sind die Passwörter mit MD5-Kodierung gespeichert.

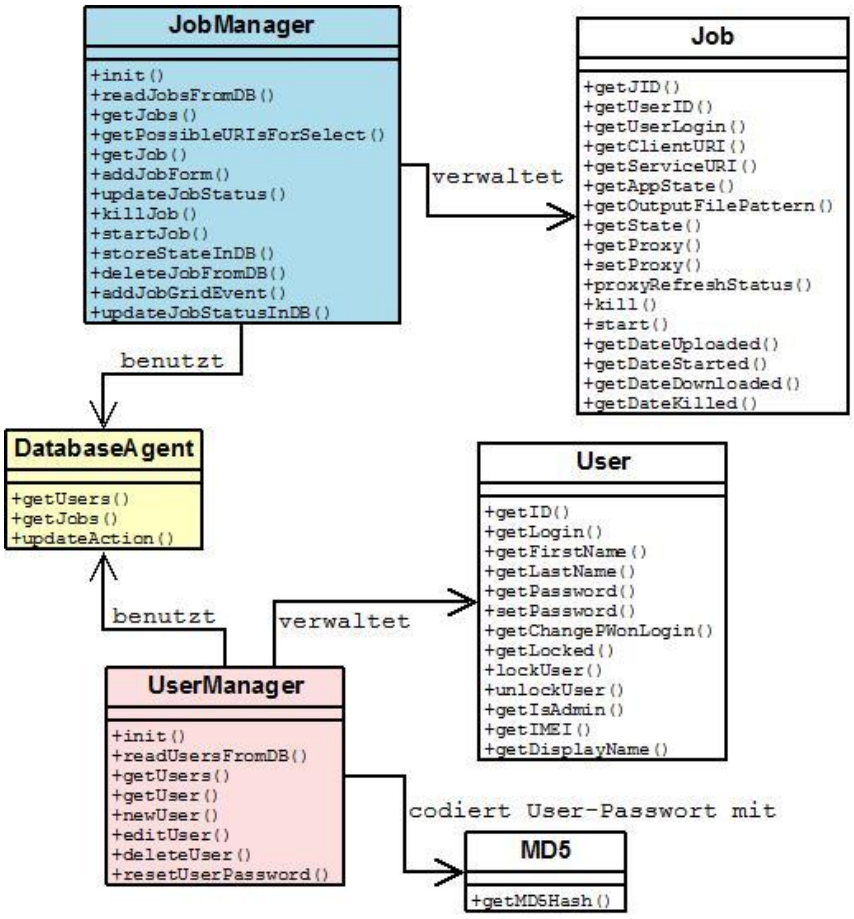



Abbildung 53: API Klassen für J2SE

Im Anhang befindet sich die JavaDoc für das high- and low-level-API in J2SE:

- JobManager: siehe Kapitel 9.2.1
- DatabaseAgent: siehe Kapitel 9.2.2

### *Web-Servlet „Job Manager Client“*

Abbildung 54 stellt die Benutzung des *JobManager* und des *UserManager* APIs durch das Web-Servlet dar. Die Klassen *ServletUtilities* und *LoadProperties* stellen allgemeine Methoden zur Unterstützung zur Verfügung. *MD5* wurde bereits im Kapitel „5.1.4 Data Resource - Realisierung mittels MySQL“ vorgestellt. Der Java Enum-Typ *ActionIndex* ist das Äquivalent zu der, mit JavaScripts implementierter Menü-Führung. Änderungen in den Java-Scripts des Web-Interfaces müssen hier nachgezogen werden, um ein synchrones Verhalten zwischen Servlet und JavaScripts zu gewährleisten.

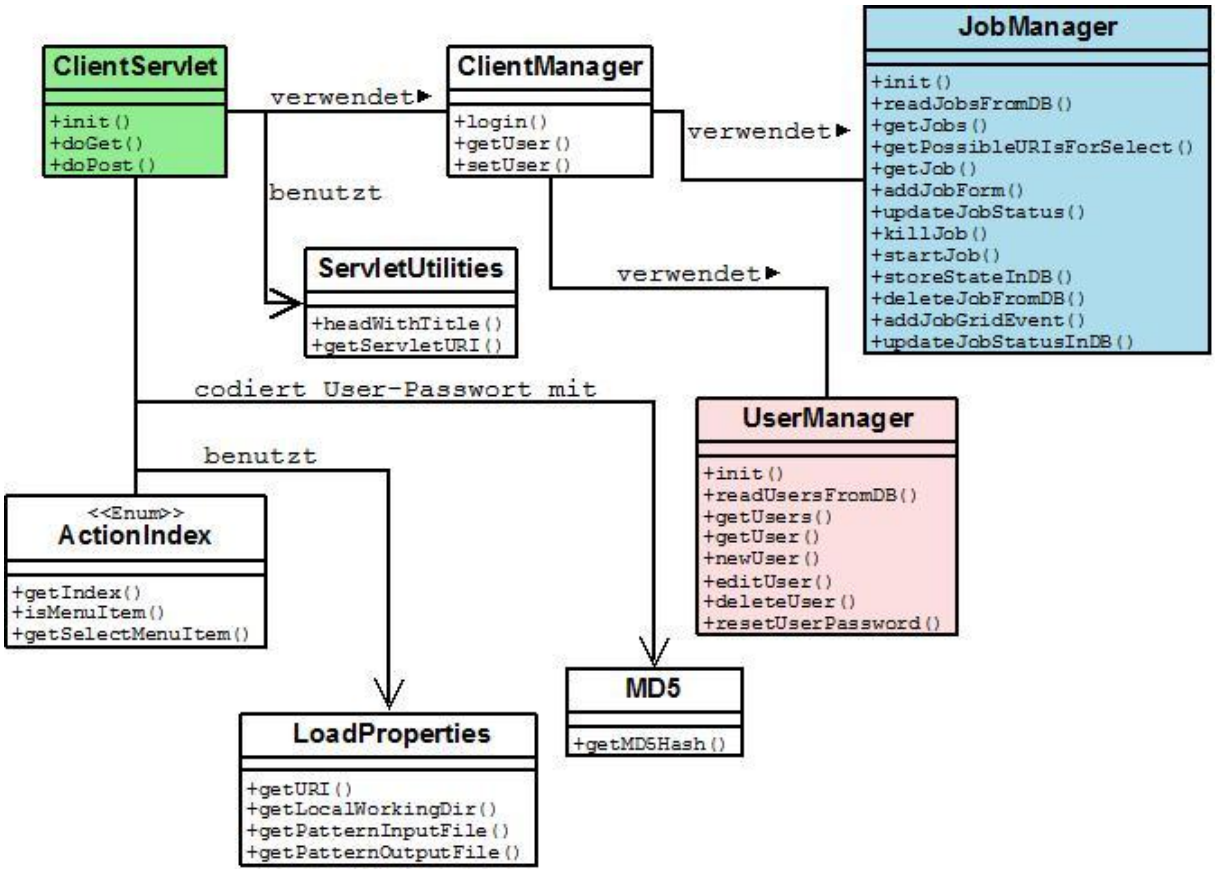



Abbildung 54: Web-Servlet Klassen

### 5.6.4 Mobile Job Manager Client

Die Klassenbeschreibung für den Mobile Job Manager Client teilt sich in das API für J2ME und die Klassen für das MIDlet.

#### *API für J2ME*

Abbildung 55 zeigt die high-level-APIs *JobManager* und *UserManager* bzw. das low-level-API *DatabaseAgent* für den Mobile Job Manager Client. Die Klasse *MD5* ist eine vollständige Implementierung des MD5-Algorithmus im Gegensatz zum Äquivalent im J2SE API, da für MIDlets keine MD5-Implementierung verfügbar ist. Analoges gilt für die *Base64*-Klasse.

Die Klasse *AppProxyState* ist das Äquivalent zum VGE Enum-Typ AppProxyState (`at.ac.univie.isc.client.proxy.AppProxyState`), da Enum-Typen in J2ME nicht unterstützt werden.

Der *DatabaseAgent* verwendet den *ProxySimulator* – wie der Name schon sagt, simuliert diese Klasse das AppProxy-API. Der *ProxySimulator* benutzt die Klasse *MyHttpTransport* zur Kommunikation mit dem Job Manager Data Service. *MyHttpTransport* ist für das Versenden und Empfangen von SOAP-Nachrichten zuständig und greift auf die im Kapitel „5.5.1 kSOAP“ beschriebene Bibliothek zurück

bzw. auf die abgeleiteten Klassen *MySoapEnvelope* und *MySoapWriter*. *MySoapEnvelope* und *MySoapWriter* sind von den entsprechenden Klassen abgeleitet, um SOAP-Nachrichten im MIME-Format zu übertragen, da kSOAP keine Attachments unterstützt.

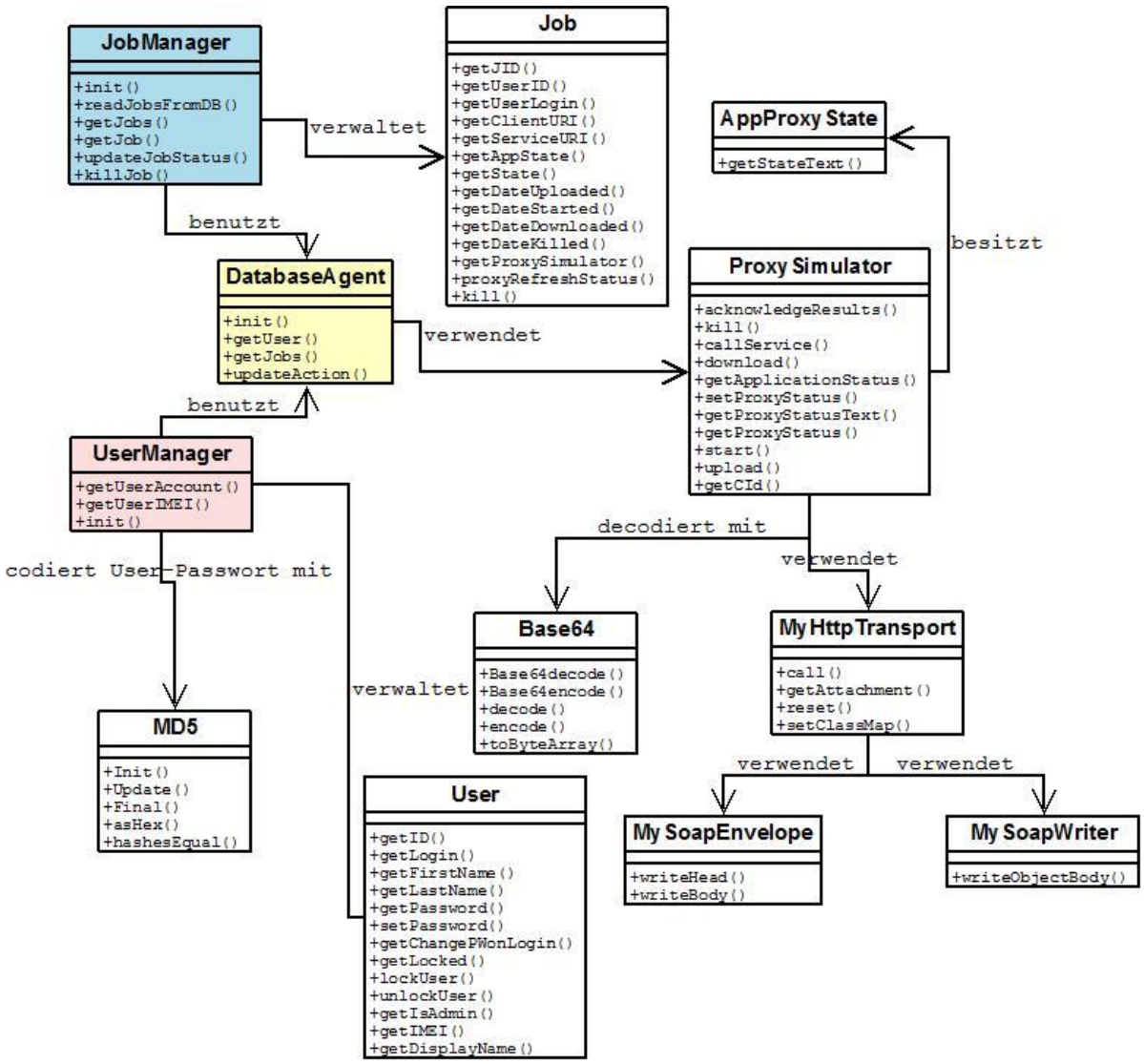



Abbildung 55: Klassenübersicht des Mobile Job Manager APIs

Im Anhang befindet sich die JavaDoc für das high- and low-level-API in J2ME:

- JobManager: siehe Kapitel 9.2.3
- DatabaseAgent: siehe Kapitel 9.2.4

### *MIDlet „Mobile Job Manager Client“*

Abbildung 56 stellt die Klassen des MIDlet-Interfaces dar. Ausgangspunkt in der Klassenhierarchie ist die Klasse ClientMidlet, abgeleitet von der J2ME MIDlet-Klasse.

*ClientManagerThread* und *StatusGaugeThread* sind jeweils von der Thread-Klasse abgeleitet. Sämtliche IO-Kommunikation muss in MIDlets in einem eigenen Thread abgewickelt werden. Im

Kontext des Mobile Job Manager Clients wird daher der Versand und Empfang von SOAP-Nachrichten durch den *ProxySimulator*, in weiterer Folge im *ClientManagerThread* abgewickelt. Der *StatusGaugeThread* ist für die Aktualisierungen der Fortschrittsanzeige (Gauge) während des SOAP-Nachrichtenversandes und –empfangens zuständig.

Die Klasse *Settings* ermöglicht das Abspeichern der Einstellungen des Mobile Job Manager Clients (URI des Job Manager Data Services, Benutzer-Kennung, etc.) durch die Benutzung von RecordStores. RecordStores ermöglichen die permanente Speicherung von Daten in mobilen Endgeräten.

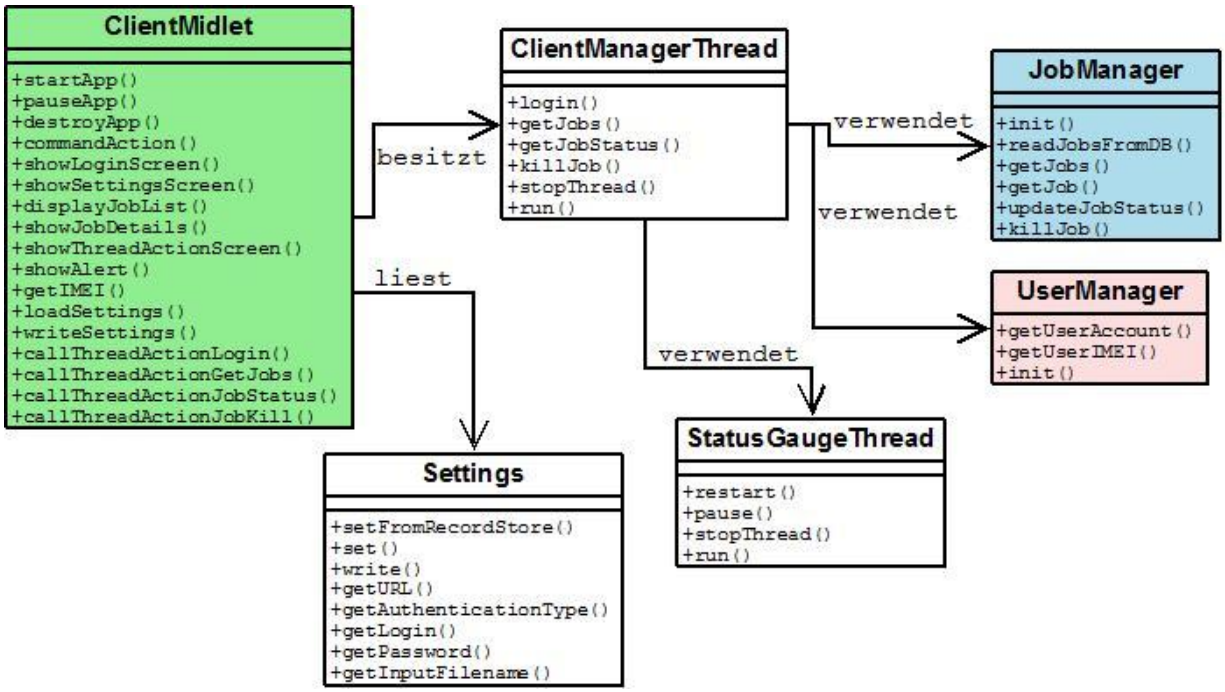



Abbildung 56: Klassenübersicht des Mobile Job Manager Interfaces

## 5.7 Client-Entwicklung

Der Erweiterung der VGE-Software durch den Job Manager wurde in einem eigenen Paket realisiert.

### 5.7.1 Job Manager Paket

Abbildung 57 stellt eine Übersicht des Job Manager-Paketes (`at.ac.univie.isc.jobmanager`) dar.

Das Sub-Paket `mobile` enthält sämtliche Klassen für den Mobile Job Manager Client. Das Sub-Paket `servlet` enthält sämtliche Klassen für den Job Manager Client.

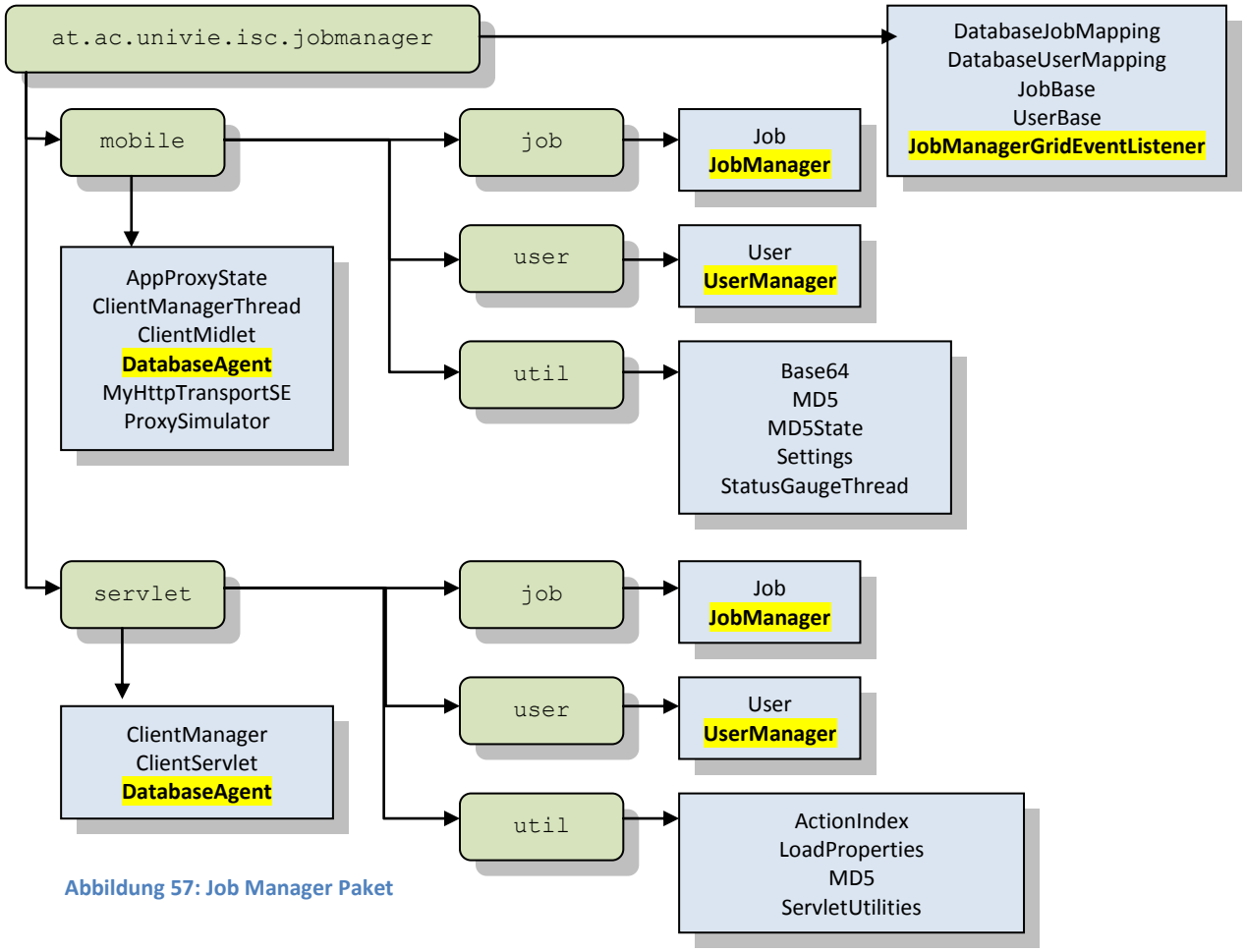



Abbildung 57: Job Manager Paket

Für die Realisierung des Mobile Job Managers wurden zwei Klassen von der kSOAP-Bibliothek (`org.ksoap`) abgeleitet, da kSOAP keine Attachments unterstützt:

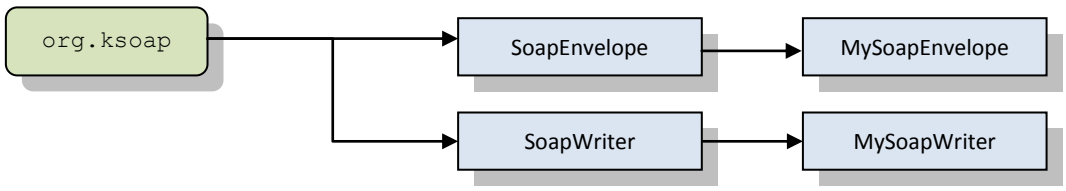



Abbildung 58: kSOAP Paket

### 5.7.2 Client-Entwicklungsumgebung für den Job Manager Client

Die ANT-basierte Entwicklungsumgebung wurde bzgl. des Targets „`jobmanager`“ erweitert.

Die Produktion sämtlicher Klassen ist somit durch

```
ant dev -Ddev.root=jobmanager
```

möglich.

### 5.7.3 MIDlet Mobile Job Manager Client

In diesem Kapitel werden einige Besonderheiten der Entwicklung des MIDlets „Mobile Job Manager Client“ angeführt.

Für die Realisierung des Mobile Job Managers wurde die im Kapitel 5.5.1 erläuterte kSOAP Bibliothek zur Kommunikation mit dem Job Manager Data Service verwendet. Da kSOAP keine Attachments unterstützt, wurden die entsprechenden Klassen mit eigenen überschrieben, um deren Übertragung zu ermöglichen.

Als Plattform wurde JTWI[38] gewählt bzw. die Konfiguration CLCD v1.1 mit dem Profil MIDP v2.0.

Das Web-Interface fordert Daten vom Job Manager Data Service im XML-Format an, d.h. das OGSA-DAI Perform-Dokument beinhaltet als Aktivität „*sqlResultsToXML*“.

Das MIDlet hingegen fordert die Daten im CSV-Format an („*sqlResultsToCSV*“), um die zu übertragende Datenmenge zu optimieren. Aus Sicht der Datenübertragung sprechen aufgrund des einfachen Formats der Daten keine Gründe dagegen. Von J2ME wird ein XML-Parser (JAXP XML Parser, JSR 172; siehe „Abbildung 25: API Selektion im Sun Java Wireless Toolkit (Screenshot)“) als zusätzliches Paket für die Auswertung von XML-Daten angeboten. Die Übertragung im XML-Format hätte zwar zur Folge, dass für die Auswertung des Perform-Dokumentes für das Web-Interface und das MIDlet die gleiche Methode benutzt werden kann. Der zusätzliche Implementierungsaufwand wiegt aber die deutlich geringere Datenübertragungsmenge auf.

Beispiel:

Es wurden drei Abfragen der Jobs des Benutzers mit der Kennung „*fab*“ durchgeführt.

Das Ergebnis waren 5, 8 bzw. 11 Datensätze – das Ergebnis im CSV-Format:


```
"jid"#"user_id"#"login"#"client_uri"#"service_uri"#"state"#"date_uploaded"#"date_started"#"date_downloaded"#"date_killed"
"fab1219278136744-81"#13#"fab"#"http://localhost:9003/gsclient/http2soap"#"http://localhost:9001/HelloWorld/scs"#2#"2008-08-21 02:22:17.0"#"2008-08-21 02:22:45.0"#"NULL"#"NULL"
"fab1222106305099-83"#13#"fab"#"http://localhost:9003/gsclient/http2soap"#"http://localhost:9001/HelloWorld/scs"#2#"2008-09-22 19:58:25.0"#"2008-09-22 19:59:00.0"#"NULL"#"NULL"
      :      :      :      :
```


[38] Java Technology for the Wireless Industry (JTWI), Java Specification Request (JSR) 185

Die Analyse der reinen Datenmenge dieser 5, 8 bzw. 11 Datensätze im XML- und CSV-Format brachte folgendes Ergebnis (ausgewertet mit TCPMonitor):

| Daten im | Anzahl der Datensätze bzw. Datengröße in Zeichen | | | | | |
|---|---|---|---|---|---|---|
| CSV-Format | 5 | 1016 | 8 | 1579 | 11 | 2143 |
| XML-Format | | 9594 | | 11003 | | 12392 |

Zu beachten ist der deutliche Overhead des OGSA-DAI Response-Dokumentes im XML-Format aufgrund der Metadaten. Der prozentuelle Anteil nimmt zwar mit steigender Datensatzanzahl ab, vor allem aber bei wenigen Datensätzen beträgt der Overhead einen beträchtlichen Anteil. Da der Mobile Job Manager Client NUR die eigenen, laufenden Jobs des Benutzers auflistet, ist dies häufig nur eine geringe Anzahl an Datensätzen.

Da das inkludierte Datenvolumen in Verträgen mit Mobil-Netzbetreibern in den meisten Fällen noch gering ist (mein Vertrag beinhaltet ein Datenvolumen von einem Megabyte), bringt eine Übertragung im CSV-Format einen wesentlichen Vorteil.

Die Entwicklungsumgebung für das MIDlet wird im Client-Provisioning-Kapitel „5.8.4 MIDlet „Mobile Job Manager Client““ beschrieben.

Im nächsten Kapitel wird das Client-Provisioning für den Job Manager beschrieben.

## 5.8 Client-Provisioning

Der Client-Provisioning Prozess unterstützt sowohl das automatisierte Installieren des Web-Interfaces „Job Manager Client“ (analog eines VGE-Web-Clients) als auch das Hinzufügen der Job Monitoring-Komponente zu einem VGE-Web-Client. Das Client-Provisioning-Environment wurde adaptiert, um die Erweiterung des VGE durch den Job Manager in den herkömmlichen Provisioning-Prozess einzubinden.

### 5.8.1 JobManager-Property Datei

Sowohl die Job-Monitoring-Komponente als auch die Web-Interface Komponente greifen im Zuge der ANT-basierten Installation auf die Properties-Datei „*jobmanager.properties*“ zu.

Die vier wichtigsten Parameter sind:

| | |
|---|---|
| `jobmanager.uri` | `http://localhost:9002/JobManager/scs/`<br>URI des Job Manager Data Services |
| `jobmanager.local.working.dir` | `e:\\Temp\\inputFile_DataService\\JobManager\\`<br>Lokales Arbeitsverzeichnis für das Hoch- und Herunterladen der Ein-/Ausgabedatei zur Nutzung des Job Manager Data Services |
| `jobmanager.pattern.input.file` | `inputFile.txt`<br>Das Job Manager Data Service wird mit diesem Namen oder Pattern für die Eingabedatei(en) installiert. |
| `jobmanager.pattern.output.file` | `outputFile.txt`<br>Das Job Manager Data Service wird mit diesem Namen oder Pattern für die Ausgabedatei installiert. |

Die restlichen Parameter sind analog einer Properties-Datei eines VGE-Clients.

### 5.8.2 Job Montioring-Komponente

Die Job Monitoring Komponente ist eine optionale funktionale Erweiterung von VGE-Clients. Eine Installation erfolgt wie andere Client-Komponenten durch Hinzufügen der Komponente „*JobMonitor*“ in der Client-Properties-Datei (*client.components=ApplicationClient,**JobMonitor***).

### 5.8.3 Web-Interface „Job Manager Client“

Das Web-Interface kann wie ein VGE-Web-Client, ANT-basiert installiert werden:

```
ant prov.jobmanager.install
```

### 5.8.4 MIDlet „Mobile Job Manager Client“

Das manuelle Provisioning des Mobile Job Manager Clients umfasst folgende Schritte:

1. Mit dem „Sun Java Wireless Toolkit“ wird ein neues Projekt angelegt.
2. Als Plattform wurde JTWI[39] bzw. die Konfiguration CLCD v1.1 mit dem Profil MIDP v2.0 gewählt.

[39] Java Technology for the Wireless Industry (JTWI), Java Specification Request (JSR) 185

3. Folgende Dateien werden in das angelegte Projekt transferiert:
    - das Sub-Paket für den Mobile Job Manager Client
      `at.ac.univie.isc.jobmanager.mobile`
    - allgemeine Klassen und Interfaces unter
      `at.ac.univie.isc.jobmanager`
    - die abgeleiteten Klassen von kSOAP (um Attachments zu übertragen):
      `org.ksoap.MySoapEnvelope & org.ksoap.MySoapWriter`
    - die kSOAP-Bibliothek in das Bibliotheken-Verzeichnis
4. Es wird ein Paket erzeugt.
5. Dieses Paket wird mit USB, BlueTooth oder „over-the-air“ von einem Web Server übertragen. Für die Übertragung von einem Web Server, im speziellen von einem Tomcat, sind folgende Schritte notwendig:
    - JAD-Datei und das Paket auf den Tomcat-Server kopieren
    - erzeugen einer HTML-Seite mit einem Link auf die JAD-Datei
    - falls nicht vorhanden, hinzufügen eines neuen MIME-Types in der `<tomcat-dir>\conf\web.xml`-Datei:

```
<mime-mapping>
    <extension>jad</extension>
    <mime-type>text/vnd.sun.j2me.app-descriptor</mime-type>
</mime-mapping>
```

Die Screenshots im nächsten Kapitel zeigen einige exemplarische Aktionen mit den Interfaces.

## 5.9 Screenshots des Job Manager Clients & Mobile Job Manager Clients

In den nächsten beiden Kapiteln sind Screenshots des Job Manager Clients und des Mobile Job Manager Clients abgebildet.

### 5.9.1 Screenshots des Job Manager Clients

Der Screenshot in der Abbildung 59 zeigt die Jobs des eingeloggten Benutzers.

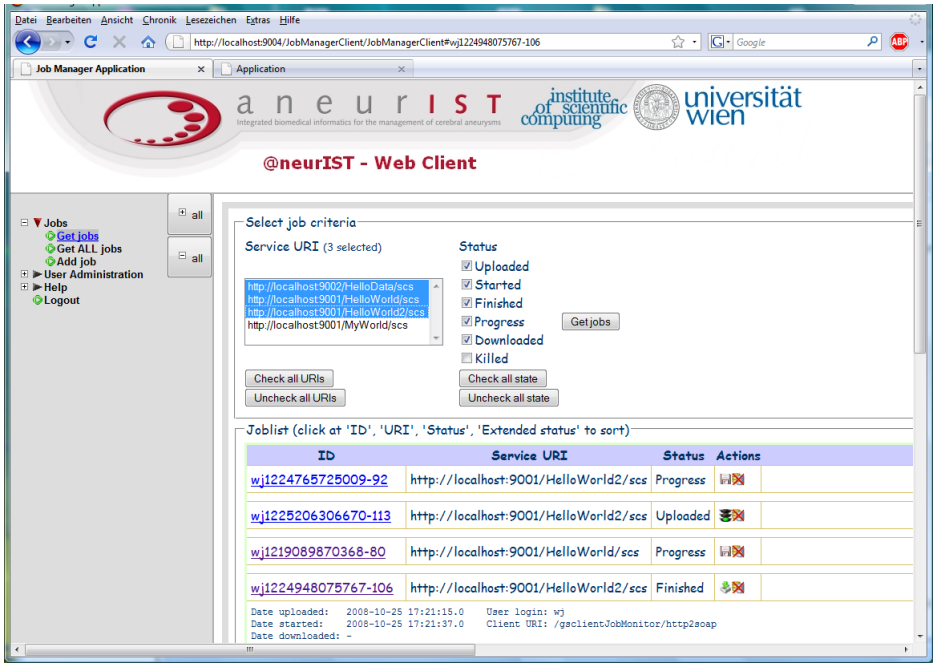



Abbildung 59: Jobs des eingeloggten Benutzers

Die Jobs aller Benutzer, für die der eingeloggte Benutzer die Berechtigung zur Administration hat, werden in Abbildung 60 gezeigt.

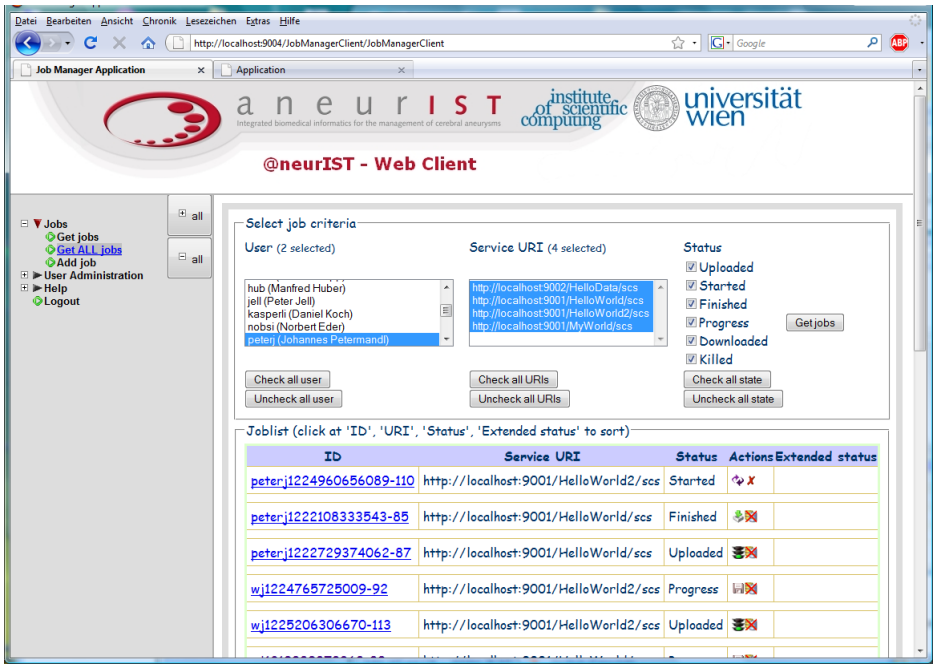



Abbildung 60: Jobs aller zur Administration berechtigten Benutzer

Die Benutzer-Verwaltung ermöglicht das Editieren, Löschen und das Ändern von Passwörten der Benutzer (siehe Abbildung 61).

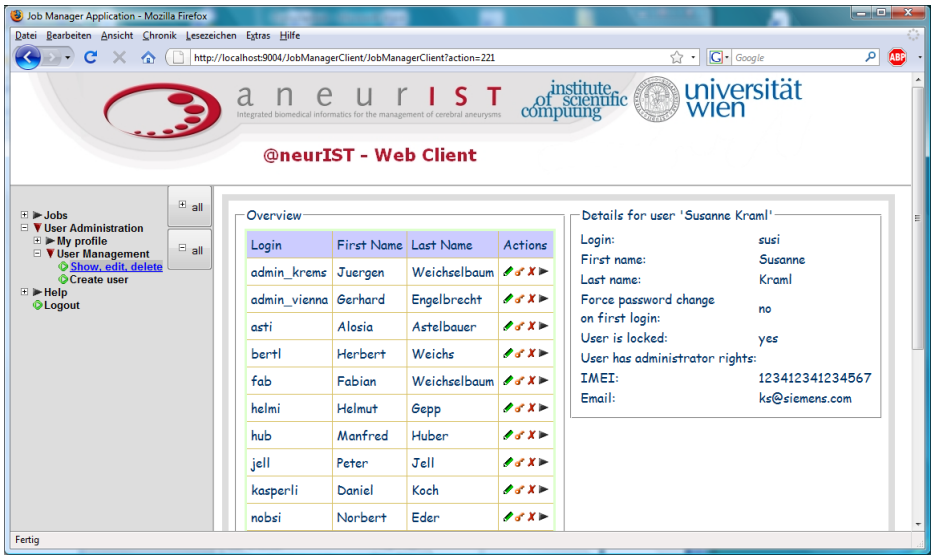



Abbildung 61: Benutzer-Verwaltung

Abbildung 62 zeigt das Benutzer-Profil im Zuge des Anlegens eines neuen Benutzers.

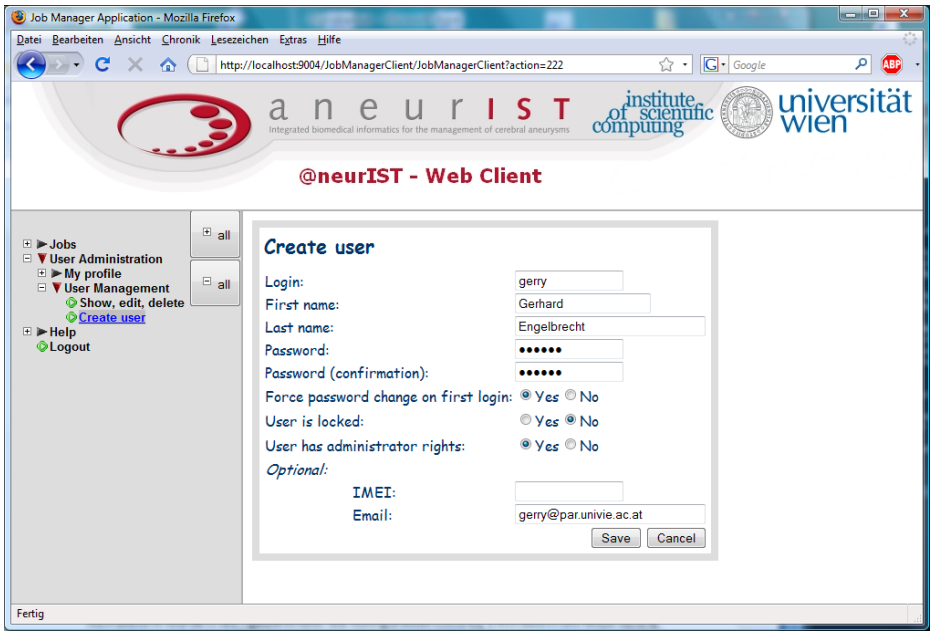



Abbildung 62: Benutzer-Profil

### 5.9.2 Screenshots des Mobile Job Manager Clients

Der LoginScreen des Mobile Job Manager Clients ist in Abbildung 63 zu sehen.

Einstellungen des Mobile Job Manger Clients, wie die URI des Job Manager Data Services, werden in Abbildung 64 gezeigt.

Die Liste laufender Jobs eines Benutzers ist in Abbildung 65 dargestellt.

Details zu einem Job werden in Abbildung 66 gezeigt.

Abbildung 63: Login-Screen des Mobile Job Manager Clients

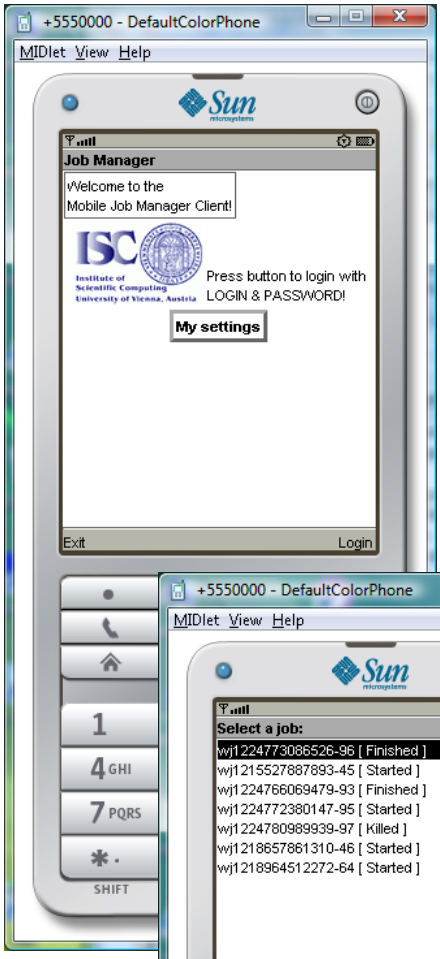
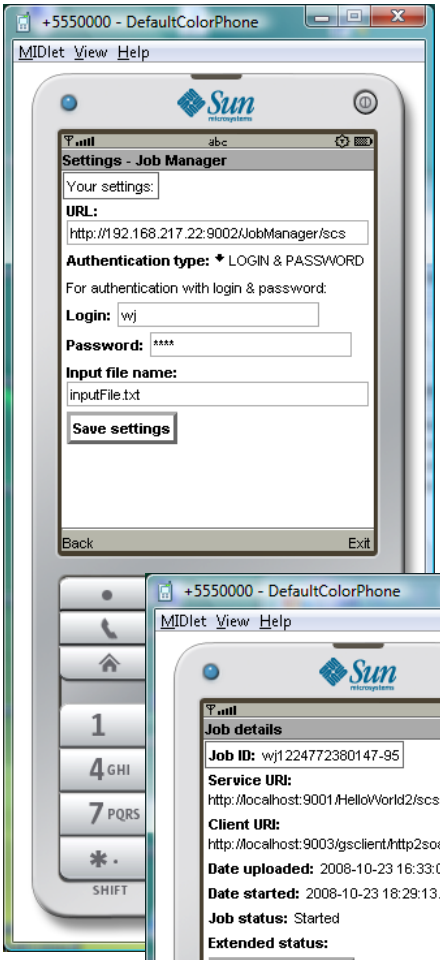



Abbildung 64: Einstellungen des Mobile Job Manager Clients



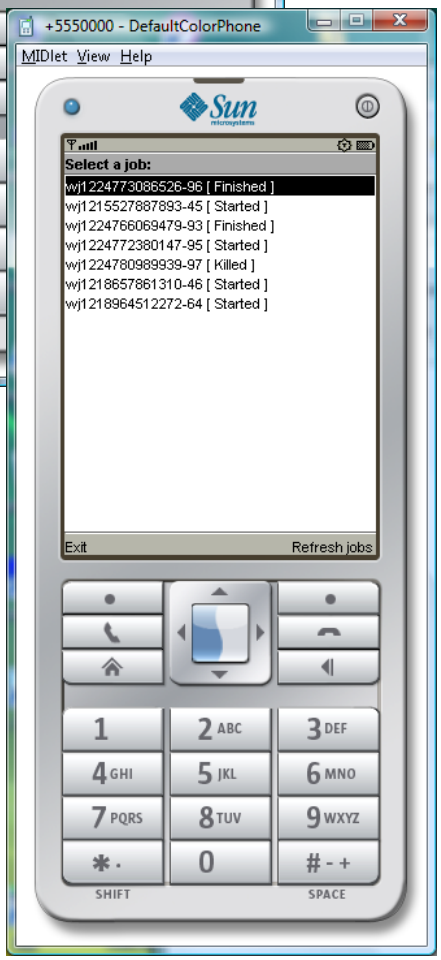




Abbildung 65: Laufende Jobs eines Benutzers

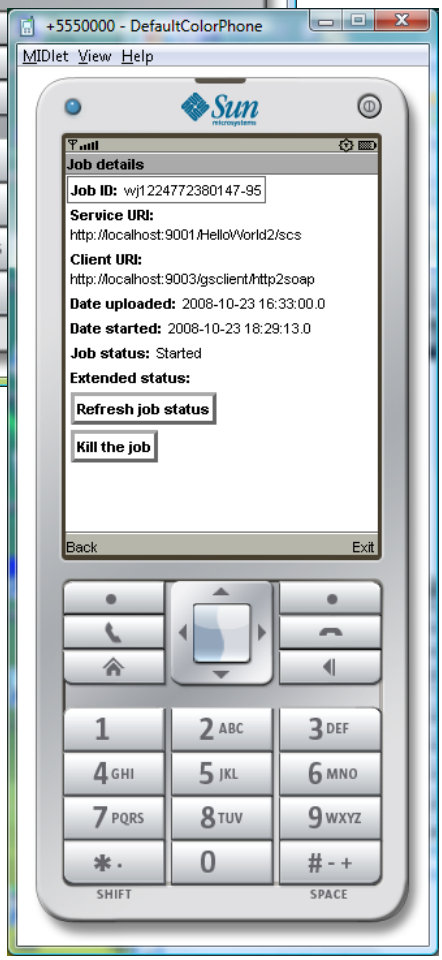



Abbildung 66: Job-Details

## 5.10 Mobile Job Manager Client-Tests

Tests des Mobile Job Manager Clients wurden einerseits mit einem Emulator, anderseits mit einem mobilen Endgerät durchgeführt. Zusätzlich wurde der TCPMonitor zur Analyse der Kommunikation eingesetzt.

### 5.10.1 Sun Java Wireless Toolkit

Zum Testen des Mobile Job Manager Clients wurde das „Sun Java Wireless Toolkit“ verwendet, welches zur Emulation der CLDC-Umgebung dient. Das Toolkit bietet die Möglichkeit die Ausführungsumgebung des MIDlets individuell den Bedürfnissen anzupassen. Außerdem werden diverse Tools wie Memory & Network Monitor, Stub Generator, Resources Manager, etc. zur Unterstützung angeboten.

Als Plattform wurde JTWI[40] gewählt bzw. die Konfiguration CLCD v1.1 mit dem Profil MIDP v2.0.


**Abbildung 67: LG U8630 mit Mobile Job Manager Client**

### 5.10.2 LG U8630

Mit dem Java Wireless Toolkit wurde ein Paket erzeugt und auf einem LG U8630 installiert, um den Mobile Job Manager Client mit realen Equipment zu testen. Dies hatte zur Folge, dass weitere Adaptionen notwendig waren.

### 5.10.3 TCPMonitor

Der TCPMonitor ist ein Teil der AXIS Software und kann zur Analyse der Kommunikation z.B. zwischen Client und Service verwendet werden. Im Zuge der Diplomarbeit wurde der TPCMonitor zur Analyse der Kommunikation zwischen dem MID mit dem Mobile Job Manager Client und dem Job Manager Data Service eingesetzt.

In Abbildung 68 ist die Funktionsweise des TCPMonitors ersichtlich. Vom MID wird ein SOAP RPC-Request auf die Netzwerkadresse des Web Servers, jedoch mit dem Port 8002 geschickt. Der TCP-Monitor registriert alle eingehenden Nachrichten auf dem Port 8002, zeigt diese im GUI an und leitet sie auf den Port 9002 um – den Port, auf dem der Web Server mit dem Job Manager Data Service installiert ist. Der Verlauf der Response-Nachricht ist analog des Requests. Das Service schickt den

[40] Java Technology for the Wireless Industry (JTWI), Java Specification Request (JSR) 185

SOAP-RPC-Response an den Sender – in diesem Fall, den TCP-Monitor - zurück. Der TCPMonitor zeigt auch die Antwortnachricht an und leitet sie zum MID um.

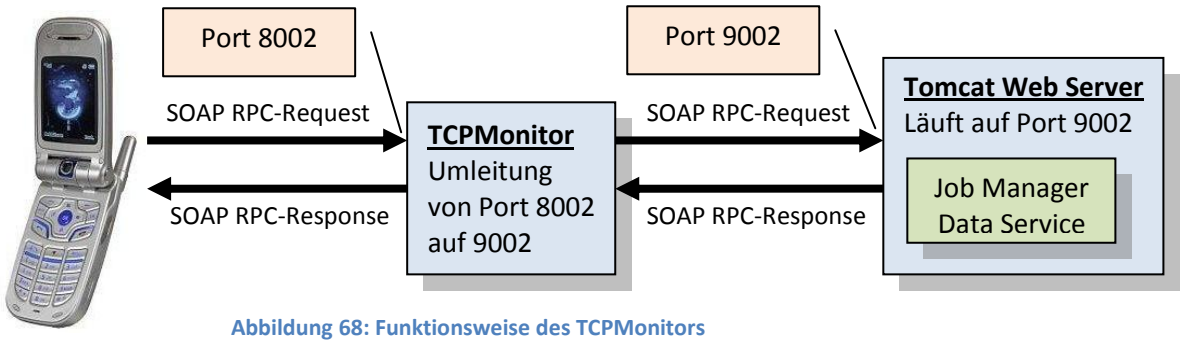



Abbildung 68: Funktionsweise des TCPMonitors

In Abbildung 69 ist der TCPMonitor mit der Umleitung von Port 8002 auf 9002 dargestellt. Die ausgewählte Nachricht ist der RPC der Methode „*downloadAttachment*“.

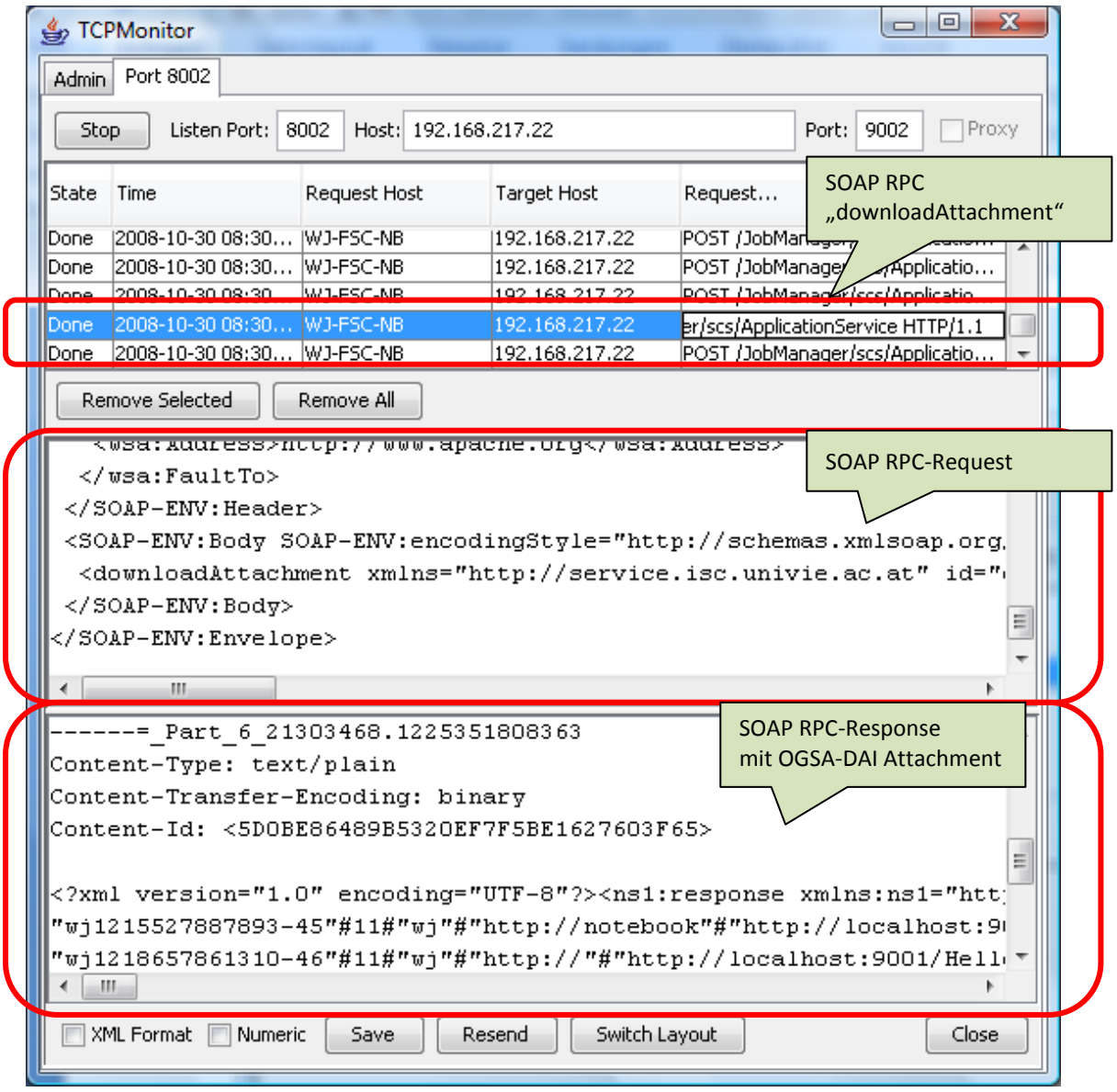



Abbildung 69: Kommunikation zwischen MIDlet und Job Manager Data Service im TCPMonitor

# 6 Prakt. Einsatzgebiete

Der Job Manager ist als optionale Client-Komponente entworfen worden. Abhängig von der Infrastruktur des VGE-Systems (Anzahl bzw. Art der Clients, Anzahl der Services) wird es sinnvoll sein, den Job Manager bzw. die Job Monitoring-Komponente teilweise oder überall einzusetzen.

Im folgenden Kapitel werden mögliche Anwendungsfälle erläutert, die als Entscheidungshilfe für den Einsatz in bestehenden Projekten dienen.

## 6.1 Anwendungsfälle / Einsatz in bestehenden Projekten

Der Einsatz des Job Managers ist überall dort sinnvoll, wo Abhängigkeiten zwischen verschiedenen Projekten gegeben sind. Aktivitäten bzw. Jobs beider Projekte können gebündelt überwacht und gesteuert werden.

Die Illustration von Jobs, die auf verschiedenen Clients gestartet wurden, die Archivierung von Job-Daten bzw. die Analyse dieser Daten zur Gewinnung eines Überblicks sind weitere Anwendungsgebiete. Mögliche Szenarien dazu wurden bereits im Kapitel 4.1 bzgl. des Job Manager Zieles angeführt.

Ein zusätzlicher Anwendungsfall, der nicht dem eigentlichen Verwendungszweck des Job Managers entsprechen würde, wäre eine Authentifizierung für bestehende VGE-Clients. Derzeit erfolgt eine Benutzerüberprüfung aufgrund lokaler Daten. Die Interfaces des Job Managers setzen eine zwingende Authentifizierung des Benutzers voraus. Bestehende Clients könnten nun statt der Überprüfung mittels lokaler Benutzerdaten nun das User-Management des Job Managers verwenden. Eine zentrale Benutzerverwaltung wird somit im VGE-System ermöglicht.

## 6.2 Einsatz im Columbus Projekt

Der Job Manager wurde im Columbus Projekt getestet.

Jobs zur Berechnung der molekularen Elektronenstruktur wurden gestartet und mit dem Job Manager mitprotokoliert. Der Status der Berechnungen wurde dann sowohl mit dem Job Manager Client als auch mit dem Mobile Job Manager Client überwacht. Ebenso wurden laufende Berechnungen mit beiden Clients abgebrochen. Abschließend wurde noch das Herunterladen der Ergebnisse mit dem Job Manager Client erfolgreich getestet.

# 7 Resümee

Rückwirkend werden einige Bereiche dieser Diplomarbeit betrachtet.

### *Aufgabenstellung*

Das Ziel der Diplomarbeit war, einen Client zu realisieren, der die clients- und orts-unabhängige Verwaltung laufender und abgeschlossener Jobs eines eindeutig identifizierbaren Benutzers ermöglicht. Dieser Client wurde einerseits als Web-Interface anderseits als Client für mobile Endgeräte realisiert.

### *Verwendete Werkzeuge*

Für die Implementierung der Clients wurde Eclipse verwendet. Mit Hilfe des „EclipseMe“ Plug-ins kann Eclipse auch zur Entwicklung von J2ME MIDlets verwendet werden. Zum Testen des MIDlets wurde das „Sun Java Wireless Toolkit“ verwendet, welches zur Emulation der CLDC-Umgebung dient. Das Toolkit bietet zwar keine Möglichkeit zum Editieren von Quellcodes, jedoch kann die Ausführungsumgebung des MIDlets individuell den Bedürfnissen angepaßt werden. Außerdem werden diverse Tools zur Unterstützung angeboten.

Der TCPMonitor wurde für die Analyse der Request & Response Nachrichten zwischen Mobile Job Manager Client und Job Manager Data Service eingesetzt.

Außerdem wurde der Mobile Job Manager nicht nur mit dem „Sun Java Wireless Toolkit“ Emulator, sondern auch mit dem Mobiltelefon LG U8360 getestet. Die Tests mit einem echten Equipment führten zu weiteren Adaptierungen der Software, da der Emulator doch in manchen Bereichen „großzügiger“ war.

### *Vorgangsweise*

Zuerst wurden die Anforderungen an einen Job Manager untersucht und mögliche Lösungsmöglichkeiten analysiert. Zur Realisierung wurde das Client-seitige Monitoring gewählt, welches als Grundlage für den Job Manager dient. Als nächstes wurde ermittelt, welche Daten für den Job Manager relevant sind und persistent verfügbar sein sollen. Anhand der ermittelten Daten wurde das Datenbank-Design entworfen. Anschließend wurde ein API entworfen und die Job Manager-Funktionalität implementiert. Mit der Umsetzung des Web-Interfaces war die Realisierung des „Job Manager Clients“ abgeschlossen und eine Benutzung des Job Managers möglich.

Für die Realisierung des Mobile Job Manager Clients wurde versucht, das bestehende API auf die eingeschränkte Einsatzumgebung in mobilen Endgeräten mit möglichst wenigen Adaptierungen zu portieren. Die Einschränkungen in der J2ME-Umgebung führten jedoch dazu, dass der Mobile Job Manager Client größtenteils als neue Implementierung des Job Manager Clients entstanden ist.

### *Erweiterung*

Mögliche Ansatzpunkte für weitere Arbeiten wären die Untersuchung von Authentifizierungs und Autorisierungsmöglichkeiten in mobilen Endgeräten. Derzeit findet eine Authentifizierung nur mit Kennung und Passwort des Benutzers statt. Da das VGE diverse Sicherheitskonzepte z.B. Zertifikate bietet, wäre eine Analyse der Möglichkeiten in mobilen Endgeräten ein Ansatzpunkt für weitere Arbeiten.

Eine Erhöhung der Benutzerfreundlichkeit würde durch eine automatische Benachrichtigung z.B. per Email oder SMS bei Beendigung eines Jobs erzielt werden. Dazu müsste jedoch, von der als Service ausgeführten Applikation, der Impuls kommen, bei Fertigstellung eine Service-Operation zur Benachrichtigung auszuführen. Dies widerspricht jedoch wieder der Service-orientierten Architektur des VGE, indem sämtliche Impulse client-seitig ausgehen. Eine mögliche Lösung wäre eine regelmäßige Überprüfung der Jobs durch den Client in bestimmten Intervallen, um den Status laufender Jobs zu ermitteln. Dieses resultiert jedoch wieder in zusätzlicher Netzwerk- und CPU-Last.

Das Einsetzen des Service-seitigen Monitorings wäre vor allem dann sinnvoll, wenn nicht auf allen Clients das Job Monitoring hinzugefügt wird. Ein Überwachen der Aktivitäten bzgl. bestimmter Services wäre dadurch möglich. Dazu müßten aber bestehende und vor allem laufende Services adaptiert werden. Vom Prinzip her wäre kein Unterschied, nur der Initiator für das Job Monitoring wäre ein anderer.

### *Fazit*

Der entstandene Job Manager bietet die Möglichkeit, auf einfache Weise Jobs in einem Grid-System zu zentralisieren bzw. zu visualisieren und gemeinsam zu verwalten. Zusätzlich wurde eine Benutzerverwaltung realisiert, durch die eine zentrale Authentifizierung der Benutzer für diverse Clients ermöglicht wird. Eine Herausforderung lag ebenfalls in der Realisierung des Mobile Job Manager Clients, der den Zugriff auf Jobs im VGE-System durch mobile Endgeräte ermöglicht. Eine Administration der Jobs im VGE-System ist nun „over-the-air“ möglich.

# 8 Samples/Listings:

## 8.1 SOAP-Listings

### 8.1.1 SOAP-/HTTP-Request und Response einer VGE-Client/Service-Kommunikation

In „Listing 11: SOAP-/HTTP-Request und Response einer VGE-Client/Service-Kommunikation“ sieht man einen Remote Procedure Call (RPC), der die Methode „getCId“ aufruft.

- Der HTTP-Header ist blau hinterlegt und umfasst die Zeilen 1 – 11.
    - Zeile 1 beinhaltet die Ressource bzw. das Web Service, auf das zugegriffen wird.
    - Zeile 2 gibt an, welche Art von Dokument der Request beinhaltet. Dies wird als Internet Media Type oder auch MIME Type bezeichnet.
    - Zeile 3 gibt an, welche Arten von Dokumenten (MIME Typen) vom Client akzeptiert werden.
    - Zeile 4 gibt an, um welchen User Agent es sich handelt.
    - Zeile 5 beinhaltet die Adresse des Hosts.
    - Zeile 6 und 7 verhindern das Verwenden eines Caches.
    - Die SOAPAction Angabe im HTTP Header kann dazu verwendet werden, um die Intention des Requests zu verdeutlichen.
- Im rot hinterlegten Teil – die Zeilen 13 bis 31 - sieht man die SOAP-Message.
- Zeile 13 beinhaltet die XML-Deklaration.
- Der SOAP-Envelope umschließt die Zeilen 14 bis 31.
- In den Zeilen 15 bis 27 steht der SOAP-Header.
- Der SOAP-Body ist in den Zeilen 28 bis 30, mit dem Namen der aufzurufenden Methode in Zeile 29.

**REQUEST:**

```
1  POST /JobManager/scs/ApplicationService HTTP/1.1
2  Content-Type: text/xml; charset=utf-8
3  Accept: application/soap+xml, application/dime, multipart/related, text/*
4  User-Agent: Axis/1.2.1
5  Host: 127.0.0.1:8900
6  Cache-Control: no-cache
7  Pragma: no-cache
8  SOAPAction: ""
9  Content-Length: 740
10 Connection: close
11 Authorization: Basic Og==
12
13 <?xml version="1.0" encoding="UTF-8"?>
14 <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"
       xmlns:wsa="http://schemas.xmlsoap.org/ws/2004/08/addressing">
```

```
15    <soapenv:Header>
16      <wsa:MessageID soapenv:mustUnderstand="0">
17        uuid:909de240-3a4a-11dd-bf17-8f1b58c305c6
18      </wsa:MessageID>
19      <wsa:To soapenv:mustUnderstand="0">
20        http://localhost:8900/JobManager/scs/ApplicationService
21      </wsa:To>
22      <wsa:From soapenv:mustUnderstand="0">
23        <wsa:Address>
24          http://schemas.xmlsoap.org/ws/2004/08/addressing/role/anonymous
25        </wsa:Address>
26      </wsa:From>
27    </soapenv:Header>
28    <soapenv:Body>
29      <getCId xmlns="http://service.isc.univie.ac.at"/>
30    </soapenv:Body>
31  </soapenv:Envelope>
```

- In Zeile 1 findet man den Statuswert bzgl. der Verarbeitung der Nachricht.
    - 200 bedeutet eine erfolgreiche Verarbeitung der Nachricht.
    - Wäre die Response-Nachricht eine SOAP Fault-Nachricht, dann wäre der HTTP-Status 500 und die Nachricht würde ein Fault-Element beinhalten.
- Im Header der Response-Nachricht wird in Zeile 20 bis 22 auf die Message-ID der dazugehörigen Request-Nachricht verwiesen.
- Die Zeilen 25 bis 29 enthalten den Response-Teil der Antwort, die Zeilen 26 bis 28 enthalten das entsprechende Ergebnis des RPCs bzw. Zeile 27 die gewünschte Client-Id.

**RESPONSE:**

```
1   HTTP/1.1 200 OK
2   Content-Type: text/xml;charset=utf-8
3   Date: Sat, 14 Jun 2008 19:46:21 GMT
4   Server: Apache-Coyote/1.1
5   Connection: close
6   <?xml version="1.0" encoding="utf-8"?>
7   <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"
        xmlns:wsa="http://schemas.xmlsoap.org/ws/2004/08/addressing">
8     <soapenv:Header>
9       <wsa:MessageID soapenv:mustUnderstand="0">
10        uuid:90a36080-3a4a-11dd-9970-a80cd794a14d
11      </wsa:MessageID>
12      <wsa:To soapenv:mustUnderstand="0">
13        http://schemas.xmlsoap.org/ws/2004/08/addressing/role/anonymous
14      </wsa:To>
15      <wsa:From soapenv:mustUnderstand="0">
16        <wsa:Address>
17          http://localhost:8900/JobManager/scs/ApplicationService
18        </wsa:Address>
19      </wsa:From>
20      <wsa:RelatesTo RelationshipType="wsa:Response" soapenv:mustUnderstand="0">
21        uuid:909de240-3a4a-11dd-bf17-8f1b58c305c6
22      </wsa:RelatesTo>
23    </soapenv:Header>
24    <soapenv:Body>
25      <getCIdResponse xmlns="http://service.isc.univie.ac.at">
26        <getCIdReturn xmlns="">
27          1213472781956-584
28        </getCIdReturn>
29      </getCIdResponse>
30    </soapenv:Body>
31  </soapenv:Envelope>
```

Listing 11: SOAP-/HTTP-Request und Response einer VGE-Client/Service-Kommunikation

### 8.1.2 SOAP-Request mit Attachment

Hier ist das Beispiel für eine SOAP-Nachricht, die ein Attachment beinhaltet. Es wird die Methode *uploadNamedAttachment* aufgerufen und das mit OGSA-DAI generierte Perform Dokument als Attachment mitgeschickt , welches sämtliche Jobs des Users mit dem Login „*wj*“ abrufen soll.

- In Zeile 2 ist der Content-Type „*multipart/related*” angegeben, es handelt sich somit um einen Request mit Attachment (*multipart/related*: eine Nachricht, die mehrere Bodies beinhaltet, deren MIME Typ unterschiedlich sein kann). Der Seperator (Parameter *boundary*) befindet sich ebenfalls in Zeile 2. Dieser String dient zur Trennung der einzelnen Teile der Nachricht:
  - in Zeile 12 vor der SOAP Nachricht
  - in Zeile 48 zur Trennung zwischen SOAP Nachricht und dem OGSA-DAI Perform Dokument.
  - In Zeile 66 am Ende des OGSA-DAI Attachments bzw. am Ende der Nachricht
- Die Zeilen 13-15 und 49-51 beinhalten jeweils MIME Typ-Angaben bzgl. des nächsten Nachrichtenteiles.
- Es wird die Methode *uploadNamedAttachment* aufgerufen – Zeile 40. Mit dem *href*-Attribut wird auf das Attachment referenziert (Zeile 41), es beinhaltet die Content-Id des dazugehörigen MIME-Teiles.
- Das Attachment bzw. Perform-Dokument findet man in den Zeilen 52-65.

**REQUEST:**

```
1  POST /JobManager/scs/ApplicationService HTTP/1.1
2  Content-Type: multipart/related; type="text/xml";
       start="<4B24A2277F05F3B2FF2C9437E7427047>";
       boundary="----= Part 15 503405.1213472782097"
3  Accept: application/soap+xml, application/dime, multipart/related, text/*
   User-Agent: Axis/1.2.1
4  Host: 127.0.0.1:8900
5  Cache-Control: no-cache
6  Pragma: no-cache
7  SOAPAction: ""
8  Content-Length: 2322
9  Connection: close
10 Authorization: Basic Og==
11
12 ------=_Part_15_503405.1213472782097
13 Content-Type: text/xml; charset=UTF-8
14 Content-Transfer-Encoding: binary
15 Content-Id: <4B24A2277F05F3B2FF2C9437E7427047>
16 <?xml version="1.0" encoding="UTF-8"?>
17 <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"
       xmlns:wsa="http://schemas.xmlsoap.org/ws/2004/08/addressing">
18   <soapenv:Header>
19     <wsa:MessageID soapenv:mustUnderstand="0">
20       uuid:90b736a0-3a4a-11dd-bf17-8f1b58c305c6
21     </wsa:MessageID>
22     <wsa:To soapenv:mustUnderstand="0">
23       http://localhost:8900/JobManager/scs/
24     </wsa:To>
```

```
25      <wsa:From soapenv:mustUnderstand="0">
26        <wsa:Address>
27          http://schemas.xmlsoap.org/ws/2004/08/addressing/role/anonymous
28        </wsa:Address>
29        <wsa:ReferenceProperties>
30          <ns1:cid
                xmlns:ns1="http://localhost:8900/JobManager/scs/"
                xmlns:soapenc="http://schemas.xmlsoap.org/soap/encoding/"
                xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
                xsi:type="soapenc:string">1213472781956-584</ns1:cid>
31        </wsa:ReferenceProperties>
32      </wsa:From>
33      <wsa:FaultTo soapenv:mustUnderstand="0">
34        <wsa:Address>
35          http://www.apache.org
36        </wsa:Address>
37      </wsa:FaultTo>
38    </soapenv:Header>
39    <soapenv:Body>
40      <uploadNamedAttachment xmlns="http://service.isc.univie.ac.at">
41        <dataHandler
              href="cid:1E31EBF94771E957CC1503078676C194"
              xsi:type="xsd:DataHandler" xmlns=""/>
42        <fileName xsi:type="xsd:string" xmlns="">
43          inputFile.txt
44        </fileName>
45      </uploadNamedAttachment>
46    </soapenv:Body>
47  </soapenv:Envelope>
48  ------= Part_15_503405.1213472782097
49  Content-Type: application/octet-stream
50  Content-Transfer-Encoding: binary
51  Content-Id: <1E31EBF94771E957CC1503078676C194>
52  <?xml version="1.0" encoding="UTF-8"?>
53  <perform xmlns="http://ogsadai.org.uk/namespaces/2005/10/types">
54    <session/>
55    <sqlQueryStatement name="SQLQuery-ogsadai-11a784578de">
56      <expression>
57        SELECT jid, login WHERE login='wj' ORDER BY jid
58      </expression>
59      <resultStream name="ogsadai-11a784578df"/>
60    </sqlQueryStatement>
61    <sqlResultsToXML name="WebRowSet-ogsadai-11a784578e0">
62      <resultSet from="ogsadai-11a784578df"/>
63      <webRowSet name="ogsadai-11a784578e1"/>
64    </sqlResultsToXML>
65  </perform>
66  ------=_Part_15_503405.1213472782097--

RESPONSE:   NUR SOAP-BODY mit Response Element
 :  :  :  :  :  :  :  :  :  :  :
24    <soapenv:Body>
25      <uploadNamedAttachmentResponse xmlns="http://service.isc.univie.ac.at"/>
26    </soapenv:Body>
27  </soapenv:Envelope>
```

Listing 12: SOAP-Request mit Attachment

### 8.1.3 SOAP-Response mit Attachment

In diesem Beispiel werden die unter „8.1.2 SOAP-Request mit Attachment“ angeforderten Daten zurückgeliefert, indem die Methode *downloadAttachment* aufgerufen wird.

**REQUEST:**

```
 :  :  :  :  :  :  :  :  :  :  :
37    <soapenv:Body>
38      <downloadAttachment xmlns="http://service.isc.univie.ac.at">
39        <fileName xsi:type="xsd:string" xmlns="">
40          outputFile.txt
41        </fileName>
42      </downloadAttachment>
43    </soapenv:Body>
44  </soapenv:Envelope>
```

- In den Zeilen 27-29 befindet sich der Response-Teil der Methode *downloadAttachment*.
- Der Rückgabewert *downloadAttachmentReturn* hat als Parameter eine *Content-Id* (Zeile 28), die auf den MIME-Teil mit dem OGSA-DAI Response Attachment referenziert.
- In den Zeilen 36-129 befindet sich das OGSA-DAI Attachment, welches das Ergebnis der Datenbankabfrage beinhaltet.

**RESPONSE:**

```
 1  HTTP/1.1 200 OK
 2  Content-Type: multipart/related; type="text/xml";
        start="<36E431C1B9B0EAB5B67249B63145312F>";
        boundary="----=_Part_12_9634080.1213472784553"
 3  Date: Sat, 14 Jun 2008 19:46:26 GMT
 4  Server: Apache-Coyote/1.1
 5  Connection: close
 6  ------=_Part_12_9634080.1213472784553
 7  Content-Type: text/xml; charset=UTF-8
 8  Content-Transfer-Encoding: binary
 9  Content-Id: <36E431C1B9B0EAB5B67249B63145312F>
10  <?xml version="1.0" encoding="utf-8"?>
11  <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"
        xmlns:wsa="http://schemas.xmlsoap.org/ws/2004/08/addressing">
12    <soapenv:Header>
13      <wsa:MessageID soapenv:mustUnderstand="0">
14        uuid:922ee280-3a4a-11dd-9970-a80cd794a14d
15      </wsa:MessageID>
16      <wsa:To soapenv:mustUnderstand="0">
17        http://schemas.xmlsoap.org/ws/2004/08/addressing/role/anonymous
18       </wsa:To>
19      <wsa:From soapenv:mustUnderstand="0">
20        <wsa:Address>http://localhost:8900/JobManager/scs/</wsa:Address>
21      </wsa:From>
22      <wsa:RelatesTo
            RelationshipType="wsa:Response"
            soapenv:mustUnderstand="0">
23        uuid:922b8720-3a4a-11dd-bf17-8f1b58c305c6
24      </wsa:RelatesTo>
25    </soapenv:Header>
26    <soapenv:Body>
27      <downloadAttachmentResponse xmlns="http://service.isc.univie.ac.at">
28        <downloadAttachmentReturn
              href="cid:DE67AAB3CCC9DEBAA47369CF1494038F"/>
29      </downloadAttachmentResponse>
30    </soapenv:Body>
31  </soapenv:Envelope>
32  ------=_Part_12_9634080.1213472784553
33  Content-Type: text/plain
34  Content-Transfer-Encoding: binary
35  Content-Id: <DE67AAB3CCC9DEBAA47369CF1494038F>
```

```
 36  <?xml version="1.0" encoding="UTF-8"?>
 37  <ns1:response xmlns:ns1="http://ogsadai.org.uk/namespaces/2005/10/types">
 38    <ns1:session id="session-ogsadai-11a784581be"/>
 39    <ns1:request status="COMPLETED"/>
 40    <ns1:result name="SQLQuery-ogsadai-11a784578de" status="COMPLETED"/>
 41    <ns1:result name="WebRowSet-ogsadai-11a784578e0" status="COMPLETED"/>
 42    <ns1:result name="ogsadai-11a784578e1" status="COMPLETED">
 43      <![CDATA[
 44        <webRowSet
               xmlns="http://java.sun.com/xml/ns/jdbc"
               xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
               xsi:schemaLocation="http://java.sun.com/xml/ns/jdbc
                   http://java.sun.com/xml/ns/jdbc/webrowset.xsd">
 45          <properties>
 46            <command></command>
 47            <concurrency>1007</concurrency>
 48            <datasource></datasource>
 49            <escape-processing>true</escape-processing>
 50            <fetch-direction>1000</fetch-direction>
 51            <fetch-size>0</fetch-size>
 52            <isolation-level>0</isolation-level>
 53            <key-columns></key-columns>
 54            <map></map>
 55            <max-field-size>1048576</max-field-size>
 56            <max-rows>0</max-rows>
 57            <query-timeout>0</query-timeout>
 58            <read-only>true</read-only>
 59            <rowset-type>ResultSet.TYPE_FORWARD_ONLY</rowset-type>
 60            <show-deleted>false</show-deleted>
 61            <table-name></table-name>
 62            <url></url>
 63            <sync-provider>
 64              <sync-provider-name/>
 65              <sync-provider-vendor/>
 66              <sync-provider-version/>
 67              <sync-provider-grade/>
 68              <data-source-lock/>
 69            </sync-provider>
 70          </properties>
 71          <metadata>
 72            <column-count>2</column-count>
 73            <column-definition>
 74              <column-index>1</column-index>
 75              <auto-increment>false</auto-increment>
 76              <case-sensitive>false</case-sensitive>
 77              <currency>false</currency>
 78              <nullable>0</nullable>
 79              <signed>false</signed>
 80              <searchable>true</searchable>
 81              <column-display-size>30</column-display-size>
 82              <column-label>jid</column-label>
 83              <column-name>jid</column-name>
 84              <schema-name></schema-name>
 85              <column-precision>30</column-precision>
 86              <column-scale>0</column-scale>
 87              <table-name>jobs</table-name>
 88              <catalog-name>vgds_jobmanager</catalog-name>
 89              <column-type>12</column-type>
 90              <column-type-name>VARCHAR</column-type-name>
 91            </column-definition>
 92            <column-definition>
 93              <column-index>2</column-index>
 94              <auto-increment>false</auto-increment>
 95              <case-sensitive>false</case-sensitive>
 96              <currency>false</currency>
 97              <nullable>0</nullable>
 98              <signed>false</signed>
 99              <searchable>true</searchable>
100              <column-display-size>15</column-display-size>
101              <column-label>login</column-label>
102              <column-name>login</column-name>
103              <schema-name></schema-name>
104              <column-precision>15</column-precision>
105              <column-scale>0</column-scale>
106              <table-name>users</table-name>
107              <catalog-name>vgds jobmanager</catalog-name>
108              <column-type>12</column-type>
109              <column-type-name>VARCHAR</column-type-name>
```

```
110            </column-definition>
111          </metadata>
112          <data>
113             <currentRow>
114               <columnValue>wj1212762509974-3</columnValue>
115               <columnValue>wj</columnValue>
116             </currentRow>
117             <currentRow>
118               <columnValue>wj1212762509974-38</columnValue>
119               <columnValue>wj</columnValue>
120             </currentRow
121             <currentRow>
122               <columnValue>wj1213198673216-41</columnValue>
123               <columnValue>wj</columnValue>
124            </currentRow>
125          </data>
126        </webRowSet>
127      ]]>
128    </ns1:result>
129  </ns1:response>
130  ------=_Part_12_9634080.1213472784553--
```

Listing 13: SOAP-Response mit Attachment

### 8.1.4 Ein Beispiel für MIME und DIME-Verfahren

Listing 14: SOAP-Nachricht mit dem MIME-Verfahren

```
Content-Type: multipart/related; type=„text/xml";
start="<9D645C8EBB837CE54ABD027A3659535D>";
boundary="--=_Part_0_1977511.1123163571138"
---=_Part_0_1977511.1123163571138
Content-Type: text/xml; charset=UTF-8
Content-Transfer-Encoding: binary
Content-Id: <9D645C8EBB837CE54ABD027A3659535D>
<?xml version="1.0" encoding="UTF-8"?>
<soapenv:Envelope xmlns:soapenv="...."....>
  ........
  <source href="cid:3936AE19FBED55AE4620B81C73BDD76E" />
  ........
</soapenv:Envelope>
---=_Part_0_1977511.1123163571138
Content-Type: text/plain
Content-Transfer-Encoding: binary
Content-Id: <3936AE19FBED55AE4620B81C73BDD76E>
... binärer Datenstrom
---=_Part_0_1977511.1123163571138-
```

Listing 15: SOAP-Nachricht mit dem DIME-Verfahren

```
1 0 0 0000000000000
010 0000000101001
00000000000000000000000010110110
http://schemas.xmlsoap.org/soap/envelope/
<envelope>
  <body>
    <importDokument>
      <metadata xsi:type="xsd:string">20010511-BEST-
          10010099.pdf</metadata>
      <image href="uuid:9A55E953-A828" />
    </importDokument>
  </body>
</envelope>
0 1 0 0000000000000
000 0000000000000
00000000000000000000011000111110000
uuid:9A55E953-A828
image/jpeg
... binärer Datenstrom
```

Quelle **[Prä06]**

### 8.1.5 SOAP-Nachricht mittels MTOM

Bei diesem Beispiel sieht man eine Nachricht mit Attachment in unterschiedlichen Formaten dargestellt: links mit inline-Kodierung (und Base64 Transformation) und rechts mittels XOP.

Werden Attachments in einer SOAP Nachricht mit der MTOM-Kodierung übertragen, so sind folgende vier Merkmale vorhanden:

1. Der MIME-Type der HTTP Nachricht ist „*multipart/related*“ – Zeile 4 in der rechten Nachricht.
2. Im Content-Type des HTTP Headers ist das *type* Attribut „*application/xop+xml*“ – Zeile 4.
3. Der Content-Type des ersten MIME Teiles (MIME Teil mit dem SOAP Envelope) ist „*application/xop+xml*“ – Zeile 11.
4. Der *start-info* Parameter im HTTP Header ist „*application/soap+xml*“.

Die Zeilen 24 und 30 beinhalten das *Include* Element für die Attachments.

```
1  POST /MTOMCatalog/Catalog HTTP/1.1
2  Content-Length: 48382
3  SOAPAction: ""
4  Content-Type: text/xml; charset=utf-8
5  Accept: text/xml, application/xop+xml,
       text/html, image/gif,
       image/jpeg, *; q=.2, */*; q=.2
6  User-Agent: Java/1.5.0 09
7  Host: localhost:8989
8  Connection: keep-alive
9
10 <?xml version="1.0" ?>
11 <soapenv:Envelope
       xmlns:soapenv="http://schemas.
           xmlsoap.org/soap/envelope/"
       xmlns:xsd="http://www.w3.org/
           2001/XMLSchema"
       xmlns:ns1="http://catalog.mtom.
           company.com/book">
12   <soapenv:Body>
13     <ns1:book>
14       <title>SOA Security in Action</title>
15       <author>Ramarao</author>
16       <pageNumber>350</pageNumber>
17       <isbn>1 932394 68 0</isbn>
18       <price>$44.99</price>
19       <desc>Implement SOA Security</desc>
20       <coverImage>/9j/4AAQSkZJRgABAgAAAQA...
             (base64 encoded cover image data
             with the rest
             omitted)</coverImage>
21       <sample>
22         <name>Introducing SOA </name>
23           <content>JVBERi0xLjQNJeLjz9MNC...
                 (base64 encoded sample content
                 with the rest
                 omitted)</content>
24       </sample>
25     </ns1:book>
26   </soapenv:Body>
27 </soapenv:Envelope>
```

```
1  POST /MTOMCatalog/Catalog HTTP/1.1
2  Content-Length: 37593
3  SOAPAction: ""
4  Content-Type: multipart/related;
       type="application/xop+xml";
       boundary=
       "----= Part 0 14615608.1167536350647";
       start-info="text/xml"
5  Accept: text/xml, application/xop+xml,
       text/html, image/gif,
       image/jpeg, *; q=.2, */*; q=.2
6  User-Agent: Java/1.5.0 09
7  Host: localhost:8989
8  Connection: keep-alive
9
10 ------=_Part_0_14615608.1167536350647
11 Content-Type: application/xop+xml;
       type="text/xml"; charset=utf-8
12
13 <?xml version="1.0" ?>
14 <soapenv:Envelope
       xmlns:soapenv="http://schemas.
           xmlsoap.org/soap/envelope/"
       xmlns:xsd="http://www.w3.org/
           2001/XMLSchema"
       xmlns:ns1="http://catalog.mtom.
           company.com/book">
15   <soapenv:Body>
16     <ns1:book>
17       <title>SOA Security in Action</title>
18       <author>Ramarao Kanneganti</author>
19       <pageNumber>350</pageNumber>
20       <isbn>1 932394 68 0</isbn>
21       <price>$44.99</price>
22       <desc>Implement SOA Security</desc>
23       <coverImage>
24         <xop:Include
               xmlns:xop="http://www.w3.org/
                   2004/08/xop/include"
               href="cid:1d0353b6-0ec7-4264-
                   b4f4-16c1e6a0812c">
25         </xop:Include>
26       </coverImage>
27       <sample>
28         <name>Introducing SOA</name>
29         <content>
30           <xop:Include
                 xmlns:xop="http://www.w3.org/
                     2004/08/xop/include"
                 href="cid:cf8f59b3-57a6-4594-
                     b04a-6925ad8db93a">
31           </xop:Include>
32         </content>
33       </sample>
34     </ns1:book>
35   </soapenv:Body>
36 </soapenv:Envelope>
```

| | | | |
|---|---|---|---|
| | | 37 | |
| | | 38 | ------=_Part_0_14615608.1167536350647 |
| | | 39 | Content-Type: application/octet-stream |
| | | 40 | Content-ID: <1d0353b6-0ec7-4264-<br>b4f4-16c1e6a0812c > |
| | | 41 | Content-transfer-encoding: binary |
| | | 42 | |
| | | 43 | ......(binary stream omitted) |
| | | 44 | |
| | | 45 | ------=_Part_0_14615608.1167536350647 |
| | | 46 | Content-Type: application/octet-stream |
| | | 47 | Content-ID: <cf8f59b3-57a6-4594-<br>b04a-6925ad8db93a> |
| | | 48 | Content-transfer-encoding: binary |
| | | 49 | |
| | | 50 | ......(binary stream omitted) |
| | | 51 | |
| | | 52 | ------=_Part_0_14615608.1167536350647-- |

SOAP Nachricht mit Binärdaten über HTTP XOP-serialisierte SOAP Nachricht über HTTP

**Listing 16: SOAP-Nachricht mit und ohne XOP-Serialisierung. Quelle [Yan07]**

Eine Anmerkung bzgl. dem *start-info* Parameter: Zuvor wurde erwähnt, dass der *start-info* Parameter im HTTP Header „*application/soap+xml*“ sein muss – dies ist gültig für die SOAP Spezifikation 1.2. Im rechten Beispiel sieht man eine SOAP-Nachricht laut Spezifikation 1.1, deshalb hat der *start-info* Parameter auch den Wert „*text/xml*“.

## 8.2 WSDL (Teil einer VGE Service WSDL-Beschreibung)

Hier ist ein Teil der WSDL Beschreibung eines VGE Applikationsservices. Sie beinhaltet die Definitionen für die unter „8.1 SOAP-Listings“ angeführten VGE Nachrichtenbeispiele:

(1) RPC Aufruf von *getCId* in „8.1.1 SOAP-/HTTP-Request und Response einer VGE-Client/Service-Kommunikation”
(2) RPC Aufruf von *uploadNamedAttachment* in „8.1.2 SOAP-Request mit Attachment”
(3) RPC Aufruf von *downloadAttachment* in „8.1.3 SOAP-Response mit Attachment”

- In den Zeilen 11 bis 46 findet man die verschiedenen Nachrichten mittels dem *message* Element definiert; *uploadNamedAttachmentRequest* und *uploadNamedAttachmentResponse* für den RPC Aufruf *uploadNamedAttachment*, usw.

  Für jede Nachricht sind die Parameter und deren Typ mit dem *part*-Element aufgelistet.

- Die Zeilen 56 bis 102 beinhalten das *portType*-Element mit den einzelnen Operationen des Services. Für jedes *operation*-Element wird die Request und Response-Nachricht (dies ist der Normalfall; zuvor mittels *message* - im gelben Teil - definiert) angegeben.
- Das *binding*-Element (Zeilen 104 – 213) beinhaltet für jede unter den zuvor definierten Operationen im *portType*-Element das konkrete Nachrichtenformat bzw. spezifische Details.

- Die Netzwerkaddresse des Services wird durch das *service*-Element festgelegt – Zeilen 215-219.

```
 1  <?xml version="1.0" encoding="UTF-8"?>
 2  <!-- This file is automatically generated by the provisioning environment -->
 3  <wsdl:definitions
 4      xmlns:xsd="http://www.w3.org/2001/XMLSchema"
 5      xmlns:soapbind="http://schemas.xmlsoap.org/wsdl/soap/"
 6      xmlns:wsdl="http://schemas.xmlsoap.org/wsdl/"
 7      xmlns:mime="http://schemas.xmlsoap.org/wsdl/mime/"
 8      targetNamespace="http://service.isc.univie.ac.at"
 9      xmlns:impl="http://service.isc.univie.ac.at">
10
11    <wsdl:message name="uploadNamedAttachmentRequest">
12      <wsdl:part name="dataHandler" type="xsd:base64Binary" />
13      <wsdl:part name="fileName" type="xsd:string" />
14    </wsdl:message>
15    <wsdl:message name="uploadNamedAttachmentResponse" />
16
17    <wsdl:message name="downloadAttachmentRequest">
18      <wsdl:part name="fileName" type="xsd:string" />
19    </wsdl:message>
20    <wsdl:message name="downloadAttachmentResponse">
21      <wsdl:part name="downloadAttachmentReturn" type="xsd:base64Binary" />
22    </wsdl:message>
 :
43    <wsdl:message name="getCIdRequest" />
44    <wsdl:message name="getCIdResponse">
45      <wsdl:part name="getCIdReturn" type="xsd:string" />
46    </wsdl:message>
 :
56    <wsdl:portType name="ApplicationService">
57
58      <wsdl:operation name="uploadNamedAttachment"
            parameterOrder="dataHandler fileName">
59        <wsdl:input message="impl:uploadNamedAttachmentRequest"
              name="uploadNamedAttachmentRequest" />
60        <wsdl:output message="impl:uploadNamedAttachmentResponse"
              name="uploadNamedAttachmentResponse" />
61      </wsdl:operation>
62
63      <wsdl:operation name="downloadAttachment" parameterOrder="fileName">
64        <wsdl:input message="impl:downloadAttachmentRequest"
              name="downloadAttachmentRequest" />
65        <wsdl:output message="impl:downloadAttachmentResponse"
              name="downloadAttachmentResponse" />
66      </wsdl:operation>
 :
93      <wsdl:operation name="getCId">
94        <wsdl:input message="impl:getCIdRequest" name="getCIdRequest" />
95        <wsdl:output message="impl:getCIdResponse" name="getCIdResponse" />
96      </wsdl:operation>
 :
102   </wsdl:portType>
103
104   <wsdl:binding name="ApplicationServiceSoapBinding"
          type="impl:ApplicationService">
105     <soapbind:binding style="rpc"
            transport="http://schemas.xmlsoap.org/soap/http" />
106
107     <wsdl:operation name="uploadNamedAttachment">
108       <soapbind:operation soapAction="" />
109       <wsdl:input name="uploadNamedAttachmentRequest">
110         <mime:multipartRelated>
111           <mime:part>
112             <soapbind:body use="literal"
                    namespace="http://service.isc.univie.ac.at"/>
113           </mime:part>
114           <mime:part>
115             <mime:content part="dataHandler" type="application/octet-stream" />
116           </mime:part>
117           <mime:part>
118             <mime:content part="fileName" type="text/plain" />
119           </mime:part>
120         </mime:multipartRelated>
```

```
121         </wsdl:input>
122         <wsdl:output name="uploadNamedAttachmentResponse">
123           <soapbind:body namespace="http://service.isc.univie.ac.at"
                  use="literal" />
124         </wsdl:output>
125       </wsdl:operation>
126
127       <wsdl:operation name="downloadAttachment">
128         <soapbind:operation soapAction="" />
129         <wsdl:input name="downloadAttachmentRequest">
130           <soapbind:body namespace="http://service.isc.univie.ac.at"
                  use="literal" />
131         </wsdl:input>
132         <wsdl:output name="downloadAttachmentResponse">
133           <mime:multipartRelated>
134             <mime:part>
135               <soapbind:body use="literal"
                      namespace="http://service.isc.univie.ac.at" />
136             </mime:part>
137             <mime:part>
138               <mime:content part="downloadAttachmentReturn"
                      type="application/octet-stream" />
139             </mime:part>
140           </mime:multipartRelated>
141         </wsdl:output>
142       </wsdl:operation>
  :
194       <wsdl:operation name="getCId">
195         <soapbind:operation soapAction="" />
196         <wsdl:input name="getCIdRequest">
197           <soapbind:body namespace="http://service.isc.univie.ac.at"
                  use="literal" />
198         </wsdl:input>
199         <wsdl:output name="getCIdResponse">
200           <soapbind:body namespace="http://service.isc.univie.ac.at"
                  use="literal" />
201         </wsdl:output>
202       </wsdl:operation>
  :
213     </wsdl:binding>
214
215     <wsdl:service name="ApplicationServiceService">
216       <wsdl:port binding="impl:ApplicationServiceSoapBinding"
              name="ApplicationService">
217         <soapbind:address location="http://localhost:9002/JobManager/
                scs/ApplicationService" />
218       </wsdl:port>
219     </wsdl:service>
220
221 </wsdl:definitions>
```

Listing 17: Teil einer VGE Service WSDL-Beschreibung

## 8.3 UDDI

### 8.3.1 SOAP-Nachricht zum Speichern einer UDDI Business Entity

```
1  POST /save business HTTP/1.1
2  Host: www.xyz.com
3  Content-Type: text/xml; charset="utf-8"
4  Content-Length: nnnn
5  SOAPAction: "save_business"

6  <?xml version="1.0" encoding="UTF-8" ?>
7    <Envelope xmlns="http://schemas/xmlsoap.org/soap/envelope/">
8      <Body>
9        <save_business generic="2.0" xmlns="urn:uddi-org:api_v2">
10         <businessKey="">
11         </businessKey>
12         <name>
13           XMethods
14         </name>
15         <description>
16           Web services resource site
17         </description>
18         <identifierBag> ... </identifierBag>
19         ...
20       </save_business>
21     </Body>
22   </Envelope>
```

Listing 18: SOAP-Nachricht zum Speichern einer UDDI Business Entity

Das *BusinessKey*-Element in Zeile 10 ist leer, da vom Operator ein eindeutiger, automatisch generierter UDDI Schlüssel diesem Verzeichniseintrag zugeordnet wird.

In den nachfolgenden Beispielen werden die im vorigen Beispiel registrierten Daten ausgelesen. Erhält man beim Registrieren des Business Elementes nicht den eindeutigen UDDI Schlüssel zurück, so muss für dessen Ermittlung eine Abfrage mit dem vollständigen Namen oder mit Teilen des Namens und Wildcards erfolgen.

```
<?xml version="1.0" encoding="UTF-8" ?>
  <Envelope xmlns="http://schemas/xmlsoap.org/soap/envelope/">
    <Body>
      <find_business generic="2.0" xmlns="urn:uddi-org:api_v2">
        <name>XMethods</name>
      </find service>
    </Body>
  </Envelope>
```

Ergebnis:

```
<businessList generic="1.0"
    operator="Microsoft Corporation"
    truncated="false"
    xmlns="urn:uddi-org:api">
  <businessInfos>
    <businessInfo businessKey="0076B468-EB27-42E5-AC09-9955CFF462A3">
      <name>XMethods</name>
      <description xml:lang="en">
        Web services resource site
      </description>
      <serviceInfos>
        <serviceInfo
            businessKey="0076B468-EB27-42E5-AC09-9955CFF462A3"
            serviceKey="1FFE1F71-2AF3-45FB-B788-09AF7FF151A4">
          <name>Web services for smart searching</name>
        </serviceInfo>
        <serviceInfo
            businessKey="0076B468-EB27-42E5-AC09-9955CFF462A3"
```

```
            serviceKey="8BF2F51F-8ED4-43FE-B665-38D8205D1333">
          <name>Electronic Business Integration Services</name>
        </serviceInfo>
      </serviceInfos>
    </businessInfo>
  </businessInfos>
 </businessList>
```

Listing 19: Lesen aller UDDI Business Einträge mit den Namen „XMethods“ & Ergebnis

Das Lesen eines bestimmten UDDI Service Eintrages kann mit der *get_serviceDetail*-Methode erfolgen:

```
<get_serviceDetail generic="2.0" xmlns="urn:uddi-org:api_v2">
  <serviceKey>8BF2F51F-8ED4-43FE-B665-38D8205D1333</serviceKey>
</get_serviceDetail>
```

Ergebnis:

```
<serviceDetail generic="2.0"
    operator="Microsoft Corporation"
    truncated="false"
    xmlns="urn:uddi-org:api">
  <businessService
      serviceKey="d5921160-8BF2F51F-8ED4-43FE-B665-38D8205D1333"
      businessKey="0076B468-EB27-42E5-AC09-9955CFF462A3">
    <name>XMethods Electronic Business Integration Services</name>
    <description xml:lang="en">Department and employee integration</description>
    <bindingTemplates>
      <bindingTemplate
          bindingKey="8B7CA31B-8ED4-43FE-B665-38D8205D1333"
          serviceKey="8BF2F51F-8ED4-43FE-B665-38D8205D1333">
        <description xml:lang="en">
          SOAP binding for Electronic Business Integration Service
        </description>
        <accessPoint URLType="http">
           http://services.xmethods.net:80/soap
        </accessPoint>
        <tModelInstanceDetails>
          <tModelInstanceInfo
              tModelKey="uuid:4EA91C08-8EA5-43FE-B665-38D8205D1333" />
        </tModelInstanceDetails>
      </bindingTemplate>
    </bindingTemplates>
  </businessService>
</serviceDetail>
```

Listing 20: Lesen eines bestimmten UDDI Service Eintrages & Ergebnis

## 8.4 Implementierungsunterschiede auf Grund von J2ME

In diesem Kapitel sind einige Tabellen aus dem Buch „Java™ 2 Micro Edition Application Development“ **[Kro02]** abgebildet, die den Unterschied von J2SE und J2ME verdeutlichen.

Es werden Klassen von J2SE (Version 1.3) mit den äquivalenten Klassen von J2ME CLDC 1.0 und CLDC 1.1 verglichen. Aus dem Buch sind die Tabellen für die Pakete *java.io*, *java.lang* und *java.util* entnommen. Sie zeigen welche Klassen von J2SE auch in J2ME implementiert sind und welche nicht. Für manche, nicht implementierte Klassen sind mögliche Workarounds angeführt.

### *java.io-Package*

Table B.57. Interfaces of the java.io Package

| J2SE Interface | Availability in CLDC |
|---|---|
| DataInput | Partially contained; see Table B.60 for details. |
| DataOutput | Partially contained; see Table B.61 for details. |
| **Not available in CLDC:** Externalizable, FileFilter, FilenameFilter, ObjectInput, ObjectInputValidation, ObjectOutput, ObjectStreamConstants, Serializable | |

Table B.58. Classes of the java.io Package

| J2SE Class | Availability in CLDC |
|---|---|
| BufferedReader, BufferedWriter | Not available in CLDC.<br><br>Workaround for readLine() using PC and UNIX encoding:<br>`static String readLine (Reader reader) throws IOException {`<br>`  StringBuffer buf = new StringBuffer ();`<br>`  while (true) {`<br>`  int c = reader.read ();`<br>`  if (c == -1) {`<br>`  if (buf.length () == 0) return null;`<br>`  break;`<br>`  }`<br>`  if (c == '\n') break;`<br>`  if (c != '\r') buf.append ((char) c);`<br>`  }`<br>`  return buf.toString ();`<br>`}` |
| ByteArrayInputStream | All J2SE methods are available in CLDC. |
| ByteArrayOutputStream | Partially contained; see Table B.62 for details. |
| DataInputStream, DataOutputStream | Partially contained; see Table B.63 and B.64 for details. |
| File, FileDescriptor, FileInputStream, FilePermission, FileReader, FileWriter | Files not available in CLDC. Use the classes of the javax.microedition.rms as an alternative. For accessing files on memory cards, some devices may provide a file:// protocol implementation in the generic connection framework (see Chapter 6, "Networking: The Generic Connection Framework"). |
| InputStream | All J2SE methods are available in CLDC. |
| InputStreamReader | Partially contained; see Table B.65 for details. |
| OutputStream | All J2SE methods are available in CLDC. |
| OutputStreamWriter | Partially contained; see Table B.66 details. |
| PrintStream | Partially contained; see Table B.67 for details. |
| PushbackReader | Not available in CLDC. kXML contains a LookAheadReader that is comparable to some extent. |

| | |
|---|---|
| RandomAccessFile | Files are not available in CLDC. Use the classes of the javax.microedition.rms as an alternative. For accessing files on memory cards, some devices may provide a file:// protocol implementation in the generic connection framework (see Chapter 6). |
| Reader | Fully available in CLDC. |
| StringBufferInputStream | Not available in CLDC. See StringReader for a workaround. |
| StringReader | Not available in CLDC.<br><br>Use<br>`new InputStreamReader (new ByteArrayInputStream (s.getBytes ()));`<br>instead of<br>`new StringReader (s);` |
| StringWriter | Not available in CLDC.<br><br>Use<br>`ByteArrayOutputStream bos = new ByteArrayOutputStream (); OutputStreamWriter sw = new OutputStreamWriter (bos); // ... write to sw String s = new String (bos.getByteArray ());`<br>instead of<br>`StringWriter sw = new StringWriter (); // ... write to sw String s = sw.toString ();` |
| Writer | Fully available in CLDC. |
| **Not available in CLDC:** BufferedInputStream, BufferedOutputStream, CharArrayReader, CharArrayWriter, FilterInputStream, FilterOutputStream, FilterReader, FilterWriter, LineNumberInputStream, LineNumberReader, ObjectInputStream, ObjectInputStream.GetField, ObjectOutputStream, ObjectOutputStream.PutField, ObjectStreamClass, ObjectStreamField, PipedInputStream, PipedOutputStream, PipedReader, PipedWriter, PrintWriter, PushbackInputStream, SequenceInputStream, SerializablePermission, StreamTokenizer | |

### *java.lang-Package*

Table B.68. Interfaces of the java.lang Package

| J2SE Interface | Availability in CLDC |
|---|---|
| Runnable | Fully available in CLDC. |
| **Not available in CLDC:** Clonable, Comparable | |

Table B.69. Classes of the java.lang Package

| J2SE Class | Availability in CLDC |
|---|---|
| Boolean | Partially contained; see Table B.72 for details. |
| Byte | Partially contained; see Table B.73 for details. |
| Character | Partially contained; see Table B.74 for details. |
| Class | Partially contained; see Table B.75 for details. |
| Double | Partially contained in CLDC-NG; see Table B.76 for details. |
| Float | Partially contained in CLDC-NG; see Table B.77 for details. |
| Integer | Partially contained; see Table B.78 for details. |
| Long | Partially contained; see Table B.79 for details. |
| Math | Partially contained; see Table B.80 for details. |
| Object | Partially contained. The CLDC version lacks the clone() and finalize() methods that are provided in J2SE. |
| Runtime | Partially contained; see the section "Runtime" for details. |
| Short | Partially contained; see Table B.81 for details. |
| String | Partially contained; see Table B.82 for details. |
| StringBuffer | Partially contained; see Table B.83 for details. |

Table B.69. Classes of the java.lang Package

| J2SE Class | Availability in CLDC |
|---|---|
| System | Partially contained; see Table B.84 for details. |
| Thread | Partially contained; see Table B.85 for details. |
| Throwable | Partially contained; see Table B.86 for details. |
| **Not available in CLDC:** Character.Subset, Character.UnicodeBlock, ClassLoader, Compiler, InheritableThreadLocal, Number, Package, Process, RuntimePermission, SecurityManager, StrictMath, ThreadGroup, ThreadLocal, Void | |

## *java.util-Package*

Table B.91. Interfaces of the java.util Package

| J2SE Interface | Availability in CLDC/PDAP |
|---|---|
| Collection | Not available in CLDC. Workaround: Vector |
| Enumeration | Fully available in CLDC. |
| EventListener | Fully available in PDAP. |
| Iterator | Not available in CLDC. Workaround: Enumeration |
| List | Not available in CLDC. Workaround: Vector |
| Map | Not available in CLDC. Workaround: Hashtable |
| **Not available in CLDC:** Comparator, ListIterator, Map.Entry, Observer, Set, SortedMap, SortedSet | |

Table B.92. Classes of the java.util Package

| J2SE Class | Availability in CLDC/PDAP |
|---|---|
| ArrayList | Not available in CLDC. Workaround: Vector |
| Calendar | Partially contained; see Table B.94 for details. |
| Date | Partially contained; see Table B.95 for details. |
| Dictionary | Not available in CLDC. Workaround: Hashtable |
| EventObject | Fully available in PDAP. |
| HashMap | Not available in CLDC. Workaround: Hashtable |
| Hashtable | Partially contained; see Table B.96 for details. |
| LinkedList | Not available in CLDC. Workaround: Vector |
| Locale | Partially contained in PDAP; see Table B.97 for details. |
| Properties | Not available in CLDC. Workaround: Hashtable |
| Random | Partially contained; see Table B.98 for details. |
| Stack | Fully available in CLDC. |
| Timer | Partially contained; see Table B.99 for details. This class is an MIDP-specific addition to CLDC. |
| TimerTask | Fully available in CLDC. This class is an MIDP-specific addition to CLDC. |
| TimeZone | Partially contained; see Table B.100 for details. |
| TreeMap | Not available in CLDC. Workaround: Hashtable |
| Vector | Partially contained; see Table B.101 for details. |
| **Not available in CLDC:** AbstractCollection, AbstractList, AbstractMap, AbstractSequentialList, AbstractSet, Arrays, BitSet, Collections, GregorianCalendar, HashSet, ListResourceBundle, Observable, PropertyPermission, PropertyResourceBundle, ResourceBundle, SimpleTimeZone, StringTokenizer, TreeSet, WeakHashMap | |

# 9 Zusätzliche Listings

## 9.1 SQL-Anweisungen zum Erzeugen der Datenbank

```
CREATE DATABASE vge_jobmanager;
GRANT SELECT,INSERT,UPDATE,DELETE,CREATE,DROP ON vge_jobmanager.* TO
'poweradmin'@'localhost' IDENTIFIED BY 'poweradmin!';
GRANT SELECT,INSERT,UPDATE,DELETE,CREATE,DROP ON vge_jobmanager.* TO 'poweradmin'@'%'
IDENTIFIED BY 'poweradmin!';
FLUSH PRIVILEGES;

CREATE TABLE `vge_jobmanager`.`users` (
  `user_id` INTEGER NOT NULL AUTO_INCREMENT,
  `login` VARCHAR(15) NOT NULL,
  `first_name` VARCHAR(20) NOT NULL,
  `last_name` VARCHAR(30) NOT NULL,
  `password` CHAR(32) NOT NULL,
  `change_pw` BIT(1) NOT NULL DEFAULT 1,
  `locked` BIT(1) NOT NULL DEFAULT 0,
  `is_admin` BIT(1) NOT NULL DEFAULT 0,
  `imei` BIGINT,
  `email` VARCHAR(60),
  PRIMARY KEY (`user_id`),
  UNIQUE KEY `login` (`login`)
)
ENGINE = InnoDB;

INSERT INTO `vge_jobmanager`.`users` (login, first_name, last_name, password,
change_pw, locked, is_admin) VALUES('poweradmin', 'power', 'admin', md5('poweradmin'),
0, 0, 1);

CREATE TABLE `vge_jobmanager`.`jobs` (
  `jid` VARCHAR(30) NOT NULL,
  `user_id` INTEGER NOT NULL,
  `client_uri` VARCHAR(100) NOT NULL,
  `service_uri` VARCHAR(100) NOT NULL,
  `state` TINYINT NOT NULL DEFAULT '-2',
  `output_file_pattern` VARCHAR(30),
  `date_uploaded` DATETIME DEFAULT NULL,
  `date_started` DATETIME DEFAULT NULL,
  `date_downloaded` DATETIME DEFAULT NULL,
  `date_killed` DATETIME DEFAULT NULL,
  PRIMARY KEY (`jid`),
  INDEX `FK_user_id`(`user_id`),
  CONSTRAINT `FK_user_id_jobs` FOREIGN KEY `FK_user_id` (`user_id`)
    REFERENCES `users` (`user_id`)
    ON DELETE NO ACTION
    ON UPDATE NO ACTION
)
ENGINE = InnoDB;

CREATE TABLE `vge_jobmanager`.`administrate_user` (
  `admin_id` INTEGER NOT NULL,
  `user_id` INTEGER NOT NULL,
  CONSTRAINT `FK_admin_id_users` FOREIGN KEY `FK_admin_id` (`admin_id`)
    REFERENCES `users` (`user_id`)
    ON DELETE NO ACTION
    ON UPDATE NO ACTION,
  CONSTRAINT `FK_user_id_users` FOREIGN KEY `FK_user_id` (`user_id`)
    REFERENCES `users` (`user_id`)
    ON DELETE NO ACTION
    ON UPDATE NO ACTION
)
ENGINE = InnoDB;

COMMIT;
```

## 9.2 JavaDoc

### 9.2.1 JobManager für J2SE

at.ac.univie.isc.jobmanager.servlet.job
**Class JobManager**

```
java.lang.Object
  └ at.ac.univie.isc.jobmanager.servlet.job.JobManager
```

```
public class JobManager
extends java.lang.Object
```

**Author:**
Weichselbaum

**Constructor Summary**

| |
|---|
| JobManager(at.ac.univie.isc.jobmanager.servlet.DatabaseAgent db) |

**Method Summary**

| | |
|---|---|
| java.lang.String | addJobForm(at.ac.univie.isc.jobmanager.servlet.job.Job job)<br>Stores a new job in the database and adds it to the internal structure. |
| static java.lang.String | addJobGridEvent(at.ac.univie.isc.jobmanager.servlet.DatabaseAgent db, java.lang.String sJID, java.lang.String sUserLogin, java.lang.String sClientURI, java.lang.String sServiceURI, java.lang.String sOutputFilename, at.ac.univie.isc.client.proxy.AppProxyState apState)<br>Adds a job into the database. |
| java.lang.String | deleteJobFromDB(java.lang.String sUserLogin, java.lang.String sJobID)<br>Deletes a job from the database. |
| at.ac.univie.isc.jobmanager.servlet.job.Job | getJob(java.lang.String cid, java.lang.String jid)<br>Returns one specific job from internal structure. |
| at.ac.univie.isc.jobmanager.servlet.job.Job[] | getJobs()<br>Returns all jobs from internal structure as job array. |
| at.ac.univie.isc.jobmanager.servlet.job.Job[] | getJobs(java.lang.String sLogin)<br>Returns user jobs from internal structure as job array. |
| at.ac.univie.isc.jobmanager.servlet.job.Job[] | getJobs(java.lang.String sLogin, java.util.List<java.lang.String> arrSelectedUsers, java.util.List<java.lang.String> arrSelectedURIs, java.util.List<java.lang.String> arrState)<br>Returns jobs from internal structure as job array. |
| java.util.HashSet<java.lang.String> | getPossibleURIsForSelect()<br>Returns all possible service URIs for select criteria in menu item "Get Jobs"/"Get ALL Jobs". |
| void | init(at.ac.univie.isc.jobmanager.servlet.DatabaseAgent db)<br>Sets the database agent. |
| java.lang.String | killJob(java.lang.String sUserLogin, java.lang.String sJobID)<br>Kills a job. |
| void | readJobsFromDB()<br>Reads all jobs from the database. |
| void | readJobsFromDB(java.lang.String sLogin, boolean bPersonal)<br>Reads jobs from the database. |
| java.lang.String | startJob(java.lang.String sUserLogin, java.lang.String sJobID)<br>Starts a job. |
| java.lang.String | storeStateInDB(java.lang.String sUserLogin, java.lang.String sJobID) |

| | |
|---|---|
| | Storing job status in database (without getting new state). |
| java.lang.String | **updateJobStatus**(java.lang.String sUserLogin, java.lang.String sJobID)<br>Gets actual proxy status and does an update in the database. |
| static java.lang.String | **updateJobStatusInDB**(at.ac.univie.isc.jobmanager.servlet.DatabaseAgent db, java.lang.String sJobID, at.ac.univie.isc.client.proxy.AppProxyState apState)<br>Stores the actual state of a job in the database. |

| **Methods inherited from class java.lang.Object** |
|---|
| equals, getClass, hashCode, notify, notifyAll, toString, wait, wait, wait |

## Constructor Detail

### JobManager

```
public JobManager(at.ac.univie.isc.jobmanager.servlet.DatabaseAgent db)
```

**Parameters:**

`db` - database agent

## Method Detail

### init

```
public void init(at.ac.univie.isc.jobmanager.servlet.DatabaseAgent db)
```

Sets the database agent.

**Parameters:**

`db` - database agent

---

### readJobsFromDB

```
public void readJobsFromDB()
                  throws java.lang.Exception
```

Reads all jobs from the database.
Stores it in internal structure.

**Throws:**

`java.lang.Exception`

---

### readJobsFromDB

```
public void readJobsFromDB(java.lang.String sLogin,
                           boolean bPersonal)
                    throws java.lang.Exception
```

Reads jobs from the database.
bPersonal == TRUE ⇒ reads only own jobs
bPersonal == FALSE ⇒ reads all jobs where user has administration rights
Stores it in internal structure.

**Parameters:**

`sLogin` - login of user
`bPersonal` - TRUE if only own jobs should be read

**Throws:**

`java.lang.Exception`

---

### getJobs

```
public at.ac.univie.isc.jobmanager.servlet.job.Job[] getJobs()
                                                    throws java.lang.Exception
```

Returns all jobs from internal structure as job array.

**Returns:**
jobs as an array

**Throws:**
`java.lang.Exception`

---

### getJobs

```
public at.ac.univie.isc.jobmanager.servlet.job.Job[] getJobs(java.lang.String sLogin)
                                                    throws java.lang.Exception
```

Returns user jobs from internal structure as job array.

**Parameters:**
`sLogin` - login of user

**Returns:**
jobs as an array

**Throws:**
`java.lang.Exception`

---

### getJobs

```
public at.ac.univie.isc.jobmanager.servlet.job.Job[] getJobs(java.lang.String sLogin,
                                                  java.util.List<java.lang.String> arrSelectedUsers,
                                                  java.util.List<java.lang.String> arrSelectedURIs,
                                                  java.util.List<java.lang.String> arrState)
                                                        throws java.lang.Exception
```

Returns jobs from internal structure as job array.

**Parameters:**
`sLogin` - login of user
`arrSelectedUsers` - list of selected users
`arrSelectedURIs` - list of selected URIs
`arrState` - list of selected states

**Returns:**
selected jobs as an array

**Throws:**
`java.lang.Exception`

---

### getPossibleURIsForSelect

```
public java.util.HashSet<java.lang.String> getPossibleURIsForSelect()
```

Returns all possible service URIs for select criteria in menu item "Get Jobs"/"Get ALL Jobs".
HashSet setPossibleURIs is filled from readJobsFromDB

**Returns:**
possible URIs as select criteria

---

### getJob

```
public at.ac.univie.isc.jobmanager.servlet.job.Job getJob(java.lang.String cid,
                                                          java.lang.String jid)
```

Returns one specific job from internal structure.

**Parameters:**
`cid` - login of user
`jid` - job id

**Returns:**
the job

---

### addJobForm

```
public java.lang.String addJobForm(at.ac.univie.isc.jobmanager.servlet.job.Job job)
                            throws java.lang.Exception
```

Stores a new job in the database and adds it to the internal structure.
Called because of manual interaction from user (no event listener call)

**Parameters:**

`job` - the new job

**Returns:**

success or error message of database insert

**Throws:**

`java.lang.Exception`

---

### updateJobStatus

```
public java.lang.String updateJobStatus(java.lang.String sUserLogin,
                                        java.lang.String sJobID)
                                 throws java.lang.Exception
```

Gets actual proxy status and does an update in the database.
Calls internal Job.proxyRefreshStatus
and afterward updateJobStatusInDB if status has changed

**Parameters:**

`sUserLogin` - login of user

`sJobID` - job id

**Returns:**

success or error message of database update or null if no state change

**Throws:**

`java.lang.Exception`

---

### killJob

```
public java.lang.String killJob(java.lang.String sUserLogin,
                                java.lang.String sJobID)
                         throws java.lang.Exception
```

Kills a job.
Calls afterward updateJobStatusInDB with new state KILLED

**Parameters:**

`sUserLogin` - login of user

`sJobID` - job id

**Returns:**

success or error message of database update or null in case of proxy failure

**Throws:**

`java.lang.Exception`

---

### startJob

```
public java.lang.String startJob(java.lang.String sUserLogin,
                                 java.lang.String sJobID)
                          throws java.lang.Exception
```

Starts a job.
Calls internal updateJobStatusInDB with new state STARTED

**Parameters:**

`sUserLogin` - login of user

`sJobID` - job id

**Returns:**

success or error message of database update or null in case of proxy failure

**Throws:**

`java.lang.Exception`

---

### storeStateInDB

```
public java.lang.String storeStateInDB(java.lang.String sUserLogin,
                                       java.lang.String sJobID)
                                throws java.lang.Exception
```

Storing job status in database (without getting new state).

**Parameters:**

`sUserLogin` - login of user
`sJobID` - job id
**Returns:**
success or error message of database update
**Throws:**
`java.lang.Exception`

---

## deleteJobFromDB

```
public java.lang.String deleteJobFromDB(java.lang.String sUserLogin,
                                        java.lang.String sJobID)
                                 throws java.lang.Exception
```

Deletes a job from the database.
Calls internal job.kill before
**Parameters:**
`sUserLogin` - login of user
`sJobID` - job id
**Returns:**
success or error message of database update
**Throws:**
`java.lang.Exception`

---

## addJobGridEvent

```
public static java.lang.String addJobGridEvent(at.ac.univie.isc.jobmanager.servlet.DatabaseAgent db,
                                               java.lang.String sJID,
                                               java.lang.String sUserLogin,
                                               java.lang.String sClientURI,
                                               java.lang.String sServiceURI,
                                               java.lang.String sOutputFilename,
                                               at.ac.univie.isc.client.proxy.AppProxyState apState)
                                        throws java.lang.Exception
```

Adds a job into the database.
Static method - is called from JobManagerGridEventListener
**Parameters:**
`db` - database agent
`sJID` - job id
`sUserLogin` - login of user
`sClientURI` - uri of client
`sServiceURI` - uri of service
`sOutputFilename` - pattern of output file name for download
`apState` - application state (normally uploaded)
**Returns:**
success or error message of database insert
**Throws:**
`java.lang.Exception`

---

## updateJobStatusInDB

```
public static java.lang.String updateJobStatusInDB(at.ac.univie.isc.jobmanager.servlet.DatabaseAgent db,
                                                    java.lang.String sJobID,
                                                    at.ac.univie.isc.client.proxy.AppProxyState apState)
                                             throws java.lang.Exception
```

Stores the actual state of a job in the database.
Static method - is also called from JobManagerGridEventListener
**Parameters:**
`db` - database agent
`sJobID` - job id
`apState` - new job state
**Returns:**
success or error message of database update
**Throws:**
`java.lang.Exception`

### 9.2.2 DatabaseAgent für J2SE

at.ac.univie.isc.jobmanager.servlet

## Class DatabaseAgent

```
java.lang.Object
  └ at.ac.univie.isc.jobmanager.servlet.DatabaseAgent
```

```
public class DatabaseAgent
extends java.lang.Object
```

## Constructor Summary

| |
|---|
| **DatabaseAgent**(java.lang.String sServiceURI, java.lang.String sWorkingDir, java.lang.String sInputFilename, java.lang.String sOutputFilename) |

## Method Summary

| | |
|---|---|
| java.util.TreeMap<java.lang.String,java.util.Hashtable<java.lang.String,Job>> | **getJobs**(java.lang.String sSQLQuery)<br>Gets jobs from the database. |
| java.util.TreeMap<java.lang.String,User> | **getUsers**(java.lang.String sSQLQuery)<br>Gets users from the database. |
| int | **updateAction**(java.lang.String sSQLUpdate)<br>Performs a SQL update action in the database. |

### Methods inherited from class java.lang.Object

equals, getClass, hashCode, notify, notifyAll, toString, wait, wait, wait

## Constructor Detail

DatabaseAgent

```
public DatabaseAgent(java.lang.String sServiceURI,
                     java.lang.String sWorkingDir,
                     java.lang.String sInputFilename,
                     java.lang.String sOutputFilename)
```

**Parameters:**

`sServiceURI` - service URI
`sWorkingDir` - local working directory to save the OGSA-DAI response document
`sInputFilename` - file name or pattern on remote site - data service parameter
`sOutputFilename` - file name or pattern to download from remote site - data service parameter

## Method Detail

getUsers

```
public java.util.TreeMap<java.lang.String,User> getUsers(java.lang.String sSQLQuery)
                                                  throws java.lang.Exception
```

Gets users from the database.

**Parameters:**

`sSQLQuery` - SQL statement

**Returns:**

TreeMap with the fetched users

**Throws:**

`java.lang.Exception`

---

## getJobs

```
public java.util.TreeMap<java.lang.String,java.util.Hashtable<java.lang.String,Job>>
                                                              getJobs(java.lang.String sSQLQuery)
                                                              throws java.lang.Exception
```

Gets jobs from the database.

**Parameters:**

`sSQLQuery` - SQL statement

**Returns:**

TreeMap with the fetched jobs

**Throws:**

`java.lang.Exception`

---

## updateAction

```
public int updateAction(java.lang.String sSQLUpdate)
                 throws java.lang.Exception
```

Performs a SQL update action in the database.

**Parameters:**

`sSQLUpdate` - SQL update statement (update, delete, insert)

**Returns:**

number of changed records in the database

**Throws:**

`java.lang.Exception`

### 9.2.3 JobManager für J2ME

at.ac.univie.isc.jobmanager.mobile.job

## Class JobManager

```
java.lang.Object
  └ at.ac.univie.isc.jobmanager.mobile.job.JobManager
```

```
public class JobManager
extends java.lang.Object
```

## Constructor Summary

| |
|---|
| **JobManager**() |

## Method Summary

| | |
|---|---|
| Job | **getJob**(java.lang.String jid)<br>Returns one specific job from internal structure. |
| Job[] | **getJobs**(boolean bReadFromDB)<br>Returns all jobs from internal structure as job array. |
| void | **init**(DatabaseAgent db, User user)<br>Initializes the JobManager (setting the database agent and the user). |
| void | **killJob**(Job job)<br>Kills a job. |
| void | **readJobsFromDB**()<br>Reads all jobs from actual user with state == AppProxyState.STARTED and stores it in internal structure. |
| void | **updateJobStatus**(Job job)<br>Gets actual proxy status and does an update in the database. |

| Methods inherited from class java.lang.Object |
|---|
| equals, getClass, hashCode, notify, notifyAll, toString, wait, wait, wait |

## Constructor Detail

JobManager

```
public JobManager()
```

## Method Detail

init

```
public void init(DatabaseAgent db,
                 User user)
```

Initializes the JobManager (setting the database agent and the user).

**Parameters:**

`db` - database agent

`user` - actual user to store internal

### readJobsFromDB

```
public void readJobsFromDB()
                    throws java.lang.Exception
```

Reads all jobs from actual user with state == AppProxyState.STARTED and stores it in internal structure.

**Throws:**

`java.lang.Exception`

---

### getJobs

```
public Job[] getJobs(boolean bReadFromDB)
             throws java.lang.Exception
```

Returns all jobs from internal structure as job array.
Calls internal readJobsFromDB before if parameter bReadFromDB == true

**Parameters:**

`bReadFromDB` - TRUE if jobs should be read from database

**Returns:**

jobs of a user as an array

**Throws:**

`java.lang.Exception`

---

### getJob

```
public Job getJob(java.lang.String jid)
```

Returns one specific job from internal structure.

**Parameters:**

`jid` - job id

**Returns:**

the job

---

### updateJobStatus

```
public void updateJobStatus(Job job)
                    throws java.lang.Exception
```

Gets actual proxy status and does an update in the database.
Calls internal Job.proxyRefreshStatus
and afterward updateJobStatusInDB if status has changed

**Parameters:**

`job` - actual job

**Throws:**

`java.lang.Exception`

---

### killJob

```
public void killJob(Job job)
             throws java.lang.Exception
```

Kills a job.
Calls afterward updateJobStatusInDB with new state KILLED

**Parameters:**

`job` - actual job

**Throws:**

`java.lang.Exception`

### 9.2.4 DatabaseAgent für J2ME

at.ac.univie.isc.jobmanager.mobile

## Class DatabaseAgent

```
java.lang.Object
  └ at.ac.univie.isc.jobmanager.mobile.DatabaseAgent
```

```
public class DatabaseAgent
extends java.lang.Object
```

## Constructor Summary

| |
|---|
| DatabaseAgent() |

## Method Summary

| | |
|---|---|
| java.util.Hashtable | getJobs(java.lang.String sSQLQuery)<br>Gets jobs from the database. |
| User | getUser(java.lang.String sSQLQuery)<br>Gets one user from the database. |
| void | init(java.lang.String sServiceURI, java.lang.String sInputFilename)<br>Initializes the DatabaseAgent. |
| int | updateAction(java.lang.String sSQLUpdate)<br>Performs a SQL update action in the database. |

| Methods inherited from class java.lang.Object |
|---|
| equals, getClass, hashCode, notify, notifyAll, toString, wait, wait, wait |

## Constructor Detail

DatabaseAgent

```
public DatabaseAgent()
```

## Method Detail

init

```
public void init(java.lang.String sServiceURI,
                 java.lang.String sInputFilename)
```

Initializes the DatabaseAgent.

**Parameters:**

`sServiceURI` - service URI

`sInputFilename` - file name or pattern on remote site (data service parameter)

### getUser

```
public User getUser(java.lang.String sSQLQuery)
            throws java.lang.Exception
```

Gets one user from the database.

**Parameters:**

`sSQLQuery` - SQL statement

**Returns:**

the fetched user

**Throws:**

`java.lang.Exception`

---

### getJobs

```
public java.util.Hashtable getJobs(java.lang.String sSQLQuery)
                           throws java.lang.Exception
```

Gets jobs from the database.

**Parameters:**

`sSQLQuery` - SQL statement

**Returns:**

Hashtable with the fetched jobs

**Throws:**

`java.lang.Exception`

---

### updateAction

```
public int updateAction(java.lang.String sSQLUpdate)
                throws java.lang.Exception
```

Performs a SQL update action in the database.

**Parameters:**

`sSQLUpdate` - SQL update statement

**Returns:**

number of changed records in the database

**Throws:**

`java.lang.Exception`

# 10 Abkürzungsverzeichnis

| | |
|---|---|
| API | Application Programming Interface |
| ASCII | American Standard Code for Information Interchange |
| CORBA | Common Object Request Broker Architecture |
| CVS | Concurrent Versions System |
| DIME | Direct Internet Message Encapsulation |
| DNS | Domain Name System |
| DOM | Document Object Model |
| FTP | File Transfer Protocol |
| GGF | Global Grid Form |
| GSM | Global System for Mobile Communications |
| GUI | Graphical User Interface |
| HPC | High-performance computing |
| HTML | Hypertext Markup Language |
| HTTP | Hypertext Transfer Protocol |
| HTTPS | Hypertext Transfer Protocol Secure |
| ICMP | Internet Control Message Protocol |
| IMEI | International Mobile Equipment Identity |
| IP | Internet Protocol |
| JavaRMI | Java Remote Method Invocation |
| JSR | Java Specification Request |
| JTWI | Java Technology for the Wireless Industry |
| KB | Kilobyte |
| KVM | Kilobyte Virtual Machine |
| MD5 | Message-Digest Algorithm 5 |
| MID | Mobile Information Device |
| MIME | Multipurpose Internet Mail Extensions |
| MTOM | Message Transmission Optimization Mechanism |
| OASIS | Organization for the Advancement of Structured Information Standards |
| OGF | Open Grid Form |
| OGSA | Open Grid Service Architecture |
| OGSA-DAI | Open Grid Services Architecture Data Access and Integration |
| OSPF | Open Shortest Path First |

| | |
|---|---|
| PDA | Personal Digital Assistant |
| PDKD | Parallel and Distributed Knowledge Discovery |
| PKI | Public Key Infrastructure |
| QoS | Quality of Service |
| RPC | Remote Procedure Call |
| RSVP | Resource Reservation Protocol |
| SAX | Simple API for XML |
| SDK | Software Development Kit |
| SETI | Search for extraterrestrial intelligence at home |
| SMTP | Simple Mail Transfer Protocol |
| SOA | Service-Oriented Architecture |
| SOAP | Simple Object Access Protocol |
| TCP | Transmission Control Protocol |
| tModel | taxonomy model |
| UDDI | Universal Description Discovery and Integration |
| UDP | User Datagram Protocol |
| URL | Uniform Resource Locator |
| UUID | Universally Unique Identifier |
| VGE | Vienna Grid Environment |
| VO | Virtuelle Organisation |
| W3C | World Wide Web Consortium |
| WS | Web Services |
| WSDL | Web Services Description Language |
| WS-I | Web Services Interoperabilty |
| WSRF | Web Services Resource Framework |
| XML | Extensible Markup Language |
| XOP | XML Binary Optimized Packaging |
| XSL | Extensible Stylesheet Language |
| XSLT | XSL Transformation |

# 11 Literaturverzeichnis

**[Ben]** Siegfried Benkner, Ivona Brandic, Gerhard Engelbrecht, Rainer Schmidt: VGE - A Service-Oriented Grid Environment for On-Demand Supercomputing (2004).

**[Ben08]** Siegfried Benkner, Gerhard Engelbrecht, Martin Köhler, Alexander Wöhrer: Virtualizing scientific applications and data sources as grid services (2008).

**[Bot04]** Miguel L. Bote-Lorenzo, Yannis A. Dimitriadis, Eduardo Gomez-Sanchez: Grid Characteristics and Uses: a Grid Definition (Santiago de Compostela, Spanien, 2004).

**[Cau01]** Patrick Cauldwel, Rajesh Chawla, Vivek Chopra, Gary Damschen, Chris Dix, Tony Hong, Francis Norton, Uche Ogbuji, Mark A. Richman, Kristy Saunders, Zoran Zaev: Professional XML Web Services (Birmingham, 2001).

**[Dos05]** Wolfgang Dostal, Mario Jeckle, Ingo Melzer, Barbara Zengler: Service-orientierte Architekturen mit Web Services (München, 2005).

**[Fos04]** Ian Foster, Carl Kesselman: The Grid 2: Blueprint for a New Computing Infrastructure (San Francisco, 2004).

**[Fos98]** Ian Foster, Carl Kesselman: The Grid: Blueprint for a new computing infrastructure (San Francisco, 1998).

**[Kro02]** Michael Kroll, Stefan Haustein: Java™ 2 Micro Edition Application Development (2002).

**[Kus02]** Michael Kuschke, Ludger Wölfel: Web Services kompakt (Heidelberg, 2002).

**[New02]** Eric Newcomer: Understanding Web services (Boston, 2002).

**[Top02]** Kim Topley: J2ME in a Nutshell (2002).

**[Zhu04]** Hai Zhuge: The Knowledge Grid (Singapore, 2004).

Onlinequellen:

**[Boo07]** David Booth, Canyang Kevin Liu: Web Services Description Language (WSDL) Version 2.0 Part 0: Primer (2007)
[http://www.w3.org/TR/wsdl20-primer/] (Zugriff: 6. Apr. 2008).

**[Box00]** Don Box, David Ehnebuske, Gopal Kakivaya, Andrew Layman, Noah Mendelsohn, Henrik Frystyk Nielsen, Satish Thatte, Dave Winer: Simple Object Access Protocol (SOAP) 1.1 (2000)
[http://www.w3.org/TR/2000/NOTE-SOAP-20000508/] (Zugriff: 14. Feb. 2008).

**[Bre07]** Peter Brezany: Informationssysteme in vernetzten Systemen (2007)
[http://www.par.univie.ac.at/~brezany/teach/kfk/03ws-vo/skriptum/03-10-07-intro.ppt] (Zugriff: 17. März 2008).

**[Bri01]** Peter Brittenham, Francisco Cubera, Dave Ehnebuske, Steve Graham: Understanding WSDL in a UDDI registry (2001)
[http://www.ibm.com/developerworks/webservices/library/ws-wsdl/] (Zugriff: 24. Mai 2008).

**[Chr01]** Erik Christensen, Francisco Curbera, Greg Meredith, Sanjiva Weerawarana: Web Services Description Language (WSDL) 1.1 (2001)
[http://www.w3.org/TR/wsdl] (Zugriff: 2. Apr. 2008).

**[Cle04]** Luc Clement, Andrew Hately, Claus von Riegen, Tony Rogers: Universal Description, Discovery and Integration v3.0.2 (UDDI) (2004)
[http://www.oasis-open.org/committees/uddi-spec/doc/spec/v3/uddi-v3.0.2-20041019.htm] (Zugriff: 15. Apr. 2008).

**[Coh03]** Frank Cohen: Discover SOAP encoding's impact on Web service performance (2003)
[http://www.ibm.com/developerworks/webservices/library/ws-soapenc/] (Zugriff: 24. Feb. 2008).

**[Fos]** Ian Foster, Carl Kesselman, Steven Tuecke: The Anatomy of the Grid: Enabling Scalable Virtual Organizations ()
[http://www.globus.org/alliance/publications/papers/anatomy.pdf] (Zugriff: 16. Mai 2008).

**[Fos02]** Ian Foster: What is the Grid? A Three Point Checklist (2002)
[http://www-fp.mcs.anl.gov/~foster/Articles/WhatIsTheGrid.pdf] (Zugriff: 3. Mai 2008).

**[Fos03]** Ian Foster: Introduction to the Grid (2003)
[http://www-fp.mcs.anl.gov/~foster/Talks/Comdex%20Grid%20Foster%20November%202003.ppt] (Zugriff: 27. Mai 2008).

**[Fos06]** Ian Foster, H. Kishimoto, A. Savva: The Open Grid Services Architecture, Version 1.5 (2006)
[http://www.ogf.org/documents/GFD.80.pdf] (Zugriff: 1. Juni 2008).

**[Fre96]** N. Freed, N. Borenstein: RFC2045: Multipurpose Internet Mail Extensions (MIME) Part One: Format of Internet Message Bodies (1996)
[http://tools.ietf.org/html/rfc2045] (Zugriff: 2. März 2008).

**[Fre961]** N. Freed, N. Borenstein: RFC2046: Multipurpose Internet Mail Extensions (MIME) Part Two: Media Types (1996)
[http://tools.ietf.org/html/rfc2046] (Zugriff: 3. März 2008).

**[Gri]** : Grid-Computing ()
[http://de.wikipedia.org/wiki/Grid-Computing] (Zugriff: 4. Juni 2008).

**[Gud05]** Martin Gudgin, Noah Mendelsohn, Mark Nottingham, Hervé Ruellan: XML-binary Optimized Packaging (2005)
[http://www.w3.org/TR/xop10/] (Zugriff: 23. Apr. 2008).

**[Haa04]** Hugo Haas, Allen Brown: Web Services Glossary (2004)
[http://www.w3.org/TR/ws-gloss/] (Zugriff: 20. Feb. 2008).

**[J2M00]** : J2ME Building Blocks for Mobile Devices (2000)
[http://java.sun.com/products/cldc/wp/KVMwp.pdf] (Zugriff: 5. Okt. 2008).

**[Jos06]** S. Josefsson: The Base16, Base32, and Base64 Data Encodings (2006)
[http://tools.ietf.org/html/rfc4648] (Zugriff: 9. Apr. 2008).

**[Mil06]** Mike Milikich, James Warden: JSR 118: Mobile Information Device Profile 2.0 (2006) [http://jcp.org/en/jsr/detail?id=118] (Zugriff: 15. Okt. 2008).

**[Mit07]** Nilo Mitra, Yves Lafon: SOAP Version 1.2 Part 0: Primer (Second Edition) (2007) [http://www.w3.org/TR/soap12-part0/] (Zugriff: 30. März 2008).

**[Mor08]** Rajiv Mordani: JSR 315: JavaTM Servlet 3.0 Specification (2008) [http://jcp.org/en/jsr/detail?id=315] (Zugriff: 2. Sep. 2008).

**[Nel97]** S. Nelson, C. Parks, Mitra: RFC2077: The Model Primary Content Type for Multipurpose Internet Mail Extensions (1997) [http://tools.ietf.org/html/rfc2077] (Zugriff: 7. März 2008).

**[OAS]** : OASIS Web Services Resource Framework (WSRF) TC () [http://www.oasis-open.org/committees/tc_home.php?wg_abbrev=wsrf] (Zugriff: 6. Juli 2008).

**[OGS]** : OGSA-DAI 2.2 Architecture (2002-2006) [http://www.ogsadai.org.uk/documentation/ogsadai-wsi-2.2/doc/background/architecture.html] (Zugriff: 13. Juli 2008).

**[OGS1]** : OGSA-DAI 3.0 Data resource product information (2007) [http://www.ogsadai.org.uk/documentation/ogsadai3.0/ogsadai3.0-axis/DataResourceProducts.html] (Zugriff: 22. Aug. 2008).

**[OGS2]** : OGSA-DAI 2.2 Interacting with Data Service Resources () [http://www.ogsadai.org.uk/documentation/ogsadai-wsrf-2.2/doc/interaction/index.html] (Zugriff: 21. Juli 2008).

**[OGS3]** : OGSA-DAI 2.2 Activities () [http://www.ogsadai.org.uk/documentation/ogsadai-wsrf-2.2/doc/interaction/Activities.html] (Zugriff: 5. Juli 2008).

**[Pow03]** Matt Powell: Grundlegendes zu DIME und WS-Attachments (2003) [http://msdn.microsoft.com/de-de/library/cc405452.aspx] (Zugriff: 17. Apr. 2008).

**[Prä06]** Ralf Prändl, Uwe Albert: Übertragung von Massendaten auf Basis von Open-Source-Technologien (2006) [http://www.sigs.de/publications/os/2006/06/praendl_albert_OS_06_06.pdf] (Zugriff: 27. Mai 2008).

**[Rig06]** Roger Riggs: JSR 218: Connected Device Configuration (CDC) 1.1 (2006) [http://www.jcp.org/en/jsr/detail?id=218] (Zugriff: 19. Okt. 2008).

**[Rig07]** Roger Riggs: JSR 139: Connected Limited Device Configuration 1.1 (2007) [http://jcp.org/en/jsr/detail?id=139] (Zugriff: 26. Okt. 2008).

**[Riv92]** Ronald L. Rivest: RFC1321: The MD5 Message-Digest Algorithm (1992) [http://tools.ietf.org/html/rfc1321] (Zugriff: 7. Nov. 2008).

**[Sch05]** Robert Schrader: Standardisierungen in der Grid Comunity (2005) [http://www.uni-koblenz.de/FB4/Institutes/IFI/AGStaab/Teaching/WS0405/seminar_semGrid/docs/I.2-GridStandards_Praesentation.pdf] (Zugriff: 28. Juli 2008).

**[Sem]** : Semantic Grid [http://www.semanticgrid.org/] (Zugriff: 17. Juni 2008).

**[Shi08]** Sang Shin, Carol McDonald: Developing Real World Web Services Using J2ME, J2SE, J2EE (2008)
[http://www.powwwerpages.com/ebooks/computer/Developing_real_world_Web_services.pdf] (Zugriff: 16. Sep 2008).

**[Szc06]** Sebastian Szczygiel, Lars Röwekamp: Application Development in J2ME (2006)
[http://www.areamobile.de/specials/J2ME/teil_I.php] (Zugriff: 29. Sep. 2008).

**[Tap]** Carlos C. Tapang: Web Services Description Language (WSDL) im Überblick ()
[http://www.microsoft.com/germany/msdn/library/xmlwebservices/WebServicesDescriptionLanguageWSDLImUeberblick.mspx?mfr=true] (Zugriff: 11. März 2008).

**[Tob03]** Tobias Frech: Mobile / Wireless Applications mit J2ME am Beispiel eines Instant-Messaging-Clients (2003)
[http://opus.bsz-bw.de/hdms/volltexte/2004/420/pdf/Diplomarbeit_Tobias_Frech.pdf] (Zugriff: 25. Sep. 2008).

**[Ver04]** Johann Verbovski: J2ME (2004)
[www.db.informatik.uni-kassel.de/Lehre/WS0405/SeminarPI/J2me.pdf] (Zugriff: 8. Sep. 2008).

**[Wah04]** Wadatullah Wahidie: Das KnowledgeGrid (2004)
[http://www.is.informatik.uni-duisburg.de/courses/grid_ss04/wahidie.pdf] (Zugriff: 13. Juli 2008).

**[Win99]** Dave Winer: XML-RPC Specification (1999)
[http://www.xmlrpc.com/spec] (Zugriff: 13. Feb. 2008).

**[Wöh06]** Alexander Wöhrer: GDMS Service Proforma ()
[http://www.nesc.ac.uk/action/esi/download.cfm?index=1701] (Zugriff: 18. Feb. 2008).

**[Won02]** William Wong: Write Once, Debug Everywhere (2002)
[http://electronicdesign.com/Articles/Index.cfm?ArticleID=2255] (Zugriff: 7. Sep. 2008).

**[WSI]** : WS-I Organization's Web site
[http://www.ws-i.org/] (Zugriff: 22. Aug. 2008).

**[Yan07]** Young Yang: Faster Data Transport Means Faster Web Services with MTOM/XOP (2007)
[http://www.devx.com/xml/Article/34797] (Zugriff: 29. Apr. 2008).

Ich habe mich bemüht, sämtliche Inhaber der Bildrechte ausfindig zu machen und ihre Zustimmung zur Verwendung der Bilder in dieser Arbeit eingeholt. Sollte dennoch eine Urheberrechtsverletzung bekannt werden, ersuche ich um Meldung bei mir.

# *ABSTRAKT*

Grid-Systeme sind durch ihren dezentralen Charakter bestimmt. Eine zentrales Job- und User-Management ist daher nicht vorgesehen. Diverse Szenarien sprechen jedoch für dessen Sinnhaftigkeit. Der Job-Manager ermöglicht eine Visualisierung der Nutzung von Diensten in dem Vienna Grid Environment (VGE). Das VGE ist ein Software-System zur Realisierung einer Grid-Umgebung und wurde vom Institut für Scientific Computing der Universität Wien entwickelt.

Der Job-Manager bietet die Möglichkeit, auf einfache Weise Jobs in einem Grid-System zu zentralisieren bzw. zu visualisieren und gemeinsam zu verwalten. Die Nutzung von Services bzw. die daraus resultierenden Jobs werden protokolliert und relevante Job-Daten persistent gespeichert. Jobs von mehreren Clients gestartet, können gemeinsam mit dem Job-Manager administriert werden; d.h. den aktuellen Status abfragen, auf die Ergebnisse zugreifen bzw. den Job abbrechen. Der Job-Manager ermöglicht dem Benutzer, seine Jobs oder die Jobs von anderen Benutzern, für die er berechtigt ist, zu administrieren. Diverse Selektions- und Sortierkritieren erleichtern die Bedienbarkeit. Eine Benutzerverwaltung ist eine zusätzliche Funktionalität, durch die eine Authentifizierung des Benutzers ermöglicht wird.

Der Job-Manager ist eine funktionale Erweiterung und kann ohne Architektur-Veränderungen in das bestehende VGE-System integriert werden. Er ist in zwei Ausprägungen realisiert: als Web-Client und als Client für mobile Endgeräte. Der Client für mobile Endgeräte, auch als „Personal Job Manager“ bezeichnet, ermöglicht einem Benutzer, die durch ihn initiierten Jobs „over-the-air“ zu administrieren.

Zur Realisierung des Job-Managers wurden Technologien wie Web Services, SOAP, WSDL und UDDI als auch die Übertragung von Attachments durch MIME, DIME und MTOM behandelt. Grid-Systeme, WSRF, OGSA-DAI, sind weitere, untersuchte Technologieschwerpunkte, ebenso wie die Realisierung von Java-Applikationen für mobile Endgeräte.

# *Lebenslauf*

**Persönliche Daten:**

| | |
|---|---|
| Vor- und Nachname: | Jürgen Weichselbaum |
| Geburtstag: | 19.07.1974 |
| Geburtsort: | Krems |
| Staatsangehörigkeit: | Österreich |
| Familienstand: | verheiratet, 2 Kinder |

**Berufsausbildung**:

| | |
|---|---|
| 1988 – 1993 | HTL in St. Pölten<br>Abteilung: Datenverarbeitung und Organisation |

**Präsenzdienst**:

| | |
|---|---|
| 1993 –1994 | 8 Monate Präsenzdienst in Mautern |

**Studium**:

| | |
|---|---|
| 1994 - 1999 | Studium Wirtschaftsinformatik an der Universität Wien |
| 2008 | Fortsetzung bzw. Abschluß des Studiums<br>Diplomarbeit „Job-Management in einem Grid-System, realisiert als Web-Servlet und MIDlet“ |

**Berufstätigkeit**:

| | |
|---|---|
| 1998 | EBF-Leasing |
| 05/1999 – 02/2008 | Siemens AG |