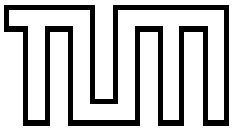
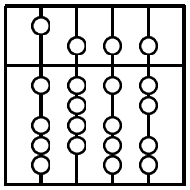
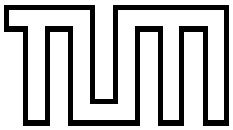


Prozess-orientiertes Monitoring virtueller Organisationen in Globus-basierten Grids

Bearbeiter: Herve Amikem Chemeza

Diplomarbeit in Informatik





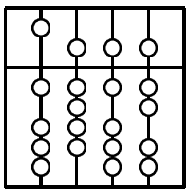
FAKULTÄT FÜR INFORMATIK
TECHNISCHE UNIVERSITÄT MÜNCHEN

Diplomarbeit in Informatik

Prozess-orientiertes Monitoring virtueller Organisationen in Globus-basierten Grids

Bearbeiter: Herve Amikem Chemeza
Aufgabensteller: Prof. Dr. Heinz-Gerd Hegering
Betreuer: Michael Schiffers
Nils gentschen Felde

Abgabedatum: 11. September 2007



Ich versichere, dass ich diese Diplomarbeit selbständig verfasst und nur die angegebenen Quellen und Hilfsmittel verwendet habe.

München, den 11. September 2007

.....

(Unterschrift des Kandidaten)

In den vergangenen Jahren ist das Interesse für innovative Rechenmodelle und IT-Infrastrukturen stark gewachsen. Zu diesen Innovationen zählt das Grid Computing, das vor allem in der Forschung und in der Industrie eingesetzt wird. Das Grid stellt einen Zusammenschluss mehrerer, einzelner Rechner, Supercomputer, Datenbanken, etc., in Form von Ressourcen oder Diensten dar, die die Bearbeitung verteilter Anwendungen ermöglichen. Eine Software kontrolliert diesen Zusammenschluss und verteilt Aufgaben im Grid, also auf die einzelnen Ressourcen. So können Ressourcen aus verschiedenen, weltweit verteilten Institutionen, Universitäten oder Forschungseinrichtungen einer gemeinsamen Nutzung zugeführt werden. Wichtig dabei ist, dass diese Institutionen oder Personen, die Informationen austauschen und kollaborieren, ihre Ressourcen und Dienste derart koordinieren, so dass ein gemeinsames Ziel erreicht wird. So ein Konsortium von Organisationen nennen wir Virtuelle Organisation (VO). Eine VO zeichnet sich hauptsächlich durch ihre dynamischen und multiinstitutionellen Merkmale aus, insofern ist das Management und das Monitoring von VOs von zentraler Bedeutung. Es soll z.B. möglich sein, den Lebenszyklus einer VO, den Status ihrer Mitglieder und ihrer Ressourcen zu überwachen.

Im Rahmen dieser Arbeit wird ein Konzept für das Monitoring von VOs in Globus-basierten Grids vorgestellt. Das Globus Toolkit ist eine Grid-Middelware, die derzeit als de facto Standard gilt und die neue Standard Grid-Architektur (OGSA) implementiert. Es soll genau erarbeitet werden, wie VO Membership Dienste und Grid Ressource Management Dienste integriert werden können, so dass daraus VO-Monitoring Dienste entstehen. Dafür wird zunächst anhand einiger Szenarien ein Anforderungskatalog ausgearbeitet. Darauf aufbauend wird der Stand der Technik vorgestellt und analysiert. Eine Analyse der vorhandenen Möglichkeiten zeigt, dass diese die erstellten Anforderungen nicht vollständig erfüllen. Daher wird im zweiten Teil ein neues VO-Monitoringkonzept vorgestellt und anschließend in einer prozess-orientierten Architektur implementiert und bewertet.

Inhaltsverzeichnis

1 Einleitung **1**
- 1.1 Motivation und Aufgabenstellung . . . 1
- 1.2 Vorgehensweise der Arbeit . . . 2
- 1.3 Gliederung der Arbeit . . . 3

2 Grundlagen **5**
- 2.1 Einführung in Grid Computing . . . 5
- 2.2 Virtuelle Organisation . . . 6
- 2.3 Grid Architektur . . . 8
- 2.4 Globus Toolkit . . . 11
- 2.5 Management von Mitgliedschaften zu VOs . . . 13
- 2.6 Zusammenfassung . . . 15

3 Anforderungsanalyse **16**
- 3.1 Szenarien . . . 16
  - 3.1.1 Szenario 1: VO-Aktivitäten . . . 16
  - 3.1.2 Szenario 2: VO-Monitoring . . . 17
- 3.2 Erstellung des Anforderungskatalogs . . . 17
- 3.3 Beschreibung eines Testbetts . . . 19
  - 3.3.1 Bereitstellung eines Dienstes für zuverlässigen Dateitransfer . . . 19
  - 3.3.2 Bereitstellung eines Dienstes für entfernte Job-Ausführung . . . 19
  - 3.3.3 Bereitstellung von Diensten für die Registrierung und Entdeckung von Grid-Ressourcen 20
  - 3.3.4 Einrichtung einer Test-VO . . . 21
- 3.4 Zusammenfassung . . . 21

4 State-of-the-Art im VO-Monitoring **22**
- 4.1 Management von VOs . . . 22
  - 4.1.1 CAS (Community Authorization Service) . . . 22
  - 4.1.2 GridShib/Shibboleth . . . 23
  - 4.1.3 VOMS/VOMRS . . . 25
- 4.2 Monitoring von Grid-Ressourcen . . . 26
- 4.3 Zusammenfassung . . . 28

5 Erstellung eines VO-Monitoringkonzepts **30**
- 5.1 Grid Resource Monitoring Service . . . 30
  - 5.1.1 Index Service Funktionsweise . . . 30
  - 5.1.2 Semantik der Query API und Struktur der Daten . . . 32
- 5.2 VO Membership Administration Service (VOMSAdmin) . . . 38
- 5.3 Integration von VOMS und Resource Monitoring Service für Monitoringszwecke . . . 38
  - 5.3.1 Problemstellung . . . 39
  - 5.3.2 Integrationskonzept . . . 39
  - 5.3.3 Plugin zur Integration von VOMS und Grid Monitoring Service . . . 42
- 5.4 Prozess-orientiertes VO-Monitoring . . . 46
  - 5.4.1 Einführung in die Prozess-orientierte Architektur (POA) . . . 47
  - 5.4.2 Modellierung von Business Prozessen (Workflow) . . . 48
  - 5.4.3 Das VO-Monitoringkonzept . . . 49

5.5 Realisierung des Konzepts für andere VO-Management Systeme 52
5.6 Zusammenfassung 53

6 Konzeptimplementierung 54
6.1 VO-Monitor Architektur 54
6.1.1 Das VO-Monitoring System 54
6.1.2 Das Frontend System 55
6.2 Datenbank Service 55
6.2.1 Datenbank Schema 56
6.2.2 Web Services Beschreibung 58
6.3 BPEL-Prozess-Beschreibung 61
6.3.1 Beschreibung der Prozessaktivitäten 66
6.4 Zusammenfassung 68

7 Bewertung 69
7.1 Bewertung der funktionalen Anforderungen 69
7.2 Bewertung der nicht-funktionalen Anforderungen 70

8 Zusammenfassung und Ausblick 72
8.1 Zusammenfassung 72
8.2 Ausblick 72

A Installation und Bedienung der Applikation 75
A.1 Verzeichnisstruktur der beigefügten CD 75
A.2 Installationsanleitung 75
A.2.1 Das VO-Monitor System 75
A.2.2 MDS-Plugin 77

B WS-BPEL Spezifikation 78

C BPEL-Prozess des VO-Monitor 86

D GLUE Schema 100
D.1 Das Computing Element (CE) Schema 100

Abbildungsverzeichnis

1.1 Vorgehensweise 3

2.1 VO-Beispiele 7
2.2 Die geschichtete Grid Architektur und ihre Beziehung zur Internet Protokoll Architektur. Da sich die Internet Protokoll Architektur von der Netzwerkschicht bis zur Applikationschicht ausstreckt, ist eine Abbildung der Grid Schichten zu Internet-Schichten möglich. [ANAT] . . . 8
2.3 OGSA Framework [OGSAv1.5] 10
2.4 Bestandteile des Globus Toolkits 4 [GT4] 12
2.5 GT4 Monitoring and Discovery System,[GT4-Paper] 13
2.6 Die VOMS Architektur [VOMS] 15

3.1 Interaktionen in einer VO 17
3.2 Einsatz von WS GRAM [GT4] 20

4.1 Shibboleth [GridShib2] 24
4.2 Klassisches GridShib Szenario [GridShib1] 25
4.3 VOMRS Architektur [VOX] 26
4.4 Informationsfluss in MDS4 [GT4] 27
4.5 Bildung eines VO-Index mit MDS4 [GT4] 29

5.1 Informationsfluss bei der Publikation der Resource Properties zum Index Service 31
5.2 WS ServiceGroup UML Diagramm [WS-SGr] 33
5.3 Bildung und Verwendung von VO-Indexen 40
5.4 Einfacher VO-Monitor 40
5.5 VO Monitoring Konzept 41
5.6 Plugin Architektur 43
5.7 Beispiel einer prozess-orientierten Architektur [Fran 03] 47
5.8 VO Monitoring Anwendungsfall-Diagramm 50
5.9 VO-Monitor Aktivitätdiagramm 51
5.10 VO-Monitoring Anwendungsfalldiagramm angepasst für CAS oder Shibboleth. 52

6.1 VO-Monitor Architektur 55
6.2 Datenbankschema 56
6.3 Aktivitätsdiagramm des VO-Monitors (annotierte Version) 62

A.1 ActiveBPEL Konfiguration 76

D.1 Computing Element UML-Klassendiagramm [GLUE] 101
D.2 Beschreibung des Elements `Computing Element` [GLUE] 102
D.3 Beschreibung der Elemente `Cluster` und `SubCluster` [GLUE] 102
D.4 Beschreibung des Elements `Host` [GLUE] 103

Tabellenverzeichnis

2.1 VO-Eigenschaften 8

4.1 Bewertung der existierenden Tools 29

5.1 Bewertung der existierenden Tools mit Integrationskonzept 41

1 Einleitung

Die wachsende Innovation in der Breitband-Technologie und der steigende Bedarf an Rechenleistung und Speicherkapazität haben die Einführung von neuen IT-Infrastrukturen wie *Grids* vorangetrieben. *Grids* ermöglichen eine gemeinsame und koordinierte Nutzung von weltweit verteilten Ressourcen wie Speicherkapazitäten, Rechenleistung oder Datenbanken über das Internet. Sie bieten verlässliche und sichere Mechanismen für das Management, das Entdecken und den Zugang zu entfernten Ressourcen in Form von Diensten oder physischen Ressourcen wie Supercomputer, Speichereinheiten, Datenbanken oder PCs (Personal Computer). Grid Computing wird charakterisiert durch:

- den Zugriff auf verteilte Ressourcen. Das Management dieser Ressourcen erfolgt dabei dezentral nach lokalen Policies,
- eine gemeinsame und koordinierte Problemlösungsstrategie, und
- die Bildung von *virtuellen Organisationen (VOs)*. Eine VO ist ein Konsortium verschiedener Personen oder Institutionen, die Informationen austauschen, kollaborieren und ihre Ressourcen und Dienste derart koordinieren, dass ein gemeinsames Ziel erreicht wird [Grid 1].

Das Konzept von Grid Computing wurde zum ersten Mal von *Ian Foster* und *Carl Kesselmann* in ihrem Buch „*The Grid: Blueprint for a New Computing Infrastructure*“ [Grid 1] vorgestellt. Aus dieser Initiative ist der aktuelle de facto Standard für Grid-Middleware, das Globus Toolkit [GT4], entstanden.

1.1 Motivation und Aufgabenstellung

Eine VO wird gewöhnlich aufgebaut, indem mehrere Organisationen (z.B. Rechenzentren, Forschungszentren, Universitäten) im Rahmen einer Kooperation Teile ihrer Infrastruktur einer gemeinsamen Nutzung zuführen. Die VO bildet somit eine zentrale Einheit, in der sich die verwendeten Ressourcen jedoch nach wie vor dezentral bei den realen Organisationen befinden. Ein besonderes Merkmal von VOs ist deren Dynamik. Sie können stark nach Zielen, Philosophie, Umfang, Dauer, Struktur oder Art und Anzahl von Teilnehmern variieren. In einer VO können beispielsweise verschiedene Institutionen Teile ihrer Hardware und Software derart zusammenstellen, dass ein einziger Dienst (z.B. Datentransfer) zur Verfügung gestellt wird. Ein so bereitgestellter Dienst könnte beispielsweise nur für eine fest definierte Zeit existieren. Die Zugriffsrechte auf diese Dienste und deren Nutzungsdauer werden mittels lokaler und globaler Einstellungen selbst durch die Ressourcenbesitzer (*Resource Provider*) festgelegt. Andere Personen oder Organisationen, die nicht die nötige Infrastruktur für bestimmte Operationen besitzen, können dann die in der VO verfügbaren Dienste verwenden. Bevor aber sie die in der VO bereitgestellten Dienste und Ressourcen überhaupt nutzen können, müssen sie über eine Mitgliedschaft in der betreffenden VO verfügen. Das Beantragen, das Erhalten und die Beendigung einer Mitgliedschaft finden in einer automatischen und dynamischen Art statt.

Aus diesen Betrachtungen ergibt sich ein erheblicher Fragenkomplex, der das Monitoring des Lebenszyklus einer VO betrifft (*das VO-Monitoring Problem*). Wir sind daran interessiert, alle Aktivitäten in einer VO zeitnah verfolgen zu können. Für eine gegebene VO wollen wir also wissen:

- Wer sind die Mitglieder dieser VO? Welche Rollen oder Zugriffsrechte besitzen sie?
- Wie ist die VO strukturiert?
- Welche Ressourcen und Dienste sind in dieser VO verfügbar? Wie lange sind sie verfügbar?
- Welche Aktivitäten finden in dieser VO statt? Welche Mitglieder haben diese Aktivitäten gestartet und welche Ressourcen kommen dabei ins Spiel?

Es gibt eine Vielzahl von Softwarelösungen, die den Aufbau und die Verwaltung von VOs und Grids unterstützen. So bietet das Globus Toolkit [GT4] beispielsweise eine Suite von Tools für den Aufbau, die Konfiguration und das Management von Grid-Infrastrukturen. Es ist eine Open-Source-Software, die von der Globus Alliance[1] eingeführt wurde und weiterentwickelt wird. Zudem besteht es aus Komponenten für Sicherheits-, Ressourcen- und Datenmanagement. Von großem Interesse in dieser Arbeit ist die Ressourcenmanagement Komponente (Monitoring and Discovery System (MDS) - siehe Kapitel 2.5 und 4.2), welche das Entdecken von Ressourcen in VOs und das Monitoring deren Lebenszyklen ermöglicht.
Andere Tools wie CAS (Community Authorization Service) [CAS] oder VOMS (Virtual Organization Membership Service) [VOMS], werden für das Management von VOs und VO-Mitgliedern verwendet. Sie bieten unter anderem Operationen für das Anlegen und das Beenden von VO-Mitgliedschaften, für die Erfassung des Mitglieder-Status bzw. Lebenszyklus oder für die Strukturierung einer VO (z.B. Bildung von Gruppen, Untergruppen oder Sub-VOs).

Diese Tools unterstützen wichtige Aspekte des VO-Managements, lösen aber das oben genannte VO-Monitoring Problem nicht oder bestenfalls teilweise.
VOMS, das im Rahmen des DataGrid[2]-Projekts von CERN[3] und des DataTAG-Projekts[4] von INFN[5] entwickelt wurde, legt beispielsweise für VO-Mitglieder Zugriffsrechte in einer VO fest. Es speichert Informationen über VO-Mitgliedschaften und VO-Mitglieder in einer zentralen Datenbank. Somit ist es möglich, durch die gespeicherten Informationen zu erfahren, welche Mitglieder in einer VO vorhanden sind und was deren Status ist. Andere wichtigen Informationen, wie die in der VO verfügbaren Ressourcen und deren Status werden von VOMS jedoch nicht zur Verfügung gestellt.

Vor diesem Hintergrund ist das **Ziel dieser Arbeit** für das eben vorgestellte VO-Monitoring-Problem, ein Lösungskonzept zu entwickeln. Erstens sollen anhand Beispielszenarien die Anforderungen an die Monitoringlösung erstellt werden. Diese Anforderungen dienen auch der Analyse und Bewertung bestehender Ansätze. Das in dieser Arbeit erstellte Konzept wird vor allem die Funktionalität existierender Tools aggregieren. Es soll unter anderem den dynamischen Aspekt von VOs in Betracht ziehen. Das heißt, trotz der starken dynamischen Struktur einer VO soll es durch das zu erstellende Monitoringkonzept möglich sein, genaue und aktuelle Informationen über alle VO-Elemente zu erfassen. Das resultierende Konzept soll dann prototypisch implementiert und bewertet werden.

1.2 Vorgehensweise der Arbeit

In diesem Abschnitt soll die Vorgehensweise der Arbeit erläutert werden. Sie besteht hauptsächlich in der Analyse der bestehenden VO-Monitoring-Ansätze, dem Entwurf eines neuen Konzepts und der Realisierung dieses Konzepts (siehe Abbildung 1.1).

- **Analyse**
  In der Analysephase werden zunächst die Grundlagen des Grid Computing eingeführt. Dabei werden die wichtigsten Begriffe wie Grid, VO, Grid Architektur, Grid Middleware definiert. Darauf aufbauend werden ein Beispielszenario und ein Testbett beschrieben. Das Beispielszenario soll das verfolgte Ziel dieser Arbeit beispielhaft beschreiben und die Erstellung eines Anforderungskatalogs unterstützen. Das Testbett beschreibt eine Installation, in der das zu entwickelnde Konzept implementiert und bewertet wird. Vorhandene Ansätze für das VO-Monitoring werden dann vorgestellt und anhand des erstellten Kriterienkatalogs analysiert und bewertet. Diese Bewertung wird dann anschließend als Motivation für die Entwicklung eines neuen VO-Monitoringkonzepts dienen.
- **Entwurf**
  In der Entwurfsphase wird das neue VO-Monitoringkonzept vorgestellt. Das neue Konzept besteht grundsätzlich in der Integration vorhandener Ansätze. Es wird erst gezeigt, warum eine Integration

[1] Die Globus Alliance (`http://www.globus.org/alliance/`) ist eine internationale Kollaboration, die die Forschung und die Entwicklung von innovativen Grid-Technologien betreibt.
[2] `http://eu-datagrid.web.cern.ch/eu-datagrid/`
[3] `http://www.cern.de/`
[4] `http://datatag.web.cern.ch/datatag/`
[5] `http://grid.infn.it/`

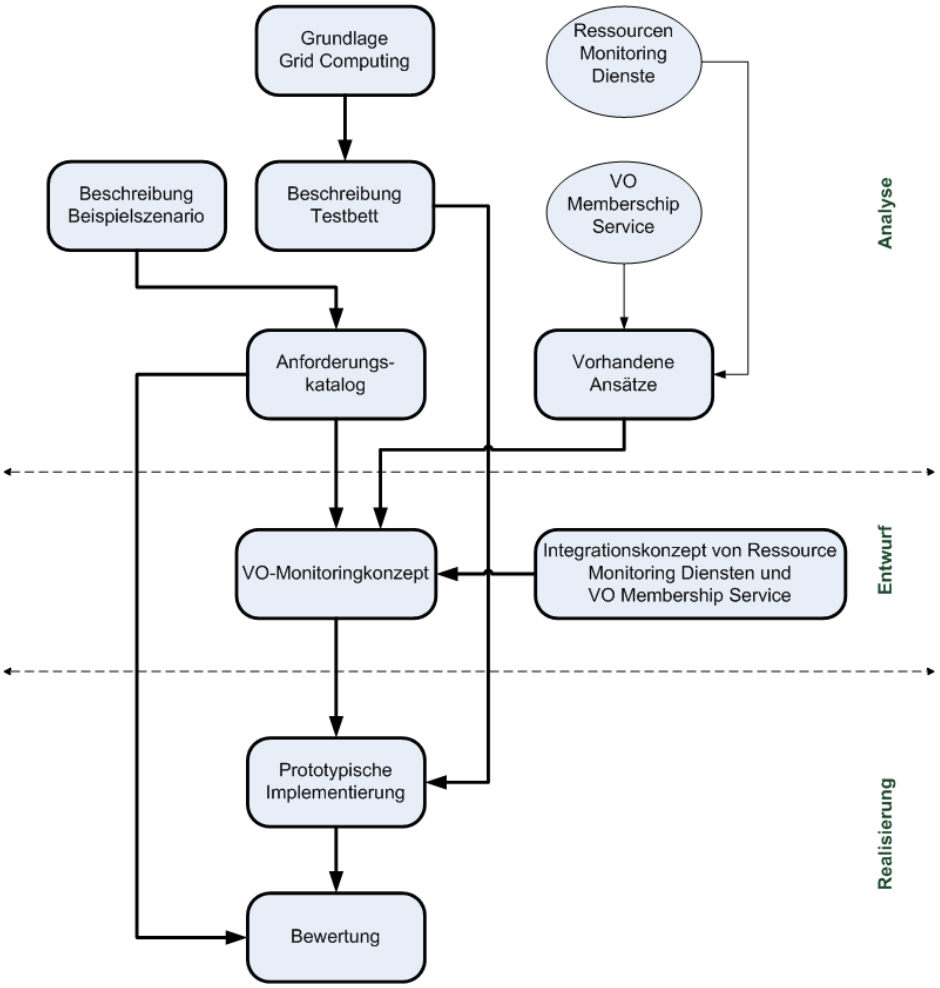


Abbildung 1.1: Vorgehensweise

notwendig ist und wie sie stattfindet. Basierend auf dieser Integration wird eine Architektur für das VO-Monitoring vorgestellt.

- **Realisierung**
  In der Realisierungsphase wird das neu entwickelte Konzept prototypisch im vorgestellten Testbett implementiert, und anschließend anhand des in der ersten Phase erstellten Anforderungskatalogs bewertet.

1.3 Gliederung der Arbeit

Die Arbeit gliedert sich in acht Kapitel.
Im ersten Kapitel wurde die Aufgabenstellung vorgestellt und die Vorgehensweise der Arbeit entwickelt.

Das zweite Kapitel befasst sich mit den erforderlichen Grundlagen. Dazu zählt eine Einführung in Grid Computing. Es soll einen Überblick über das Grid Computing und die Definitionen wichtiger Begriffe und Konzepte vermitteln. Neben der Vorstellung einer Grid-Architektur und der Beschreibung des VO-Begriffs, werden aktuelle Grid Software wie Globus Toolkit [GT4] und VOMS [VOMS] vorgestellt.

Das dritte Kapitel stellt ein Beispielszenario vor, das die Notwendigkeit eines umfassenden Monitoringkonzepts demonstriert. Aus diesem Beispielszenario wird ein entsprechender Anforderungskatalog erstellt. Zudem wird ein Testbett beschrieben, in dem das Konzept implementiert und bewertet werden soll.

Im vierten Kapitel werden bestehende Lösungsansätze vorgestellt und anhand des Anforderungskatalogs bewertet. Es wird sich dabei zeigen, dass diese Ansätze die gestellten Anforderungen nicht vollständig erfüllen.

Das fünfte Kapitel befasst sich mit der Entwicklung eines neuen VO-Monitoringkonzepts. Hier wird zunächst ein Integrationskonzept vorgestellt, das die Potenziale bestehender Tools aggregiert. Diese Integration ist nötig, um das VO-Monitoringkonzept entwickeln zu können.

Das sechste Kapitel stellt die Realisierung des Monitoringkonzepts im Testbett dar, das im Kapitel 3.3 vorgestellt wurde.

Im siebten Kapitel erfolgt eine Bewertung des neu entwickelten Konzepts.

Das abschließende achte Kapitel beinhaltet eine Kurzzusammenfassung der Diplomarbeit und gibt Anregungen für zukünftige Systementwicklungsprojekte/Fortgeschrittenenprojekte und Diplomarbeiten.

2 Grundlagen

In diesem Kapitel wird der Begriff *Grid* eingeführt und erklärt. Dabei werden anhand konkreter Beispiele die wichtigsten Konzepte, die sich hinter diesem Begriff verbergen, und eine Grid Architektur vorgestellt. Einer der wichtigsten Begriffe im Grid Computing ist der der virtuellen Organisationen (VOs). Sie bilden in Grids ein fundamentales Konzept, um organisationsübergreifend komplexe IT-Lösungen bereitzustellen. Sie ermöglichen Gruppen von (realen) Organisationen, Teilorganisationen und/oder Individuen die kontrollierte Verwendung von Ressourcen oder Diensten, um ein gemeinsames Ziel auf der Geschäftsgrundlage einer kooperativen Zusammenarbeit für die Dauer der Existenz der VO zu erreichen [Schi 07]. Da VOs in dieser Arbeit von erheblicher Bedeutung sind, wird diesem Begriff ein eigener Abschnitt gewidmet.
Nach einer kurzen Einführung werden die Tools vorgestellt, die für die Implementierung des in dieser Arbeit entwickelten Konzepts eingesetzt werden:

- das *Globus Toolkit (GT)* ist eine Grid Middleware und
- der *Virtual Organization Membership Service (VOMS)* wird für die Verwaltung von Mitgliedschaften in einer VO eingesetzt.

2.1 Einführung in Grid Computing

Der Begriff *Grid* hat seinen Ursprung in der Analogie zum Stromnetz (engl. *Power Grid*). Ein Stromnetz steht in ursprünglicher Form für ein Netzwerk von elektrischen Stromleitungen. Genauso wie Strom über Stromleitungen und durch eine Steckdose zur Verfügung gestellt wird, soll das Grid einem Benutzer Ressourcen wie z.B. Speicherkapazität oder Rechenleistung über das Internet zur Verfügung stellen.
Das Grid ist eine neuartige IT-Infrastruktur, die eine dezentralisierte Zusammenarbeit zwischen Personen oder Institutionen im Businessbereich, in der Wissenschaft, im Ingenieurwesen oder in anderen Bereichen ermöglicht. Grid Computing fasst alle Rechenverfahren und Protokolle zusammen, die einen direkten und kontrollierten Zugriff zu Computern, Software, Daten und anderen Ressourcen in einer heterogenen Umgebung gewährleisten. Die motivierende Zielsetzung für die Entwicklung der Grid-Technologie war die Möglichkeit von einer gemeinsamen und koordinierten Nutzung von Ressourcen für die Lösung gemeinsamer Problemen innerhalb dynamischer, institutionübergreifender, virtueller Organisationen (VOs). Das bedeutet beispielsweise folgendes: nach der Festlegung von Abrechnungsdaten und Rechten soll ein direkter Zugang zu Rechenleistungen, Anwendungen, Daten oder Instrumenten gemeinschaftlich ermöglicht werden. Eine VO ist in diesem Zusammenhang ein Konsortium verschiedener Personen oder Institutionen, die Informationen austauschen, kollaborieren und ihre Grid-Ressourcen und Dienste derart freigeben, dass ein gemeinsames Ziel verfolgt wird oder neue komplexere Ressourcen und Dienste zusammengestellt werden. VOs erfüllen in der Regel wohl definierte Eigenschaften (*VO-Eigenschaften*) (siehe Tabelle 2.1) [Grid 2] , die detailliert in Abschnitt 2.2 erläutert werden.

Das wachsende Interesse an Grid Technologien und Applikationen in der Forschung und der Industrie hat zur Bildung von Grid Standardisierungsgremien geführt. Das Open Grid Forum (OGF)[1] ist ein Gremium von Benutzern, Entwicklern, Unternehmen und Universitäten, die sich mit der Entwicklung neuer Grid Standards wie die *Open Grid Service Architecture (OGSA)* [OGSAv1.5] beschäftigt. OGSA ist eine Entwicklung in Richtung Grid Architektur, die auf die Web Services Technologie basiert. Die Globus Alliance ist eine Organisation, die die Entwicklung des Globus Toolkits vorantreibt, dem de facto Standard einer Grid Middleware, das das OGSA Framework implementiert. Mehr über Globus Toolkit ist im Abschnitt 2.4 zu finden.

[1] Das Global Grid Forum (GGF) und die Enterprise Grid Alliance (EGA) haben sich im Open Grid Forum (OGF) zusammengeschlossen und wollen künftig gemeinsam für die weitere Verbreitung von Grid-Technik sorgen. `http://www.ogf.org/`

Grid Computing findet in verschiedenen Bereichen Anwendung, insbesondere in e-Science[2] und e-Business[3] Projekten. Beispielweise soll die D-Grid Initiative[4] dabei helfen, Methoden des e-Science zu etablieren, indem sie Projekten aus verschiedenen wissenschaftlichen Gebieten (u.a. Astronomie, Hochenergiephysik, Meteorologie, Medizin, Ingenieurwissenschaften und Geisteswissenschaften) eine Grid-Infrastruktur mit den benötigten Diensten und Benutzer-Support zur Verfügung stellt. Im Folgenden sind einige konkrete Projektsbeispiele aufgelistet, die auf einer Grid Infrastruktur aufbauen:

- **AstroGrid-D**, das German Astronomy Community Grid (GACG)[5] wird im Rahmen der D-Grid-Initiative vom Bundesministerium für Bildung und Forschung (BMBF)[6] gefördert. Ziel des AstroGrid-D ist die Einbindung der astronomischen Forschungsinstitute in Deutschland in eine einheitliche Grid-basierte Infrastruktur, um verteiltes, kollaboratives Arbeiten zu fördern. Existierende Hard- und Softwareressourcen, u.a. astronomische Datenarchive und robotische Teleskope in den beteiligten Forschungsinstituten, sollen in die Infrastruktur integriert werden.
- Ziel des **TextGrid**[7], das auch auf der D-Grid Infrastruktur aufbaut, ist ein Community-Grid für die Geisteswissenschaften zu etablieren. TextGrid errichtet eine Grid-fähige Workbench für die gemeinschaftliche philologische Bearbeitung, Analyse, Annotation, Edition und Publikation von Textdaten für die Philologie, Linguistik und angrenzende Wissenschaften.
- Das **Earth System Grid (ESG)**[8] ist ein vom U.S. Department of Energy unterstütztes Projekt zur Modellierung und Simulation des Ökosystems der Erde. Durch eine Kombination von Grid Technologien und der gemeinschaftlichen Nutzung von Superrechnern wird eine mächtige Umgebung bereitgestellt, die die nächste Generation der Klimaforschung ermöglicht.

Wie am Anfang dieses Abschnittes bereits erläutert, erfüllen VOs wohl definierte Eigenschaften. Beispielsweise können VOs stark nach Größe, Ziel, Struktur, Soziologie, Lebenzyklus, Breite und Gemeinschaften variieren [Schi 07]. Das heißt, die Ressourcen und Mitglieder einer VO können dynamisch kommen und gehen.
Im Folgenden werden kurz die Vorteile des Grid Computing gegenüber den aktuellen Technologien erläutert.
Aktuelle Internet-Technologien ermöglichen zwar die Kommunikation und den Informationsaustausch zwischen Rechnern, sie verfügen aber nicht über integrierte Ansätze zur koordinierten Nutzung verteilter Ressourcen für gemeinsame Rechen- und Datenhaltungszwecke.
Heutige Internet-Browser unterschützen zwar Authentifizierung über TLS (*Transport Layer Security*) [tls 00]. Aber sie unterschützen kein Single Sign-On [SSO] und keine Delegation. Andere Beispiele sind Application und Storage Service Providers. Sie benutzen meistens VPN-Technologien (*Virtual Private Network*) [Lipp 06] für die Sicherheit und die Bildung von virtuellen privaten Netzwerken und bieten ihre Dienste auf der Basis von HTTP, FTP, etc. Kunden müssen gewöhnlich SLAs (*Service Level Agreement*) [SLA] abschließen, um die Dienste des Providers zu nutzen. Diese Vorgehensweise ist sehr statisch, sie unterstützt nicht die Dynamik, die bei VOs erforderlich ist. Ein VPN ist daher keine VO, da es kein dynamisches Kommen und Gehen von Rechnern im Netzwerk unterstützt. Eine manuelle und aufwändige Konfiguration ist immer notwendig.
Alle diese Beispiele deuten darauf hin, dass das Grid kein Ersatz der heutigen Technologien ist, sondern ein Zusatz für die Entwicklung von dynamischen, flexibleren und leistungsstärkeren verteilten Anwendungen darstellt.

2.2 Virtuelle Organisation

Der Begriff *virtuelle Organisation (VO)* wurde bereits oben intuitiv eingeführt. Im Folgenden werden anhand eines Beispiels die Eigenschaften virtueller Organisationen erläutert. Allgemein definiert eine VO eine Form der Organisation, bei der sich rechtlich unabhängige Unternehmungen und/oder auch Einzelpersonen virtuell

[2] Unter dem Schlagwort e-Science wird ein Bereich der Wissenschaft - die enhanced Science (zu deutsch etwa erweiterte Wissenschaft) - verstanden.
[3] E-Business ist die integrierte Ausfuhrung aller automatisierbaren Geschäftsprozesse eines Unternehmens mit Hilfe von Informations- und Kommunikationstechnologie.
[4] http://www.d-grid.de/
[5] http://www.gac-grid.org/
[6] http://www.bmbf.de/
[7] http://www.textgrid.de/
[8] www.earthsystemgrid.org/

(meist über das Internet) für einen gewissen Zeitraum zu einem gemeinsamen Verbund zusammenschließen. Im Grid-Kontext ist eine VO ein permanentes oder zeitlich begrenztes Konsortium geographisch verteilter Individuen, Gruppen, Organisationeinheiten oder ganzer Organisationen, die Teile ihrer physischen oder logischen Ressourcen und Dienste derart koordinieren, dass ein gemeinsames Ziel erreicht wird. [Schi 07]

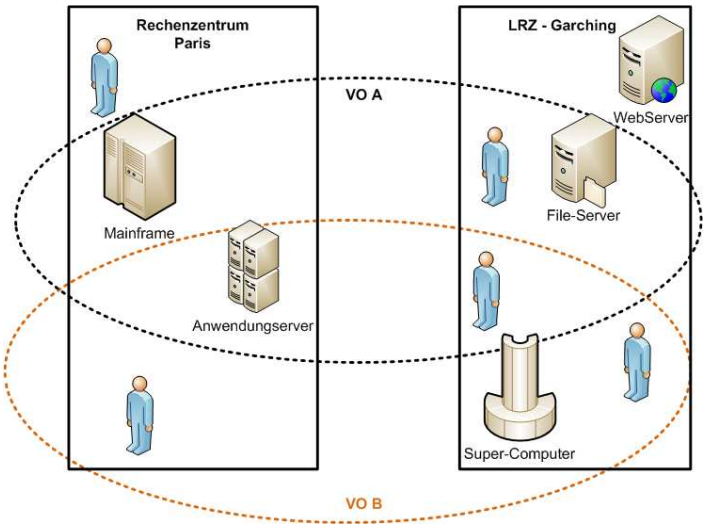


Abbildung 2.1: VO-Beispiele

In der Abbildung 2.1 sind zwei VOs (VO-A und VO-B) zu sehen.

- Die VO-A stellt Dienste zur Speicherung großer Datenmenge und Dienste für schnellen und zuverlässigen Datentransfer zur Verfügung.
- Die VO-B stellt Dienste für die Simulationen von Klimaänderungen zur Verfügung.

Die Mitglieder dieser VOs (*VO-Mitglieder*) haben Zugriff auf bestimmte Ressourcen, die von realen Organisationen bereitgestellt wurden. Diese realen Organisationen können selbst VO-Mitglieder sein. Unter einer realen Organisation verstehen wir die realen Institutionen oder Personen, die Grid-Ressourcen besitzen und sie in virtuellen Organisationen bereitstellen (z.B. Rechenzentrum-Paris und LRZ-Garching in Abbildung 2.1). Diese Bereitstellung wird gewöhnlich durch Zugriffszeiten und Zugriffsrechte (*Constraints*) eingeschränkt. Beispielsweise kann in der Abbildung 2.1 das LRZ die Zugriffszeit der Ressource „*File-Server*“ derart festlegen, dass sie nur eine bestimmte Zeit in der VO-A zugreifbar ist und danach weiter in einer anderen VO (VO-C zum Beispiel). Ebenso können die Zugriffsrechte so festgelegt werden, dass nur bestimmte Mitglieder innerhalb einer VO eine Ressource für eine beschränkte Zeit nutzen dürfen. Die Mitgliedschaft (*Membership*) zu einer VO erfolgt gewöhnlich durch ein Agreement. Dafür gibt es in jeder VO einen Verantwortlichen (meistens ein VO-Administrator) [Schi 07], der die Rechte von jedem Mitglied verwaltet. Die Zugriffsregeln auf die Ressourcen werden selbst von den Besitzern (*Resource Provider*) bestimmt.
Wegen der Dynamik der VOs sind diese Zugriffsregeln auf Grid-Ressourcen oft nicht für einzelne Benutzer festgelegt. Sie werden vielmehr durch implizite Regeln festgelegt, z.B. ein Benutzer kann eine Ressource nutzen, wenn er sich als Administrator oder Student ausweisen kann. Dafür müssen Mechanismen implementiert werden, die genau diese Identität der VO-Teilnehmer prüfen können.

Um eine solche Nutzungsrelation zwischen den VO-Mitgliedern und den Grid-Ressourcen zu realisieren, werden Verfahren benötigt, die zu einer bestimmten Zeit Auskunft darüber geben können, welche Benutzer und Ressourcen sich innerhalb einer VO befinden, und welche Beziehungen zwischen allen diesen Elementen bestehen. So muss zum Beispiel ein neuer VO-Teilnehmer in der Lage sein, herauszufinden, welche Ressourcen er nutzen darf, was die Eigenschaften dieser Ressourcen sind und welche Zugriffsregeln (*Policies*) den Zugriff auf diese Ressource steuern.
Nachdem ein Benutzer innerhalb einer VO einen Job[9] auf einem Rechner gestartet hat, kann es in manchen Situationen auftreten, dass dieser Job wiederum einen anderen Job auf einer entfernten Maschine starten muss,

[9] Ein Job bezeichnet hier eine Aufgabe oder eine Programmausführung.

was die Delegierbarkeit von Zugriffsrechten voraussetzt. Dabei sind auch Mechanismen für die Koordination und Monitoring von Jobs auf mehrere verteilte Ressourcen zu berücksichtigen. Es soll zudem die Fähigkeit zur Verfügung gestellt werden, dass eine einzelne Ressource gleichzeitig für mehrere verschiedene Zwecke verwendet werden kann. Beispielweise kann ein einziger Rechner in einer VO (VO1) nur ein Teil einer Software bereitstellen, während die komplette Software in einer anderen VO (VO2) verwendet werden darf. Die Tabelle 2.1 fasst alle diese Mechanismen und Dienste zusammen:

Autorisierung
Authentifizierung
Delegation
Ressourcemanagement
Datamanagement
VO-Administration
Execution Management
Job- und Ressource-Monitoring

Tabelle 2.1: VO-Eigenschaften

Um alle diese VO-Eigenschaften zu erfüllen, hat sich eine Grid Architektur etabliert. Diese geschichtete Architektur besteht aus Protokollen, die Interoperabilität garantieren und Dienste für Ressourcemanagement, Sicherheit, Authentifizierung, Autorisierung, Delegation, Monitoring, etc. definieren. Im folgenden Abschnitt wird kurz auf diese Architektur eingegangen und anschließend wird die *Open Grid Service Architectur - (OGSA)* [OGSAv1.5] vorgestellt.

2.3 Grid Architektur

In der Abbildung 2.2 ist eine geschichtete Grid Architektur dargestellt, die nicht die Implementierungsdetails betrachtet und ebenso wenige Restriktionen an den Aufbau und die Implementierung festlegt. Die Komponenten der Grid Architektur liefern die Basis zur Erfüllung der VO-Eigenschaften, die oben dargestellt wurden. Die Architektur besteht aus fünf Schichten (*Layers*), die jeweils einen Satz von Protokollen definieren. Im Folgenden wird jede Schicht kurz beschrieben.

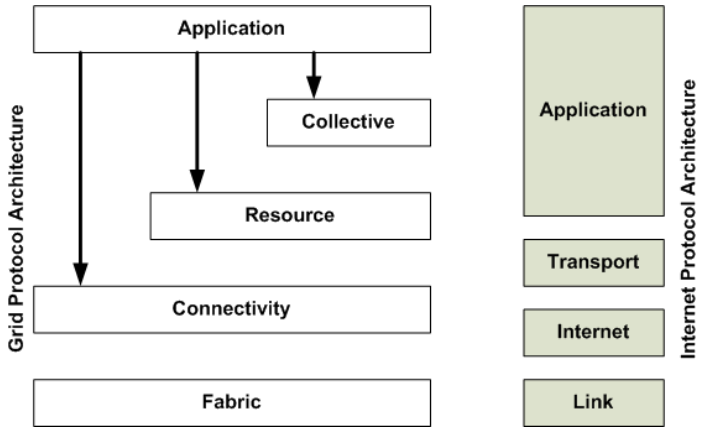


Abbildung 2.2: Die geschichtete Grid Architektur und ihre Beziehung zur Internet Protokoll Architektur. Da sich die Internet Protokoll Architektur von der Netzwerkschicht bis zur Applikationschicht ausstreckt, ist eine Abbildung der Grid Schichten zu Internet-Schichten möglich. [ANAT]

- **Fabric Layer** (Beispiele: *Portable Batch System - PBS* [PBS]; *General-purpose Architecture for Reservation and Allocation - GARA* [GARA])
  In dieser Schicht befinden sich alle physischen und logischen Ressourcen (Recheneinheiten, Speichereinheiten, Netzwerk Ressourcen, Computercluster, Distributed File System, Code Repositories, Cata-

logs , etc.), die durch Grid-Protokolle im Grid bereitgestellt werden. Hier werden alle lokalen und Ressourcen-spezifischen Operationen implementiert, die dann von Komponenten höherer Ebene genutzt werden können, um auf die Ressourcen zuzugreifen. Gewöhnlich wird nicht die gesamte Funktionalität der verfügbaren Ressourcen implementiert, sondern nur eine Teilmenge davon. Es können auch zwei oder mehrere lokale Funktionalitäten zu einer Einzigen aggregiert werden. Diese Schicht implementiert außerdem Mechanismen für Introspektion (Mechanismen für die Entdeckung von Ressourceneigenschaften und Zuständen wie Hardwareeigenschaft, Softwareeigenschaft, CPU, Bandbreite, etc.) und Ressourcenmanagement.

- **Connectivity Layer** (Beispiel: *Grid Security Infrastructure- GSI* [GT4])
  Diese Schicht definiert Kommunikations- und Authentifizierungsprotokolle für Grid Operationen. Kommunikationsprotokolle definieren Mechanismen für Transport, Routing und Naming. Authentifizierungsprotokolle definieren Mechanismen, die sichere Grid-Transaktionen garantieren, basierend auf Identitätsprüfungen der Grid-Ressourcen und Benutzer. Diese Verfahren sollen außerdem das Single Sign-On, die Delegation, die Autorisierung und lokale Sicherheitskonfigurationen (z.B. wie bei Unix) unterstützen.

- **Resource Layer** (Beispiel: *Grid Resource Access and Management - GRAM, GridFTP* [GT4])
  In dieser Schicht werden hauptsächlich Informationsprotokolle und Managementprotokolle definiert. Informationsprotokolle werden angewendet, um Informationen über die Struktur und den Zustand einer Ressource zu erhalten. Managementprotokolle definieren Mechanismen für den Zugriff und das Starten von Programmen (Jobs) auf entfernten Ressourcen. Die Zugriffseinschränkung und die Prüfung der Identität des Aufrufers werden durch den darunter liegenden Connectivity Layer übernommen.

- **Collective Layer** (Beispiel: *Monitoring and Discovery System - MDS* [GT4])
  Diese Schicht definiert Mechanismen zum Monitoring und Finden von Grid-Ressourcen. Außerdem können auch VO-spezifische Dienste implementiert werden wie Directory Services, Data Replication Service, Collaborating Services, Brokering Services.

- **Application Layer**
  In dieser Schicht befinden sich alle Benutzer-Applikationen, die innerhalb einer VO operieren. Die Pfeile auf der Abbildung 2.2 deuten darauf hin, dass Programme des Application Layers unmittelbar Komponente des Connectivity, Resource oder Collective Layers ansprechen können.

Die OGSA, die eine dienstorientierte Grid Architektur spezifiziert, wird im Folgenden vorgestellt.

Seit den ersten Spezifikationen der Grid-Architektur, wurden diverse Implementierungsprojekte (insbesondere das Globus Toolkit) gestartet. Das Globus Toolkit [GT4], eine Open Source Middelware zum Aufbau von Grid, hat enorm zur Weiterentwicklung von Grid-Architekturen beigetragen. So wurden beispielsweise erste Versuche für den Aufbau einer Grid Middelware nach OGSA-Standards mit dem Globus Toolkit Version 3 (GT3) durchgeführt. OGSA ermöglicht die Kreation, die Verwaltung und die Verwendung von einer Menge von Diensten, die eine Schnittstelle zu den physischen Ressourcen darstellen. Mit der Einführung von OGSA als Standard Grid-Architektur wurden folgende Ziele verfolgt:

- die Festlegung von Standard Protokollen in der Grid-Infrastruktur
- der Einsatz einer Service Oriented Architecture (SOA) [SOA1]. Grid-Ressourcen werden als Dienste (*Grid Services*) dargestellt.

Die Architektur in der Abbildung 2.2 beschreibt Schichten mit verschiedenen Komponenten. Alle Komponenten bieten Dienste meistens mit proprietären Protokollen und nutzen selbst Dienste anderer Schichten. So kann beispielsweise in der Fabric-Schicht ein Dienst für Dateizugriff und -speicherung zur Verfügung gestellt werden, der durch einen Samba File-Server eingerichtet ist. Ein Client muss eventuell zusätzlich eine Software installieren, um diesen Dienst nutzen zu können (falls er das verwendete Protokoll nicht unterstützt).
Eines der wichtigsten Ziele von OGSA ist diese Lücke zu schließen. Das geschieht durch die Definition von Standard-Schnittstellen für die Virtualisierung von Grid-Diensten. Unter Virtualisierung wird im File-Server-Beispiel verstanden, dass statt direkt eine Dienstoperation aufzurufen, eine Standard-Schnittstelle definiert wird, die dann die eigentlichen (physischen) Operationen implementiert. Auf diese Art braucht der Client sich nicht um die Implementierungsdetails zu kümmern. Er braucht nicht mehr zu wissen, welche (proprietären) Protokolle für den Dienst eingesetzt werden. Er benötigt *nur* die Definition einer Standard-Schnittstelle. Diese

Virtualisierung von Diensten (*service virtualization*) wird durch eine Dienst-orientierte Architektur (SOA) sichergestellt. Dafür können z.B. *Web Services* eingesetzt werden. Ein Web Service ist ein Softwaresystem, das eine Nachrichten-orientierte Kommunikation zwischen Softwarekomponenten über das Internet ermöglicht. Er definiert XML-basierte Mechanismen und Protokolle für den Zugriff und die Entdeckung von Softwarekomponenten im Netz. Vorteil dabei ist, dass die Kommunikation durch offene Standards stattfindet. Für die Beschreibung der Dienstschnittstelle wird *WSDL (Web Service Description Language)* [WSDLv1.1] eingesetzt. *SOAP (Simple Object access protocol)* [SOAPv1.2] oder *XML-RPC* [xml 02] wird zur Kommunikation und *UDDI (Universal Description, Discovery and Integration)* [UDDI] als Verzeichnisdienst (*Directory Service*) zur Registrierung und zum dynamischen Finden von Web Services eingesetzt. Die vollständige Definition von Web Services ist nicht Bestandteil dieser Arbeit und kann in der entsprechenden Literatur nachgelesen werden [SOA1].

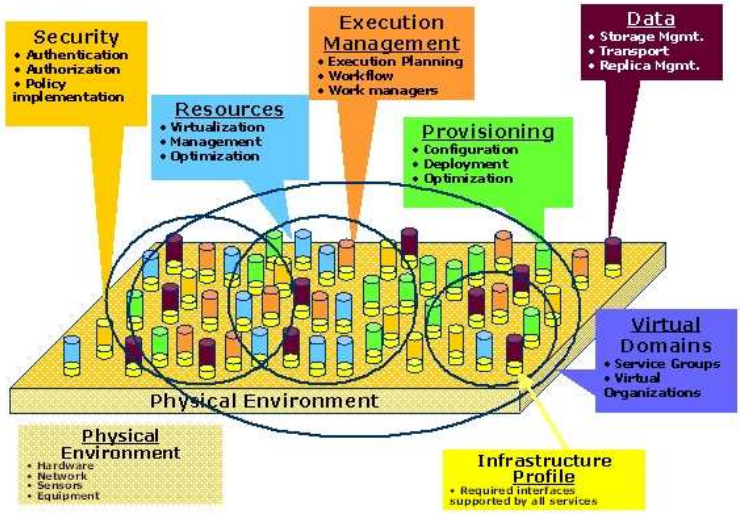


Abbildung 2.3: OGSA Framework [OGSAv1.5]

Die OGSA-Dienste können in die geschichtete Grid Architektur der Abbildung 2.2 wie folgt eingegliedert werden: In der Connectivity-Schicht haben wir eine kleine Menge von Diensten für die Authentifizierung, die Zuordnung vom Berechtigungsnachweis, die Autorisierung, die Verifizierung von Zugriffsrechten etc. Auf der Ressource-Schicht gibt es dann Dienste für den Datenzugriff, das Starten von Programmen, die Allokation von Bandbreite, etc. Manche Dienste der Ressource-Schicht können wiederum in der Collective-Schicht verwendet werden, wenn sie spezialisiert werden sollen. Die Abbildung 2.3 stellt die Hauptkomponente der Open Grid Architecture dar. Die vollständige Liste der Komponente ist in [OGSAv1.5] zu finden:

- **OGSA Dienste**
  OGSA Dienste sind in Abbildung 2.3 durch Zylinder dargestellt. Sie implementieren Standard-Schnittstellen für den Zugriff auf Grid-Ressourcen. Zu den Standard OGSA-Diensten gehören Dienste für Sicherheit-Management, Ressourcenmanagement, Datenmanagement und Execution Management. Eine wichtige Motivation von OGSA ist das Komposition-Paradigma, wobei die Funktionalitäten mehrerer Ressourcen oder Dienste zu einem einzigen Dienst oder zu einer kleinen Dienstmenge aggregiert werden können.
- **virtual Domains**
  Sie repräsentieren virtuelle Organisationen (In Abbildung 2.3 durch Ellipsen dargestellt).
- **Infrastructure Profile**
  Sie sind Schnittstellen, die eine Interaktion zwischen OGSA-Diensten und den physichen Ressourcen ermöglichen.
- **Physical Environment**
  Hier befinden sich alle physichen Ressourcen wie Hardware, Sensoren, Netzwerk-Infrastrukturen, etc.

Bisher wurden die Hauptkonzepte des Grid Computing vorgestellt, die für das Verständnis dieser Arbeit notwendig sind. Für die Implementierung des in dieser Arbeit entwickelten Konzepts sind noch die folgenden Komponenten (Globus Toolkit, VOMS) von Bedeutung.

2.4 Globus Toolkit

In den Abschnitten oben wurde das Globus Toolkit (GT) mehrmals erwähnt, als eine Menge von Software Komponenten zum Aufbau von Grid-Anwendungen. Das Globus Toolkit gilt derzeit als de facto Standard einer Grid Middleware, die in der Version Globus Toolkit 4 (GT4) weitgehend OGSA-kompatibel ist.

Die ersten Versionen von Globus Toolkit wurden auf keiner formalen technischen Spezifikation aufgebaut. Die verwendeten Techniken waren proprietär und folgten keiner Standard Spezifikation. Ziel des Globus Toolkit war damals, Tools für die Entwicklung von SOA-basierten verteilten Anwendungen bereitzustellen. Mit der Versionreihe 3.x wurde der Standardisierungsprozess durch die Einführung von OGSI [OGSI] gestartet. GT4 (April 2005) verwendete Web Services Standards und folgte auch im Vergleich zu vorherigen Versionen vollständig einer dienstorientierten Architektur (OGSA). Zudem bot GT4 mehr Funktionalität, Robustheit, Performanz und die Einhaltung von Standards. Die aktuelle stabile Version ist GT 4.0.5 und kann auf der Webseite (`http://www.globus.org/toolkit/`) heruntergeladen werden.

Abbildung 2.4 gibt einen Überblick über die einzelnen Komponenten:

- Die *WS Components* sind Komponenten, die durch Web Services implementiert werden und
- die *Non-WS Components* dementsprechend Komponenten, die nach wie vor proprietär sind.

Im Folgenden werden die wesentlichen Komponenten und ihre Funktionalität beschrieben. Das MDS (Monitoring and Discovery System), das für das Monitoring von VOs wichtig ist, wird später im Kapitel 5 detailliert dargestellt:

- **Security**
  Grid Computing basiert wesentlich auf einem organisationsübergreifenden Sicherheitskonzept. Dementsprechend stellt Globus Toolkit mit seiner *Grid Security Infrastructure (GSI)* ein mächtiges Tool zum Aufbau sicherer Grid-Anwendungen zur Verfügung. In der Abbildung 2.4 sind in der ersten Spalte alle Komponenten der Grid Security Infratructure (GSI) aufgelistet. GSI bietet die notwendigen Grundlagen und Mechanismen für sichere Kommunikation, Authentifizierung, Delegation und Autorisierung. Da GSI auch nach einer dienstorientierten Architektur aufgebaut ist und Web Services verwendet, werden SOAP-Nachrichten für die Kommunikation angewendet. Für die sichere Kommunikation werden zwei Konzepte eingesetzt: *Transport Level Security* [tls 00] (die ganze SOAP-Nachricht wird -wie im Web-HTTPS-Protokoll für den sicheren Zugriff auf Webseiten- kodiert) und *Message Level security* (nicht der gesamte Datenverkehr wird verschlüsselt sondern nur die Inhalte z.B. des SOAP Body. Web Service Standards wie WS-Security [WS-S] und WS-SecureConversation [WS-SC] werden verwendet, um diese Funktionalität zu gewährleisten). In einer sicheren Grid-Umgebung müssen alle Ressourcen, Dienste, Personen oder Institutionen eine Identität besitzen. Die Identität wird bei der Authentifizierung benötigt, damit der Zugriff auf bestimmte Dienste nur autorisierten Benutzern erlaubt ist. Zu diesem Zweck werden X.509 Zertifikate verwendet. GSI ermöglicht aber auch die User-Password-, und die anonyme Authentifizierung. In vielen Anwendungsfällen erhalten alle authentifizierten VO-Mitglieder auch direkt die Zugriffsrechte auf die Ressourcen, die in der VO bereitgestellt werden. Aber innerhalb dieser VO kann auch in manchen Fällen der Zugriff auf bestimmte Ressourcen nur für bestimmte Mitglieder einschränkt werden. Dafür werden Autorisierungsverfahren verwendet. Die *Authorization Framework*-Komponente von GSI bietet verschiedene Autorisierungsverfahren wie Gridmap oder SAML [SAML]. Eine weitere wichtige Funktionalität von GSI ist die Delegation von Rechten (*credentials*), die durch X.509 Proxy-Zertifikate [ProxyCert] realisiert wird und z.B. das Single Sign-On ermöglicht. Da Autorisierungsverfahren in dieser Arbeit von großer Bedeutung sind, werden sie in den folgenden Kapiteln noch detaillierter behandelt (Kapitel 2.5).
- **Data Management**
  In der zweiten Spalte der Abbildung 2.4 werden die Dienste für das Datenmanagement dargestellt.

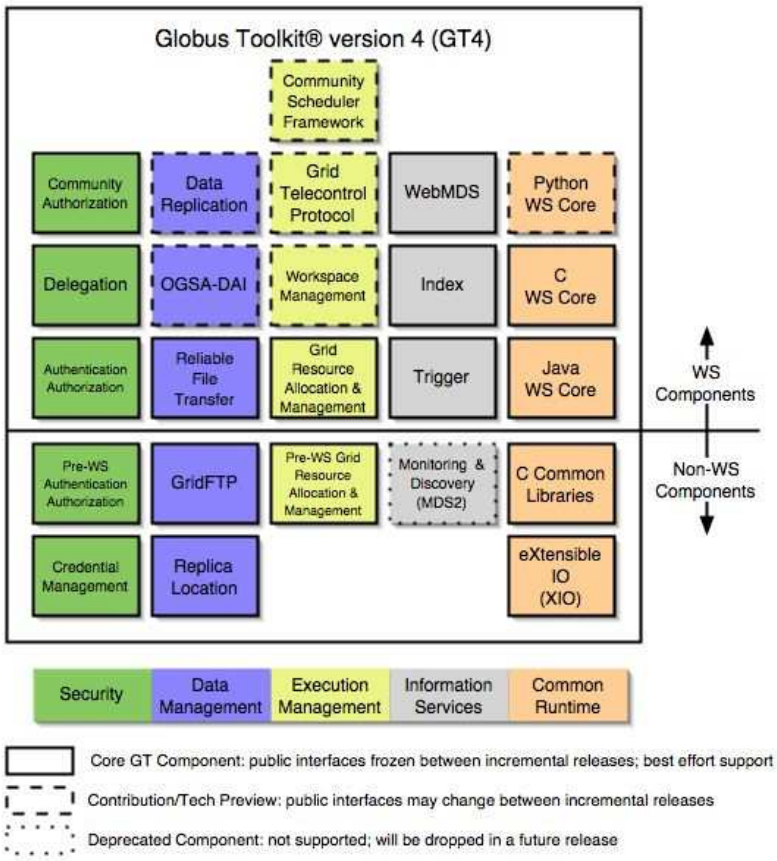


Abbildung 2.4: Bestandteile des Globus Toolkits 4 [GT4]

RLS (Replica Location Service) ist ein Dienst zum Registrieren und Entdecken entfernter Kopien von Datensätzen. Das GridFPT, das keine Web Services Schnittstelle implementiert, ist ein sicheres, zuverlässiges Dateitransfer-Protokoll. Dieses Protokoll basiert auf FTP und unterstützt zusätzlich zu den normalen FTP Operationen auch das Striping[10] und die Übertragung über Dritte (*Third-party Transfer*). Die letzte Komponente ist das *RFT (Reliable File Transfer)*. Es ist ein Web Service, der Standard SOAP-Nachrichten verwendet, um eine oder mehrere GridFTP Übertragungen starten, protokollieren oder verwalten zu können (siehe Kapitel 3.3.1).

- **Execution Management**
  Um die Ausführung von Programmen auf entfernten Rechnern zu ermöglichen, bietet das Globus Toolkit 4 das *GRAM (Grid Resource Allocation and Management)*, wie in der dritten Spalten der Abbildung 2.4 dargestellt. GRAM bezeichnet eigentlich zwei Komponente: *Pre-WS GRAM* und *WS-GRAM*. Beide Komponenten ermöglichen die entfernte Ausführung von Programmen, bieten APIs zum Starten, Monitoring und Beenden von Jobs. WS-GRAM ist die Web Service Implementierung von GRAM, und bietet die gleiche Funktionalität durch Web Services-Operationen. Nachdem ein Job durch WS-GRAM eingereicht ist, werden die Anfragen zum Gatekeeper (Dämonprozess auf der entfernten Maschine, der auf entfernte Job-Anfragen wartet) des entfernten Rechners weitergeleitet. Der Gatekeeper verarbeitet dann die Anfragen und legt einen Job Manager an. Der Job Manager, der nur einmal auf einen Rechner existieren kann, startet lokal das aufgerufene Programm und teilt dem Prozessaufrufer Statusinformationen über des Jobs mit. Auf diese Weise kann der entfernte Nutzer Prozesse starten, terminieren, anhalten und Prozesszustände sehen.
- **Information Services: Monitoring and Discovery System (MDS)**

[10] Striping ist eine Mapping-Technik von Disk-Arrays, bei der Datenblöcke in einem rotierenden System auf zwei oder mehrere Festplatten verteilt werden. So wird beispielsweise der Datenblock **A** auf der Festplatte 1 abgelegt, der Datenblock **B** auf 2, der Datenblock **C** auf 3 usw.

Dienste für Monitoring und Entdeckung von Ressourcen sind vital für alle verteilten Anwendungen. Im Grid-Kontext bietet das MDS Webdienste für das Monitoring und die Entdeckung von Grid-Ressourcen innerhalb einer VO. Somit ist es für jedes autorisierte Mitglied in einer VO möglich, Ressourcen nach bestimmten Kriterien zu suchen und auch deren Status anzusehen. WS MDS (MDS4), die Web Services Implementierung von MDS, besteht in GT4 aus:

- dem *Index Service*, der die Eigenschaften von Grid-Ressourcen sammelt und sie über einen Webdienst publiziert. Diese Ressourceneigenschaften werden als „*Resource Properties*“ bezeichnet und werden nach WSRF [WSRF] und WS-Notification [WSN] implementiert. Der Index Service selbst folgt der WS-ServiceGroup Spezifikation [WS-SGr].
- dem *Trigger Service*, der Daten aus Grid-Ressourcen sammelt und bestimmte Programme ausführt, wenn vordefinierte Ereignisse auftreten.
- dem *WebMDS*, das als Web-Applikation alle MDS-Dienste aggregiert. WebMDS ermöglicht eine benutzerfreundliche Nutzung des MDS durch einen Webbrowser.

In dieser Arbeit werden mittels MDS-Diensten VO-Monitoring Dienste entwickelt. Das MDS und das darunter liegende Framework (*Aggregator Framework*) werden in den Kapiteln 4 und 5 weiter behandelt. Die Abbildung 2.5 stellt die MDS-Architektur in GT4 dar.

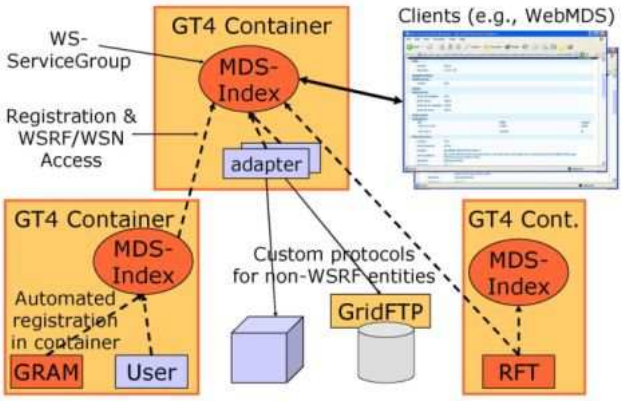


Abbildung 2.5: GT4 Monitoring and Discovery System,[GT4-Paper]

- **Common Runtime**
  Die Runtime-Komponenten bieten Web und Pre-Web Services durch Bibliotheken und Tools für den Aufbau plattformunabhängiger Dienste. Die WS-Komponenten liegen in den Sprachen Java, C und Python vor.

2.5 Management von Mitgliedschaften zu VOs

VOs - wie sie in dieser Arbeit betrachtet werden - haben eine hierarchische Struktur, sie können in Gruppen oder Untergruppen unterteilt werden, wobei die VO selbst als Wurzel der Hierarchie betrachtet werden kann. Somit können Mitglieder einer VO zu einer oder mehreren Gruppen gehören. Innerhalb einer Gruppe können die Mitglieder noch nach weiteren Kriterien (wie die Rolle oder die Fähigkeit (*capability*)) eingeteilt werden. Betrachten wir als Beispiel eine Gruppe „Mitarbeiter“ innerhalb einer beliebigen VO. In dieser Gruppe kann eine weitere Einteilung stattfinden, indem manche Mitarbeiter mit der Rolle „Administrator“ versehen werden. Diese Administratoren, die sich in derselben Gruppe wie anderen Mitarbeiter befinden, erhalten zusätzlich spezielle administrative Zugriffsrechte auf vertrauliche Daten. Alle anderen Mitglieder haben nur Zugriff auf nicht vertrauliche Daten in dieser Gruppe. Diesem Muster folgt auch das Autorisierungsschema für den Zugriff auf Grid-Ressourcen in einer VO. Zur Gewährleistung der Autorisierung von VO-Mitgliedern für Grid-Ressourcen werden zwei Komponenten benötigt:

- ein VO-Management System, das Informationen über die VO-Mitglieder und ihre Beziehung zu der VO speichert und verwaltet. Dieses System gibt Auskunft darüber, welche Gruppen oder Untergruppen in der VO vorhanden sind, welche Mitglieder zu diesen Gruppen gehören und über welche Rollen oder Fähigkeiten sie verfügen.
- eine Zugriffskontrollliste (*Access Control List*) für jede Ressource, die gewöhnlich eine Liste von Benutzern enthält, die diese Ressource verwenden dürfen. Diese Zugriffskontrollliste wird vom Ressourcenbesitzer angelegt und verwaltet. Der Ressourcenbesitzer ist zwar meistens selbst ein VO-Mitglied, muss es aber nicht sein.

Die Struktur der Einträge in der Zugriffskontrollliste wird von dem VO-Management System spezifiziert. Das heißt, wenn das VO-Management System die Identität der einzelnen Mitglieder verwendet, um Zugriffsrechte innerhalb der VO zu vergeben, können dann die Zugriffskontrolllisten beispielsweise Gridmap-Files sein. Eine Gridmap-File ist eine Datei, die aus einer Liste von Distinguished Names (DN) -von X.509-Zertifikaten- besteht. Die *PKI (Public Key Infrastructure)* [pki 02] garantiert, dass ein Benutzer-DN weltweit eindeutig ist. Ein Beispiel eines DN ist */DC=org/DC=mygrid/OU=People/CN=Herve Amikem*. Eine Ressource gehört dann zu einer VO erst wenn mindestens ein Mitglied aus dieser VO in ihrer Zugriffskontrollliste eingetragen ist (siehe Kapitel 5.3.2).

Als VO-Management System wird in dieser Arbeit der VOMS (Virtual Organization Membership Service) eingesetzt. VOMS setzt zur Authentifizierung ein X.509 Zertifikat voraus und speichert alle Informationen über die VO-Mitgliedschaften in einer zentralen Datenbank. VOX (*VOMS eXtension*) [VOX] ist eine Weiterentwicklung von VOMS, die die Speicherung und Verwaltung von zusätzlichen VO-Mitglieder-Informationen, Institutions- und standortspezifische Informationen ermöglicht. VOX bietet mit VOMRS auch einen Dienst für die Registrierung von Benutzern bei VOs. VOMRS (*VO Membership Registration Service*) besteht aus einem Server, der die Registrierung von Benutzern bei VOs und die Koordination des Registrierungsprozesses über verschiedene VOs ermöglicht. Im Folgenden werden die VOMS Architektur und die Integration von VOMS in Globlus Toolkit 4 kurz dargestellt.

Die Abbildung 2.6 zeigt die verschiedenen VOMS-Komponenten. Der VOMS Server bietet zwei Dienste:

- Der *VOMS Admin* Service erlaubt einem VO-Manager, jegliche Änderungen vorzunehmen wie Gruppen definieren, neue Mitglieder mit Rechten und Gruppenzugehörigkeit anlegen, etc.
- Der *VOMS Core Service* erhält Anfragen von Clients und gibt Informationen über das angefragte VO-Mitglied zurück. Clients bestehen aus den Komponenten *voms-proxy-init, voms-admin* und *Web-Browser*. Über den voms-proxy-init wird eine Anfrage zum VOMS Server gesendet, und die Autorisierungsinformationen eines Benutzers (der ein VO-Miglied ist) zurückgegeben. Aus diesen zurückgegebenen Informationen wird dann ein Proxy-Zertifikat generiert, das der Benutzer für die Autorisierung verwenden kann. Der voms-admin und der Web Browser sind zwei verschiedene Implementierungen (Command Line- und Web-Applikation) des Admin Services. Der VOMS Admin Service besitzt eine SOAP-Schnittstelle. Da wir in dieser Arbeit eine SOA-Applikation entwickeln wollen, wird diese Schnittstelle für die Erstellung des VO-Monitoringkonzeptes betrachtet. Der Benutzer oder die Applikation, der die Schnittstelle nutzen möchte, muss entsprechende Administratorrechte (durch X.509 Zertifikate) besitzen.

In einem Grid, das durch das Globus Toolkit aufgebaut wurde, kann VOMS als Software für Management von Autorisierungsinformationen eingesetzt werden. Damit dies möglich ist, muss die Autorisierungskomponenten von GT4 durch neue Bibliotheken (*Policy Decision Point - PDP*) erweitert werden, um die VOMS-Attribute erkennen zu können. Eine Installation ist also nötig, weil das GT4 standardmäßig VOMS-Attribute nicht unterstützt. Ein VOMS-Attribut ist eine Zeichenkette, die die Mitgliedschaft eines Benutzers zu einer Gruppe repräsentiert. Beispielweise repräsentiert das Attribut */VO1/Mitarbeiter/Role=Administrator/Capability=NULL* alle Mitglieder der „VO1“ in der Gruppe „Mitarbeiter“ mit der Rolle „Administrator“ . Dieses Beispiel zeigt auch die Syntax der Attribute. Die VOMS-Attribute werden gewöhnlich in der Zugriffskontrollliste eingetragen und legen somit fest, welche Benutzer aus welcher VO-Gruppe oder mit welcher Rolle die Ressourcen verwenden dürfen.
Nachdem die Autorisierungsmodule von GT4 für VOMS-Attribute angepasst sind, kann dann die Autorisierung auf Ressourcen mittels VOMS-Attributen stattfinden. Das wird umgesetzt, indem die Zugriffskontrolllisten, die mit den Ressourcen assoziiert sind, auch oder nur VOMS-Attribute enthalten. Somit wird die Autori-

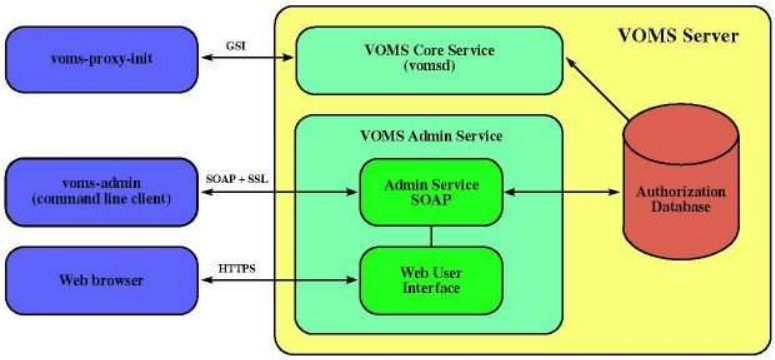


Abbildung 2.6: Die VOMS Architektur [VOMS]

sierung auf Grid-Ressourcen nicht mehr für einzelne Benutzer festgelegt (wie bei der Gridmap-File), sondern für eine Gruppe von Benutzern. Dieser Ansatz ist gut, wenn man in einer groß-skalierten Grid-Umgebung Ressourcenzugriff erlauben möchte. Das unterstützt Ressourcenbesitzer insofern, als sie nicht die Identität aller Benutzer benötigen, sondern die Zugriffskontrolle auf die Mitgliedschaften in einer VO reduzieren können.
Wenn ein Benutzer eine Grid-Ressource oder einen Dienst nutzen möchte, verwendet er den Befehl *voms-proxy-init*, um ein Proxy-Zertifikat zu generieren. Dieses Zertifikat enthält die VOMS-Attribute als Erweiterung (*Certificate Extensions*). Das heißt, das Proxy-Zertifikat enthält die Gruppe und die Rolle des Benutzers. Diese Informationen werden beim Zugriff auf die Ressourcen extrahiert und mit den Gruppen und Rollen in den Zugriffskontrolllisten verglichen. Ist die Benutzergruppe oder die Benutzerrolle in dieser Liste enthalten, wird der Zugriff gestattet, anderenfalls wird er abgelehnt. So findet die Autorisierung statt, wenn man VOMS für das Management von Benutzerrechten einsetzt. Es gibt andere Tools (CAS [CAS], Permis [PERMIS], Liberty [LIB], Shibboleth [GridShib2]), die die gleiche Funktionalität liefern. CAS und Shibboleth werden im Kapitel 4 beschrieben. Mehr über diese Werkzeuge ist in der entsprechenden Literatur zu finden.

2.6 Zusammenfassung

In diesem Kapitel wurden die Begriffe *Grid* und *Virtuelle Organisation* eingeführt. Anhand konkreter Beispiele wurden die Einsatzmöglichkeiten von Grids dargestellt. Da in dieser Arbeit nicht nur die Funktionalität, sondern auch die Struktur des Grids zum Verständnis wichtig sei, wurde mit der Open Grid Services Architecture der aktuelle Standard kurz vorgestellt. Darauf aufbauend wurden die Tools erwähnt, die für die Realisierung dieser Arbeit eingesetzt werden. Damit sind nun die Grundlagen des Grid Computing gelegt.
Im nächsten Kapitel wird zunächst ein Beispielszenario beschrieben, aus dem der Anforderungskatalog erstellt wird. Dieser Katalog wird dazu dienen, bestehende und in dieser Arbeit zu erstellende Ansätze zu analysieren und zu bewerten.

3 Anforderungsanalyse

Bei der Einleitung im Kapitel 1 wurden die Ziele dieser Arbeit erläutert. Es soll ein Monitoringkonzept für virtuelle Organisationen erstellt werden.
Im diesem Kapitel wird zu Beginn ein Beispielszenario für VO-Monitoring beschrieben. Hierbei werden die Ziele der Arbeit nochmals im Beispielkontext vorgestellt und darauf aufbauen wird ein Anforderungskatalog erstellt. Dieser Katalog wird dann als Grundlage zur Analyse, Entwicklung und Bewertung des Monitoringkonzeptes dienen. Im zweiten Teil wird ein Testbett vorgestellt, das Installationen am MNM-Institut (*Munich Network Management*) beschreibt, die für die Einrichtung einer VO eingesetzt werden. Aus diesem Testbett werden Beispieldaten bei der Konzeptentwicklung (im Kapitel 5) und -Implementierung (im Kapitel 6) verwendet.

3.1 Szenarien

In diesem Abschnitt werden zwei Szenarien beschrieben. Das erste Szenario beschreibt die Einrichtung einer VO und wie die bereitgestellten Dienste mit den VO-Migliedern interagieren. Das zweite Szenario, das auf dem Ersten aufbaut, beschreibt das Monitoring der eingerichteten VO. Aus diesem werden dann die Anforderugen des VO-Monitorings im Abschnitt 3.2 folgen.

3.1.1 Szenario 1: VO-Aktivitäten

Für die Verwaltung einer VO wird VOMS verwendet. VOMS wurde bereits im Kapitel 2.5 eingeführt und wird hier nicht mehr vorgestellt. Es wurden zudem bereits Autorisierungsverfahren mit VOMS-Attributen im Kapitel 2.5 vorgestellt und beschrieben.
VOMS wird von einem Administrator, dem VO-Manager, verwaltet, der die Rechte besitzt, beliebige Benutzer zu gruppieren oder sogar deren Mitgliedschaft zu beenden. Im diesem Szenario existieren drei verschiedene Benutzer (Benutzer-A, Benutzer-B und Benutzer-C), die sich bei dieser VO registriert haben. Der VO-Manager kann diesen Benutzern (VO-Mitgliedern) Rechte einräumen, indem er sie Gruppen zuordnet. Welche Rechte ein Benutzer erhält, hängt von dem Agreement ab, das er bei seiner Registrierung abgeschlossen hat. Unabhängig davon werden die Ressourcen Ressource-A und Ressource-B jeweils von den Institutionen Institution-A und Institution-B betreut, die autonom festlegen, welche Benutzer oder Gruppen von Benutzern ihre bereitgestellten Ressourcen verwenden dürfen. Die Abbildung 3.1 zeigt alle Personnen, Institutionen oder Ressourcen, die in dieser VO interagieren.

Dieses Beispielszenario umfasst drei Hauptaktivitäten:

- Die VO-Mitglieder (Benutzer-A, Benutzer-B und Benutzer-C) versuchen asynchron auf die bereitgestellten Ressourcen zuzugreifen.
- Die Ressourcenbesitzer (Institution-A und Institution-B) ändern ihre Ressourcen-Autorisierungsregeln nach einem bestimmten Zeitintervall.
- Der VO-Manager ändert die Rechte der VO-Mitglieder nach einem bestimmten Zeitintervall.

Diese drei Aktivitäten spiegeln reale Situationen wider, in denen die Akteure unabhängig voneinander operieren und so zur Dynamik des Gesamtsystems beitragen.

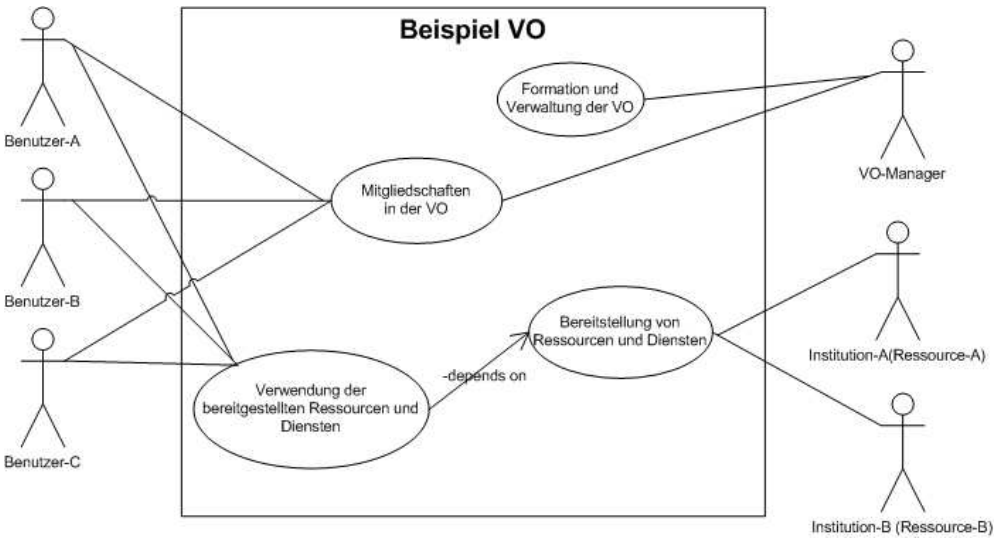


Abbildung 3.1: Interaktionen in einer VO

3.1.2 Szenario 2: VO-Monitoring

Das zweite Szenario beschreibt das eigentliche VO-Monitoring, das als Grundlage für weitere Überlegungen in dieser Arbeit betrachtet wird.
Im vorherigen Szenario wurde gezeigt, welche Aktivitäten in einer Beispiel-VO stattfinden und wie sie voneinander anhängen. In dieser VO gibt es zwei Entitäten (die Ressourcen und die VO-Mitglieder), auf die diese Aktivitäten wirken. Startet beispielsweise der Benutzer-A einen Dateitranfer auf der Ressource-B, dann wird der Status dieser Ressource geändert. Wenn der VO-Manager die Zugriffsrechte eines VO-Mitglieds ändert, kann es unter Umständen geschehen, dass es einen beschränkten oder gar keinen Zugriff mehr auf bestimmte Ressourcen hat.
Ziel des VO-Monitorings ist, solche Änderungen möglichst zeitnah zu erfassen. Von Interesse ist nicht der Zustand des VO-Managers oder der Institutionen-A, B und C, sondern der Zustand der VO-Elemente (Mitglieder und Ressourcen/Dienste) und der globale Zustand der VO selbst.

3.2 Erstellung des Anforderungskatalogs

Aus den beschriebenen Szenarien lässt sich eine Reihe von Anforderungen an einen VO-Monitor herleiten. Im Folgenden werden sie erläutert und beschrieben.
Allgemein bezeichnet ein Monitoringsystem ein System für die systematische Erfassung, Beobachtung oder Überwachung eines Vorgangs oder Prozesses mittels technischer Hilfsmittel oder anderer Beobachtungssysteme. In dieser Arbeit handelt sich um die Erfassung und Überwachung aller Einheiten und Aktivitäten innerhalb einer VO. Anhand der vorgestellten Szenarien können die folgenden Anforderungen für das VO-Monitoring gestellt werden. Im Folgenden steht der Begriff „*VO-Monitor*“ für die VO-Monitoring-Anwendung, die in dieser Arbeit entwickelt werden soll:

1. **Anforderung 1. Überwachung der VO Struktur**
   Wie in früheren Kapiteln bereits beschrieben, haben VO eine wohl definierte Struktur, die sich im Laufe der Zeit ändert. Eine Änderung kann auftreten wenn:
   - neue Gruppen oder Rollen hinzugefügt oder gelöscht werden,
   - neue Mitglieder hinzugefügt oder gelöscht werden,
   - die VO zerstört wird,
   - oder eine neue VO innerhalb der aktuellen VO gebildet wird.

Der VO-Monitor soll trotz dieser Dynamik, alle Gruppen, Rollen, Mitglieder erfassen können.

2. **Anforderung 2. Erfassung der VO Mitglieder**
   Mitglieder einer VO kommen und gehen dynamisch: Manche werden vom VO-Manager gekündigt, manche stellen Ressourcen bereit, andere besitzen keine eigenen Ressourcen. Ein VO-Monitor soll in der Lage sein, alle Mitglieder einer VO zu einer beliebigen Zeit zu erfassen. Es soll möglich sein, Informationen wie Rolle, Status, Name, Institution, Ort, eigene Ressourcen, etc., über diese Mitglieder zu besitzen.

3. **Anforderung 3. Erfassung der Ressourcen und Diensten in einer VO**
   Grid-Ressourcen und Dienste werden durch fremde Institutionen oder selbst von VO-Mitgliedern bereitstellt. Hier ist es von großem Interesse, zu wissen, welche Ressourcen in der VO verfügbar sind und wer auf sie zugreifen darf. Zudem soll der VO-Monitor alle Ressourceneigenschaften zu beliebiger Zeit liefern können: Typ, Beschreibung, Funktionalität, Verfügbarkeit, Besitzerinformationen, Status, etc.

4. **Anforderung 4. Erfassung von Jobs in einer VO**
   Ziel der Bereitstellung von Ressourcen in einer VO ist es, anderen VO-Mitgliedern die Möglichkeit zu geben, Jobs auf diesen Ressourcen auszuführen. Ein Job kann ein einfacher Systemaufruf, ein Dateitransfer, eine Dateispeicherung, eine Datenbankabfrage, etc. sein. Ein VO-Monitor soll die Möglichkeit bieten, alle Informationen und Eigenschaften über alle verfügbaren Jobs in einer VO zu haben, unabhängig davon, ob sie gestartet sind oder sie auf ihre Ausführung warten. Von Interesse sind hier insbesondere die Startzeit, die Identität des Aufrufers, die verwendeten Ressourcen, die Beschreibung des Jobs, die aktuelle Dauer des Jobs, etc.

5. **Anforderung 5. Erfassung der Beziehung Ressource - VO-Mitglieder**
   Aus Monitoring-Sicht ist nicht nur der Status der betrachteten Elemente allein wichtig, sondern auch ihre Beziehungen und Interaktionen zueinander. Daher muss der VO-Monitor beispielsweise Informationen darüber liefern können, welche Mitglieder auf welche Ressourcen in der VO zugreifen dürfen oder von wem die verschiedenen Jobs gestartet wurden.

6. **Anforderung 6. Nicht -Funktionale Anforderungen (Quality of Service Requirements)**[1]
   Neben den funktionalen Anforderungen (Anforderugen 1 bis 5) sind auch nicht-funktionale Anforderungen zu beachten (nach dem ISO 9126 Standard)[2]:

   - *Verwendung von Globus Toolkit 4*: Diese Arbeit beschränkt sich auf Globus-basierte Grids, da Globus Toolkit 4 (GT4) die OGSA-Standards implementiert. Der VO-Monitor muss daher alle Ressourcen erfassen können, die mittels Globus Toolkit 4 bereitgestellt wurden.
   - *Leistungsverhalten/Effizienz*: Der VO-Monitor soll möglichst zeitnah und exakt alle verfügbaren Informationen in einer VO erfassen können.
   - *Zuverlässigkeit*: Der VO-Monitor erfasst den Status verteilter Elemente, mit denen er mittels Nachrichten kommuniziert. Trotz dieser loosen Kopplung soll er in der Lage sein, Fehler abzufangen und darüber hinaus spezielle Routinen für Fehlerbehandlung oder Wiederherstellung starten.
   - *Skalierbarkeit*: Das System (VO-Monitor) muss gleichzeitig viele VOs überwachen und mit großen Datenmengen umgehen können.
   - *Benutzbarkeit*: Es soll eine einfache Benutzerschnittstelle (Frontend) implementiert werden, die jedem Benutzer, der für das Monitoring einer VO verantwortlich ist, Informationen in einfach lesbarer Form zur Verfügung stellt.
   - *Sicherheit*: Der VO-Monitor soll nur vom autorisierten Benutzer verwendet werden.
   - *Übertragbarkeit* (engl.*Portability*): Die erstellte Applikation soll in möglichst vielen Systemen (mit unterschiedlichen Hardware) betrieben werden können.

[1] Eine Nicht-Funktionale Anforderung ist eine Beschreibung einer Beschaffenheit oder einer Charakteristik, die ein System aufweisen muss oder eine Rahmenbedingung die eingehalten werden muss, die den Freiheitsgrad bei der Konstruktion des Systems (also die Umsetzung der funktionalen Anforderungen) einschränkt.

[2] `http://www.issco.unige.ch/projects/ewg96/node13.html`

3.3 Beschreibung eines Testbetts

Das Testbett besteht in der Einrichtung von Ressourcen und Diensten, die dann in einer Test-VO bereitgestellt werden. In der Forschungsgruppe MNM (*Munich Network Management*) wurde im Rahmen einiger Projekte [GLO05], [VOMS05] eine einfache Grid-Infrastruktur aufgebaut. Diese Infrastruktur besteht aus vier Rechnern (Grid1, Grid2, Grid3 und Grid4), die alle das Globus Toolkit 4 installiert haben. Sie können zusammen benutzt werden, um eine virtuelle Organisation aufzubauen. In diesem Testbett beschränken wir uns auf zwei Rechner (Grid1 und Grid3).
Es wird zunächst beschrieben welche Art von Grid-Diensten und Grid-Ressourcen auf diese Rechner bereitgestellt werden und wozu sie verwendet werden können. Anschließend wird beschrieben, wie die installierten Softwarekomponenten funktionieren, um VO-spezifische Anwendungsfälle zu realisieren. Im Kapitel 7 wird anhand dieses Testbetts das zu entwickelnde VO-Monitoringkonzept bewertet.
Die Beschreibung der Installationen erfolgt an dieser Stelle, da sie spezielle Grid-Dienste (RFT und WS-GRAM) vorstellt, die in den weiteren Kapiteln verwendet werden.

3.3.1 Bereitstellung eines Dienstes für zuverlässigen Dateitransfer

Bevor VOs eingerichtet werden können, müssen zunächst die Grid-Ressourcen und -Dienste erstellt werden, die dann in diesen VOs bereitgestellt werden sollen.
Es geht hier um die Erstellung eines Dienstes für zuverlässigen Dateitransfer (*Reliable File Transfer - RFT*). Das Globus Toolkit bietet dafür zwei Komponenten (*GridFTP* und RFT). Das GridFTP ist ein Protokoll, das von dem Global Grid Forum spezifiziert wurde. Es erlaubt einen sicheren und schnellen Transfer von sehr großen Dateien. Dieses Protokoll basiert auf FTP und unterstützt zusätzlich zu den normalen FTP Operationen auch das Striping und die Übertragung über Dritte (Third-party Transfer). GridFTP-Servers wurden in verschiedenen Installationen und Szenarien bewertet und haben ihre Effizienz gegenüber herkömmlichen FTP-Servern gezeigt [GridFTP]. Der RFT ist ein Web Service, der eine oder mehrere GridFTP-Übertragungen zu protokollieren oder verwalten kann. Da GridFTP keine Web Services Protokolle (SOAP, WSDL) unterstützt und eine offene Socket-Verbindung an der Client-Seite während eines Dateitransfers erfordert, werden daher andere Mechanismen benötigt, die einen Transfer beliebig anhalten, stoppen oder wiederstarten können. Der RFT stellt diese flexible Funktionalität zur Verfügung. Möchte ein Benutzer beispielsweise einen GridFTP-Transfer starten, ruft er stattdessen die *create*- und anschließend die *start*-Operationen des RFT-Dienstes auf. RFT wird dann den eigentlichen GridFTP-Transfer starten und Informationen über diesen Transfer in eine Datenbank schreiben. Der Benutzer, der den Job (den Dateitransfer) gestartet hat, verfügt über eine eindeutige Transfer-ID, mit der er nachträglich über den RFT Service die gespeicherten Transferinformationen aus der Datenbank abrufen kann. Der RFT bietet also auch die Möglichkeit, den Status von Dateiübertragungen abzufragen.
In unserem Testbett wird auf Grid3 einen RFT Service bereitgestellt.

3.3.2 Bereitstellung eines Dienstes für entfernte Job-Ausführung

Im Testbett wird versucht, möglichst verschiedene Facetten einer realen Grid-Infrastruktur widerzuspiegeln. Deswegen wird ein zweiter Dienst beschrieben, der in einer VO bereitgestellt werden kann. Dieser Dienst bietet Operationen zum Starten und Monitoring von Jobs an. Zur Realisierung dieses Dienstes wird das WS-GRAM (*Grid Resource Allocation and Management*) eingesetzt. Das WS-GRAM ist eine Software-Komponente von Globus Toolkit 4, die Mechanismen für das Management von Jobs innerhalb einer Grid--Infrastruktur zur Verfügung stellt. Es bietet Web Services Schnittstellen für die Kreation, das Management und das Monitoring von Jobs auf entfernten Maschinen an. Das WS-GRAM kann auch für das Management GridFTP-Transfers eingesetzt werden. Sein Vorteil gegenüber RFT ist, dass es nicht nur für Dateitransfer-Jobs, sondern für alle anderen Arten von Jobs eingesetzt werden kann.
WS-GRAM besteht aus zwei Hauptdiensten: dem *ManagedJobFactory Service* und dem *ManagedJob Service*. Der ManagedJobFactory Service ist ein Dienst, der Operationen für die Kreation von Job-Ressourcen anbietet. Diese Job-Ressourcen (oder auch Jobs) heißen dann *ManagedJob*. Nachdem ein Benutzer einen ManagedJob

durch die *createManagedJob*-Operation des ManagedJobFactory Services angelegt hat, bekommt er eine eindeutige ID von diesem Job und darüber hinaus die totale Kontrolle über den Job. Somit kann er nachträglich Operation wie Stoppen, Anhalten, Wiederstarten auf diese Job-Ressource ausführen.
Ein Benutzer startet einen Job auf einer Maschine über eine SOAP-Nachricht. Der Jobauftrag wird dann von einem lokalen Job Scheduler (z.B. *fork*) interpretiert und zu der zuständigen Ressource weitergeleitet. Außerdem können die Job-Informationen wie Zustand, Startzeit, Job-Besitzer aus diesem Scheduler zum WS-GRAM Service publiziert werden. Fork ist ein lokaler Scheduler, der direkt die Betriebsystemfunktionalität verwendet. Möchte man komplizierte oder aufwändige Jobs ausführen, die möglicherweise eine andere Infrastruktur (z.B. Computer-Cluster) voraussetzen, dann kann man andere Job Schedulers einsetzen (z.B *PBS (Portable Batch System)*[3], *LSF*[4], *Loadleveler*[5], *Condor*[6] [TTL 02]).
Möchte man beispielsweise ein einfaches Betriebsystemprogramm ausführem, wird dann der Fork Job Scheduler verwendet, der den Betriebsystemaufruf fork() von Unix nutzt, um neue Jobs zu starten. Die Jobinformationen werden in diesem Fall durch das SEG *(Scheduler Event Genarator)* [GT4] zum WS-GRAM Service propagiert. Die Abbilddung 3.2 zeigt ein Beispielszenario für den Einsatz von WS-GRAM.

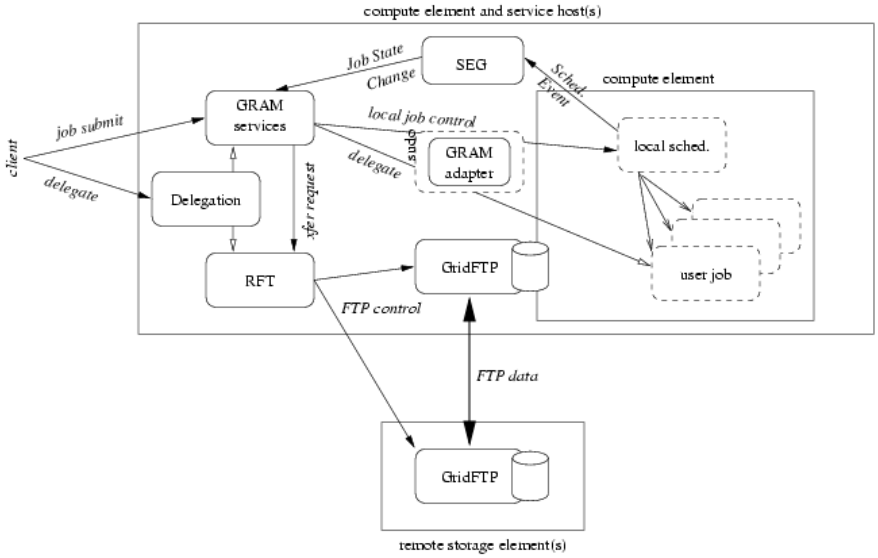


Abbildung 3.2: Einsatz von WS GRAM [GT4]

In unserem Testbett wird auf Grid1 ein WS-GRAM Service bereitgestellt. Für Testzwecke wird ein Skript geschrieben, der wiederholt ausgeführt wird. Dieses Skript, das den einfachen Betriebsystemaufruf „*/bin/date*“ ausführt, soll dann durch einen autorisierten Benutzer über den WS-GRAM-Dienst gestartet werden. Die Ausgabe der Ausführung (also das aktuelle Datum) wird in einer angegebenen Ausgabedatei geschrieben. Somit können Status-Informationen wie Anzahl gestarteter Jobs, Anzahl fertige Jobs, etc. jederzeit durch den WS-GRAM Service abgefragt werden.

3.3.3 Bereitstellung von Diensten für die Registrierung und Entdeckung von Grid-Ressourcen

In den vorangehenden Abschnitten wurde viel Wert auf die Publizierung von Ressourcen- und Job-Eigenschaften gelegt. Da das Ziel dieser Arbeit die Entwicklung eines Monitoringkonzeptes ist, das alle Informationen über Ressourcen und Benutzer in einer VO liefern soll, ist es notwendig, die Ressourcen so bereitzustellen, ihre Status und Eigenschaften so zur Verfügung zu stellen, dass andere Benutzer sie durch spezielle Verzeichnisdienste durchsuchen und finden können. Damit eine Ressource im Grid überhaupt benutztbar wird, muss

[3] `http://www.openpbs.org`
[4] `http://www.platform.com/Products/Platform.LSF.Family/Platform.LSF/`
[5] `http://www.redbooks.ibm.com/redbooks/pdfs/sg246038.pdf`
[6] `http://www.cs.wisc.edu/condor/`

sie sich mindestens bei einem Verzeichnisdienst registriert haben. Wir setzen also im Folgenden voraus, dass alle verwendeten Grid-Ressourcen bereits bei einem Verzeichnisdienst registriert sind.
Im Globus Toolkit 4 wird so ein Verzeichnisdienst durch den MDS-Index Service implementiert. Das MDS wurde bereits im Kapitel 2 eingeführt und wird hier nicht weiter betrachtet. Wie man aber eine Ressource über den Index Service registriert und wie man Ressourceneigenschaften abruft, wird in Details in den Kapiteln 4 und 5 beschrieben.
Im Testbett werden beide Dienste (RFT und WS GRAM) bei einem einzigen Index Service (VO-Index - siehe Kapitel 4.2) registriert. Der Index Service wird selbst im Grid3 bereitgestellt.

3.3.4 Einrichtung einer Test-VO

Die in den oberen Abschnitten beschriebenen Ressourcen und Dienste werden dann in einer Test-VO bereitgestellt. Wie man eine VO einrichtet, wurde bereits im Kapitel 3.1 bei der Szenariobeschreibung gezeigt. In dieser Test-VO wird der VO-Monitor implementiert, getestet und bewertet werden.

3.4 Zusammenfassung

In diesem Kapitel wurde ein Beispielszenario beschrieben. Aus dem Szenario wurden die Anforderungen an ein Monitoringkonzept abgeleitet, das zu einer beliebigen Zeit Informationen darüber zur Verfügung stellt: wer die Mitglieder einer VO sind, welche Ressourcen in der VO bereitgestellt wurden, wie lange diese Ressourcen verfügbar sind, welche Jobs ausgeführt werden und welche Zugriffsrechte die VO-Mitglieder auf die Ressourcen haben. Es wurde zudem ein Testbett vorgestellt, in dem eine VO mit Ressourcen und Ressourcen Management Diensten eingerichtet wurde.
Im nächsten Kapitel werden bestehende Management und Monitoring Tools anhand der ober spezifizierten Anforderungen vorgestellt und bewertet.

4 State-of-the-Art im VO-Monitoring

In den vorangegangenen Kapiteln wurden die Grundlagen des Grid Computing dargestellt. Zudem wurden erste Ideen über VO-Monitoring formuliert und Anforderungen für ein VO-Monitoringkonzept erstellt.
In diesem Kapitel wird der State-of-the-Art im VO-Monitoring analysiert und bewertet. Es soll vor allem das Monitoring-Potenzial existierender Tools anhand der in Kapitel 3.2 erläuterten Anforderungen (insbesondere die funktionalen Anforderungen - Anforderung 1 bis 5) bewertet werden. Es wird sich zeigen, dass die verwendeten Tools, alleine betrachtet, die VO-Monitoring-Anforderungen nicht erfüllen und dass ein neues Konzept benötigt wird, welches das Potenzial aller Tools aggregieren soll, um die erwünschten Anforderungen zu erfüllen.
Zuert werden in diesem Kapitel Tools für das Management von Autorisierungsinformationen und Ressourcen in einer VO beschrieben.

4.1 Management von VOs

In den vorangegangenen Kapiteln wurden VOs definiert und deren Struktur beschrieben. Es wurde außerdem VO-Management Systeme vorgestellt und gezeigt, wofür sie verwendet werden können. VO-Management Systeme spielen gewöhnlich zwei verschiedene Rollen: Einerseits dienen sie als Systeme für die Verwaltung von Benutzern, Gruppen und Ressourcen in einer VO. Andererseits implementieren sie die Autorisierunsmechanismen auf der Grid-Ressourcen-Ebene und legen dadurch die Zugriffsrechte der VO-Mitglieder fest. VOs können nicht ohne solche Systeme existieren. Sie sind notwendig für den Aufbau und die Verwaltung von VOs. Es gibt eine Vielzahl von VO-Management Systemen, die alle verschiedene Funktionalität bieten. Beispielsweise ermöglichen manche (z.B. CAS) gleichzeitig die Verwaltung von Benutzern und Ressourcen in einer VO, während andere (z.B. VOMS) nur die Verwaltung von Mitgliedern erlauben.
In diesem Abschnitt werden einige Systeme vorgestellt und deren Einsatz beim Aufbau von virtuellen Organisationen gezeigt.

4.1.1 CAS (Community Authorization Service)

CAS [CAS] wurde im Rahmen des Globus Projekts entwickelt. Die Architektur verwendet Public Key Authentifikation und basiert auf der im Globus Toolkit enthaltenen Grid Security Infrastructure (GSI). CAS ermöglicht dem Ressourcenbesitzer (*Resource Provider*) die Verwaltung der Zugriffsrechte auf seine Ressourcen an eine VO oder eine Gemeinschaft (*Community*) zu delegieren. Das heißt, ein Resource Provider erlaubt einer VO, die administrative Kontrolle über seine Ressourcen auszuüben. Dazu definiert der Resource Provider lokale Zugriffsrechtegruppen. Jeder Gruppe wird eine Menge von Ressourcen zugeordnet. Während der Resource Provider die Zugriffsrechte grob granular festlegt, übernimmt der VO-Manager die Definition fein granularer Rechte. Er legt fest, welcher Benutzer zu welcher Gruppe gehört und welcher Benutzer auf welche Ressource zugreifen darf.
Möchte beispielsweise ein Resource Provider in einer VO jeweils der Gruppe-A Leserechte und der Gruppe-B Schreibrechte zuweisen, muss er dafür den *CAS Service* kontaktieren und ihm diesen Wunsch mitteilen. In einer Handshake ähnlichen Prozedur wird festgelegt, wie die Autorisierungsinformationen dargestellt werden. Ab diesem Punkt kümmert sich der Resource Provider nicht mehr um die Verwaltung der Benutzerrechte. Die Aufgabe für die Zuteiltung der Rechte zu den einzelnen VO-Mitgliedern innerhalb einer Gruppe wird dem VO-Administrator überlassen. Der Benutzer (VO-Mitglied) erhält damit die Zugriffsrechte auf eine Ressource vom Administrator und nicht vom Ressourcenbesitzer.

Eine VO wird gegründet, indem ein *CAS Server* für die VO installiert und darauf ein CAS Service bereitgestellt wird. Alle Ressourcen, die in dieser VO zur Verfügung gestellt werden sollen, müssen sich bei dem CAS

Service registrieren (wie eben beschrieben). VO-Mitglieder registrieren sich ebenso durch diesen Service, und erhalten Rechte innerhalb der Community abhängig von der Einigung, die sie mit dem VO-Administrator abgeschlossen haben.
Typischerweise existiert für eine VO ein einziger CAS Service, den von Benutzern kontaktiert werden, um ihre Genehmigung für Operationen auf Ressourcen abzufragen. Nachdem ein Benutzer den CAS Service kontaktiert hat, erhält er seine Zugangsdaten (*CAS Permissions*) in einem Proxy-Zertifikat. Mit Hilfe dieses Proxy-Zertifikats wird er beim Resource Provider autorisiert. Der Resource Provider extrahiert aus diesem Proxy-Zertifikat die Zugangsdaten, welche vom CAS Service generiert wurden. Er vergleicht diese mit den lokalen Informationen und gestattet oder lehnt eine Erlaubnis ab.

Unter dem CAS Service läuft eine Datenbank mit Informationen über die Benutzer, die Ressourcen, die Aktionen und die Zugriffsrechte. Aktionen definieren Operationen (z.B. löschen, lesen...), die auf Ressourcen ausgeführt werden können. CAS bietet eine SOAP Schnittstelle (Web Service) für administrative Zwecke. Mit dieser Schnittstelle ist es möglich, den kompletten Status der Datenbank abzufragen, Benutzer und Ressourcen in die VO aufzunehmen oder aus der VO zu löschen.
Diese Schnittstelle kann zwar für das Monitoring der VO eingesetzt werden, sie hat aber einige Nachteile. Sie liefert beispielsweise keine *detaillierten* Informationen über die Ressourcen. Da in der CAS Datenbank Ressourcen mit einer Bezeichnung und einem Namensraum repräsentiert werden, ist es nicht möglich, die Aktivitäten, die auf diesen ausgeführt werden, zu erkennen. Die Anforderungen 3, 4 und 5 werden daher nicht erfüllt.

4.1.2 GridShib/Shibboleth

GridShib[1] [GridShib2] wurde von der NSF Middleware Initiative (NMI) entwickelt. Ziel von GridShib ist eine Integration von Globus Toolkits und Shibboleth[2], um eine Attribut-basierte Autorisierung in einer VO zu ermöglichen.
Shibboleth wird vom *Middleware Architecture Committee for Educationist (MACE)* entwickelt. Es ist eine Open-Source Software, die eine Single Sign-On Authentifizierung und Mechanismen für die Zugriffskontrolle über Web-Ressourcen implementiert. Shibboleth implementiert die OASIS SAML v1.1 Spezifikation [SAML]. SAML ist ein XML-Standard zum Austausch von Authentifizierungs- und Autorisierungsdaten zwischen zwei Organisationen, also zwischen einem *Identity Provider* (produziert Aussagen über einen Benutzer in Form von XML Daten - *SAML Assertion*) und einem *Service Provider* (nutzt die Aussagen für die Autorisierung). Konzeptionell besteht Shibboleth aus drei Komponenten:

- Der *Handle Service* dient dazu, einen Benutzer zu authentifizieren. Nachdem ein Benutzer diesen Dienst aufgerufen hat, wird die Authentifizierungsanfrage zu einem weiteren Dienst, in der Heimatorganisation des Benutzers (*home organisation*), weitergeleitet. Falls die Authentifizierung bei der Heimatorganisation erfolgreich ist, wird der Handle Service informiert und ein *Handle Token* dem Benutzer zurückgegeben. Dieser Handle Token, der gewöhnlich eine SAML Assertion spezifiziert, enthält die Identität des Benutzers (in kodierter Form). Der Benutzer ist damit authentifiziert. Für die Authentifizierung bei der Heimatorganisation kann ein beliebiger Dienst (z.B. LDAP) eingesetzt werden.

- die *Attribute Authority (AA)* ist ein Server, der Informationen (wie Zertifikat, Rolle, Funktion, Gruppe, etc.) über einen Benutzer in Form von Attributen speichert. Nachdem ein Benutzer einen Handle Token erhalten hat, kann er diesen nutzen, um sich bei dem Resource Provider zu autorisieren. Dabei wird der Resource Provider (*Target Resource*) die Benuter-ID extrahieren, sie zu dem *Shibboleth Attribute Authority Service* senden und dann nach den richtigen Attributen des Benutzers fragen. Die Attribute werden auch hier als SAML Assertions zurückgegeben.

- die *Target Resource* ist der Resource Provider. Er verwendet Shibboleth-Bibliotheken, die es ihm ermöglichen, die Heimatorganisation des Benutzers und die richtige Attribute Authority (AA) zu finden. Sobald die Benutzer-Attribute bei einer AA abgefragt wurden, kann der Resource Provider basierend auf diesen Attributen entscheiden, ob der Benutzer die angeforderte Ressource benutzen darf.

[1] `http://gridshib.globus.org/`
[2] `http://shibboleth.internet2.edu/`

Die Abbildung 4.1 zeigt die Architektur von Shibboleth und den Informationsfluss bei der Autorisierung. Die Ziffern bestimmen die Reihenfolge der Operationen. Im Schritt 3 registriert der Handle Service den Benutzer bei einer Attribute Authority. Dabei wird ein Handle Token generiert, der zum Benutzer zurückgegeben wird (Schritt 4). Im Schritt 5 sendet der Benutzer eine Anfrage für einen Zugriff zu dem Resource Provider mit seinem Handle Token. Bevor dem Benutzer den Zugriff zu erlauben, muss der Resource Provider die Identität dieses Benutzers prüfen. Um die Identität zu prüfen, kontaktiert er einen Shibboleth Service und sendet ihm den erhaltenen Handle Token (Schritt 6). Im Schritt 7 wird die Anfrage nach der Identität zu der Heimatorganisation des Benutzers weitergeleitet. Falls die Heimatorganisation den Benutzer erkennt, können dann im Schritt 8 Attribute, die den Benutzer richtig identifizieren, zurückgegeben werden. Shibboleth kümmert sich also um die Verwaltung der Identität zwischen den verschiedenen Verhandlungspartnern.
Eine VO kann gegründet werden, indem verschiedene Benutzer aus verschiedenen Heimatorganisationen und verschiedene Resource Providers ein Konsortium (*federation*) bilden. Dafür kann z.B. eine Attribute Authority angelegt werden, die nur Attribute der VO-Mitglieder enthält.

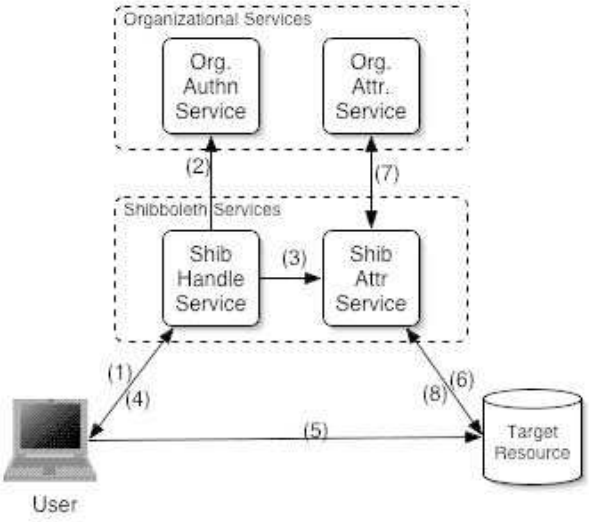


Abbildung 4.1: Shibboleth [GridShib2]

GridShib [GridShib1] nutzt die Autorisierungsinfrastruktur von Shibboleth und die Grid Security Infrastructure (GSI) von Globus Toolkit (GT), um Autorisierungmechanismen für VOs zu realisieren. Die Autorisierung basiert auf Shibboleth Attributen (SAML Assertions). GridShib besteht aus einem Plugin für GT (*Grid SP - Grid Service Provider*) und einem Plugin für Shibboleth (*IdP - Identity Provider*). Grid SP ist ein normaler GT-Dienst (z.B. RFT), der aber den GridShib PDP (Policy Decision Point) als Autorisierungsmodul implementiert hat. Der PDP ist dafür verantwortlich, den IdP zu kontaktieren und die Autorisierung zu gestatten oder abzulehnen. Der Resource Provider, der diesen Dienst bereitstellt, muss auch für die freigegebenen Ressourcen Zugriffskontrolllisten anlegen. Diese Zugriffskontrolllisten (siehe Kapitel 2.5) enthalten SAML Assertions, die festlegen, welche Benutzer oder Benutzergruppen Zugriffsrechte auf die Ressource haben.

Die Abbildung 4.2 zeigt ein klassisches GridShib Szenario. Die Ziffern bestimmen die Reihenfolge der Operationen. Im ersten Schritt ruft ein Benutzer einen Grid SP (z.B. GRAM oder RFT) auf. Dieser Service Provider extrahiert den Benutzer DN (Distinguished Name) aus dem X.509-Zertifikat. Im zweiten Schritt formuliert der Grid SP eine Anfrage mit dem extrahierten DN an den Identity Provider des Benutzers. Aus der Attribute-Authority (AA) des IdP werden im dritten Schritt die Benutzersattribute abgefragt. Die Antwort wird als SAML Assertion an den Grid SP zurückgeschickt. Ab diesem Punkt kann dann der Grid SP diese Attribute mit den lokalen Werten vergleichen und eine Autorisierungsentscheidung treffen. Ist der Benutzer berechtigt, diese Ressource zu benutzen, dann erhält er die Ergebnisse seines Aufrufs, ansonsten eine Ablehnung wegen mangelnder Zugriffsrechte.

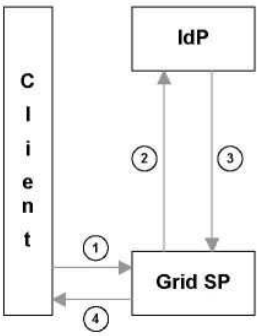


Abbildung 4.2: Klassisches GridShib Szenario [GridShib1]

Shibboleth bietet SOAP Schnittstellen an und verwaltet Benutzerattribute. Es ist daher möglich, Dienste zu definieren, die Informationen über VO-Mitglieder liefern können. Leider speichert Shibboleth keine Informationen über Grid-Ressourcen. GridShib selbst ist nur ein Plugin für Grid-Ressourcen und besitzt keine Komponenten für die Speicherung von Ressourceneigenschaften. Würde man GridShid und Shibboleth für den Aufbau eines VO-Monitors einsetzen, könnten daher die Anforderungen 3, 4 und 5 nicht erfüllt werden.

4.1.3 VOMS/VOMRS

Im Kapitel 2.5 wurde VOMS bereits beschrieben. Es wurde auch gezeigt, wie VOMS in das Globus Toolkit integriert werden kann, um die Realisierung der Autorisierungsmechanismen und die Verwaltung der Mitglieder in einer VO zu gewährleisten. In diesem Abschnitt wird mit VOMRS *(VO Management Registration Service)* ein zu VOMS komplementärer Dienst kurz beschrieben. VOMRS ist eine Erweiterung vom VOMS, der Dienste für die automatische Registrierung von Benutzern bei einer VO zur Verfügung stellt. VOMRS ist die Hauptkomponente von *VOX (VO eXtension* [VOX]. Der VOX Projekt wurde vom Fermilab[3] in Kollaboration mit U.S. CMS[4] gestartet mit dem Ziel, VOMS zu erweitern und zu verbessern. Wenn sich ein Benutzer bei einer VO nicht durch den VOMS Administrator registrieren konnte, musste sich der VO-Manager manuell um die Registrierungen kümmern. Für eine kleine VO mit wenigen Mitgliedern ist dies zwar machbar, für große VOs jedoch nicht mehr. VOMRS bietet daher automatische Prozessabläufe für die Registrierung und die Aufnahme von VO-Mitgliedern. VOMRS erweitert auch die Möglichkeit, mehr Informationen (z.B. Email, Addresse, etc) über ein VO-Mitglied zu speichern. Die Abbildung 4.3 zeigt die Architektur von VOMRS und den Informationsfluss während einer Registrierung. VOMRS besteht hauptsächlich aus einer Datenbank (VOMRS DB), einem Web Client, einem Command Line Interface (CLI) und einer Schnittstelle für die Administration (VOMRS Admin). Der Web Client und das CLI kommunizieren mit dem VOMRS Admin über SOAP Nachrichten. Sobald ein Benutzer mit der Registrierung beginnt, wird im Server ein Registrierungsprozess gestartet, der die Kontrolle über alle Schritte der Registrierung besitzt. Auf der Benutzerseite läuft der Registrierungsprozess über mehrere Web-Formulare. Zur Bestätigung der Registrierung wird dem Benutzer eine Email geschickt (sendmail). Sobald eine Registrierung erfolgreich ist, können dann die Daten relativ zur Mitgliedschaft des Benutzers in der VO in die VOMS Datenbank publiziert werden. VOMRS kommuniziert mit VOMS durch den VOMS Admin. Die Arbeit von VOMS wird durch diesen Prozess nicht beanträchtigt. Der VO-Manager benutzt den VOMS Admin wie gewohnt um die VO-Mitglieder-Zugriffsrechte zu verwalten.

Jede VO besitzt einen VOMRS Service und einen VOMS Service. Weil die verschiedenen Komponenten über SOAP kommunizieren, können die Server auf dem gleichen Rechner oder auf weit entfernten Maschinen installiert werden. VOs werden gewöhnlich durch große Institutionen (Universitäten, Rechenzentren, große Unternehmen, etc) gegründet. Diese Institutionen stellen auch ihre Ressource bereit, damit sie von der Community verwendet werden können. Bevor ein Benutzer sich für eine VO registrieren kann, muss er zumindest Partner von einer Institution sein oder irgendeine Verbindung mit einer dieser Institutionen haben.

Für das VO-Monitoring bedeutet dies: Durch den VOMS Admin ist es möglich, Informationen über die Mit-

[3] http://www.fnal.gov/
[4] http://www.uscms.org/

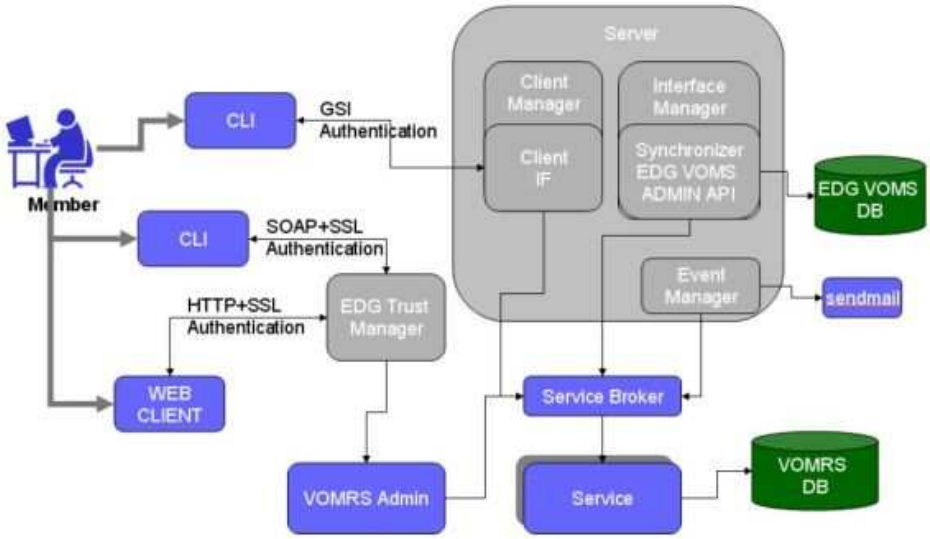


Abbildung 4.3: VOMRS Architektur [VOX]

gliedschaft der Benutzer in der VO abzufragen. Durch den VOMRS Admin ist möglich, andere Informationen wie Benutzerzertifikat, Registrierungszeit, Benutzerprofil, abgeschlossene Verträge, etc, zu bekommen. Somit sind die Anforderungen 1 und 2 erfüllt. Anforderungen 3, 4 und 5 sind nach wie vor nicht erfüllt, da es keinerlei Möglichkeit gibt, durch diese Schnittstellen Informationen über die Grid-Ressourcen zu erhalten.

4.2 Monitoring von Grid-Ressourcen

Im Kapitel 2.4 wurde bereits WS MDS definiert. In diesem Abschnitt wird nun beschrieben, wie die verschiedenen Komponenten von WS MDS funktionieren und es wird gezeigt, wie diese Komponenten zusammen interagieren.
Das MDS *(Monitoring and Discovery System)* bietet Mechanismen zum Publizieren und Entdecken von Ressourceneigenschaften und Konfigurationsinformationen. Die frühere Version von MDS (MDS2 oder Pre-WS MDS) folgt keiner SOA-Architektur und bietet daher auch keine Web Services Schnittstellen. Stattdessen werden die Ressourceneigenschaften in einem LDAP Server (OpenLDAP[5]) publiziert. Die publizierten Informationen können dann von jedem autorisierten Benutzer durch einen LDAP Browser oder Web Browser (z.B. durch eine PHP Web-Applikation) abgefragt und gelesen werden. In der aktuellen Version von Globus Toolkit (GT4) ist der Pre-WS MDS nicht standardmäßig installiert und gilt als veraltet (*deprecated*). Wir konzentrieren uns deswegen auf das WS MDS, die aktuellste Version von MDS. Es folgt einer SOA-Architektur und bietet Web Services (WSRF konform) an.
WS MDS bildet eine Suite von Web Services, die das Monitoring und die Entdeckung von Grid-Ressourcen und Diensten ermöglicht. Mit WS MDS ist es möglich, herauszufinden, welche Ressourcen zu einer bestimmten VO gehören und welche Informationen über diese Ressourcen relevant sind. Im Kapitel 3.3 wurde gezeigt, dass Grid-Ressourcen nur über Web Services (z.B. WS GRAM, RFT) zugreifbar sind. Diese Web Services nutzen bestimmte Tools (*Information Provider*), um die Eigenschaften der Grid-Ressourcen über die Web Services verfügbar zu machen. Die Web Services repräsentieren also die Ressource, und die Eigenschaften oder Zustände dieser Ressourcen werden als Web Services Zustände dargestellt. Diese Mechanismen werden im Rahmen des *WSRF (Web Service Resource Framework)* spezifiziert. Die komplette Beschreibung WSRF ist nicht Bestandteil dieser Arbeit und kann in der entsprechende Literatur [WSRF] gefunden werden. Im Folgenden werden Ressourceneigenschaften „*Resource Properties*“genannt, und statt WS MDS wird vereinfach die Bezeichnung „*MDS4*“verwendet.

Die Abbildung 4.4 zeigt die MDS4 Komponenten und den Informationsfluss zwischen diesen Komponenten

[5] www.openldap.org/

im Überblick.
MDS4 bietet derzeit zwei WSRF- basierte Dienste an:

- der *Index Service* sammelt Daten und bietet selbst Schnittstellen zur Abfrage dieser Daten oder zur Registrierung an. Er ist dem UDDI [UDDI] ähnlich, bietet aber mehr Funktionalität. Für jede Ressource werden die publizierten Eigenschaften durch den Index Service als *Entry* dargestellt. Ein *Entry* (*Index Entry*) ist ein wohl definiertes XML-Schema (nach der WS-ServiceGroup Spezifikation [WS-SGr]), das dazu dient, Resource Properties zu beschreiben. Beispiele von Index Entries sind im Kapitel 5 zu finden. Jedes *Entry* hat eine feste Lebensdauer und wird gelöscht, falls die Resource Properties innerhalb dieser Zeit nicht aktualisiert werden. Somit hat man ein Verfahren, das jederzeit aktuelle Daten über die Ressourcen liefern kann. Index Services können sich ansonsten bei anderen Index Services so registrieren, dass eine Hierarchie aufgebaut wird, in der alle Daten von der Wurzel abgefragt werden können, weil die Kindknoten ihre Daten (*Entry*) zu den Elternknoten weiterleiten.
- der *Trigger Service* sammelt ebenfalls Daten, kann Aktionen aber ausführen, wenn die gesammelten Daten bestimmte Bedingungen erfüllen.
- der *Archive Service*[6] (derzeit noch nicht implementiert) wird die Möglichkeit bieten, Daten zu archivieren.

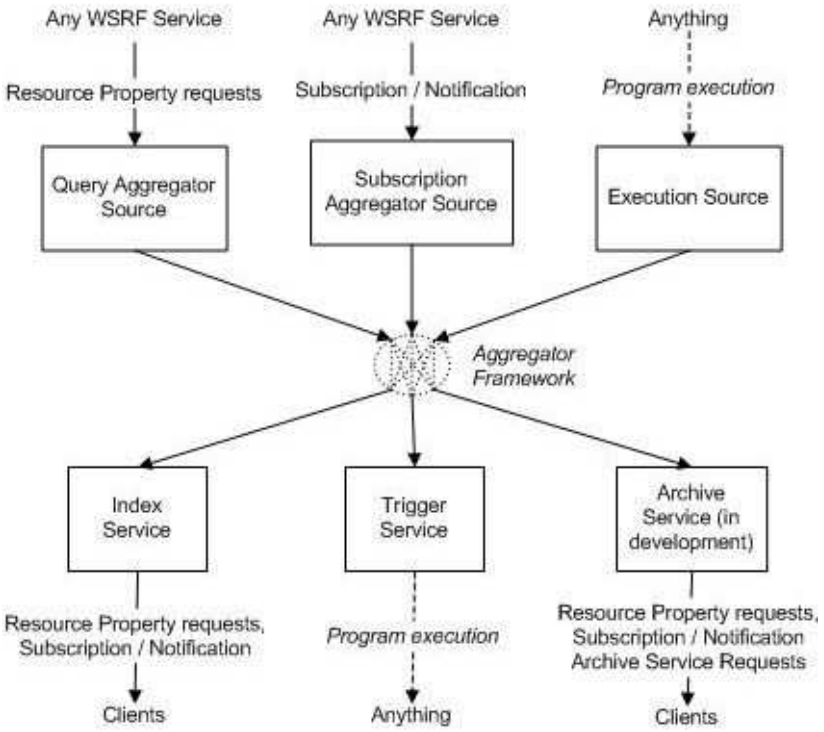


Abbildung 4.4: Informationsfluss in MDS4 [GT4]

MDS4 beruht auf einem Rahmenwerk (*Aggregator Framework*), mit dessen Hilfe Mechanismen zum Sammeln und Aggregieren von Daten definiert werden können. Das Aggregator Framework besteht aus:

- *Aggregator Source* ist eine Komponente (Java Klasse), die Informationen (XML formatierte Daten) sammelt, und sie zum Aggregator Service sendet. Es gibt drei verschiedene Typen einer Aggregator Source: die *Query Aggregator Source* sammelt Daten aus WSRF-konformen Web Services (WS GRAM, RFT, RLS, Index Service). Die *Subcription Aggregator Source* sammelt Daten nach dem *subscription/notification* Verfahren [WSRF]. Die *Execution Aggregator Source* führt ein externes Programm (meisten

[6]Der Archive Service wird erst in der GT Version 4.1.2 implementiert werden. Eine Dokumentation von diesem Service ist unter `http://www.globus.org/toolkit/docs/development/4.1.2/info/archive/` zu finden.

ein Unix-Skript) aus, um Daten zu sammeln. Durch die Execution Aggregator Source können beliebige XML Daten im Index Service publiziert werden.

- *Aggregator Service* implementiert die Dienste, die von MDS4 angeboten werden. Beispiele sind der Index Service und der Trigger Service. Nachdem die Resource Properties von der Aggregator Source gesammelt wurden, werden sie zum Aggregator Service gesendet. Der Aggregator Service verarbeitet diese Daten und stellt sie für andere Dienste oder Programme zur Verfügung.

Das Aggregator Framework basiert auf den *Web Services ServiceGroup* [WS-SGr] und *Web Services ResourceLifetime* [WS-ResLt] Spezifikationen. Für Details wird auf diese Spezifikationen verwiesen. Es stellt zusätzlich einen Dienst (*ServiceGroup*) für die Registrierung von *WS-Ressourcen* bereit. *WS-Ressourcen* sind Grid-Dienste, die die physischen Ressourcen repräsentieren (z.B. RFT- siehe Kapitel 3.3). ServiceGroups bieten die Möglichkeit, bestimmte Services in Gruppen zusammenzufassen. Die zusammengefassten Services (WS-Ressourcen) können dann durch einen zentralen Punkt (die ServiceGroup selbst) erreicht werden. Nach der Registrierung einer WS-Ressource wird ein Platzhalter (*ServiceGroup Entry*) für diese WS-Ressource erstellt. Die Konfigurationsparameter, die während der Registrierung übermittelt wurden, bestimmen, welche Aggregator Source und welcher Aggregator Service für diese WS-Ressource genutzt werden soll. Die gewählte Aggregrator Source wird dann anfangen, die Daten zu sammeln und sie zu dem Aggregator Service zu senden. Der Aggregator Service verarbeitet die erhaltenen Daten und setzt sie in den Platzhalter (der während der Registrierung für diese WS-Ressource angelegt wurde). Dieses Szenario wiederholt sich nach einem bestimmten Zeitintervall.
Vor diesem Hintergrund stellt sich die Frage, ob MDS4-Dienste für VO-Monitoringszwecke eingesetzt werden können. Die Abbildung 4.5 zeigt ein Beispielszenario, in dem verschiedene Ressourcen in einem *VO-Index* registriert werden. Der *VO-Index* ist der Index Service, der Informationen über alle registrierten Ressourcen in der VO liefert. Aus der Abbildung wird deutlich, dass Ressourcen nicht unbedingt bei dem gleichen Index Service registriert sein müssen. Allerdings können mehrere verteile Index Services zu einem einzigen VO Index aggregiert werden. Sie bilden daher eine hierarchische Struktur mit dem VO-Index als Wurzel. Der VO-Index kann so für das Monitoring der VO eingesetzt werden. Betrachtet man die VO-Monitoring Anforderungen aus Kapitel 3, kann der VO-Index die Anforderungen 1 und 2 nicht erfüllen. Durch einen derartigen VO-Index (der ein MDS4-Dienst ist) können zwar Informationen über Ressourcen und Jobs geholt werden, aber es besteht keinerlei Möglichkeit, Informationen über die VO-Mitglieder zu erhalten wie bei VOMS und Shibboleth. Die Anforderung 5 wird daher auch nicht erfüllt, da keine Beziehung zwischen Ressourcen und VO-Mitgliedern erstellt werden kann, solange keine Daten über VO-Mitglieder verfügbar sind.

4.3 Zusammenfassung

In diesem Kapitel wurden Tools für das Management von VO-Mitgliedern und das Monitoring von Ressourcen vorgestellt und bewertet. Ziel der Bewertung war, zu analysieren, ob die jeweiligen Tools für das Monitoring von VOs eingesetzt werden können. Die Bewertung basierte auf den im Kapitel 3 gestellten funktionalen Anforderungen (Anforderungen 1 bis 5). Dabei zeigte sich, dass VO-Management Systeme allgemein die Anforderungen 1 und 2 erfüllen, während Ressourcen Management Systeme eher die Anforderungen 3 und 4. Die Tabelle 4.1 fasst die Ergebnisse zusammen. Die Erfüllung der Anforderung 5 kann durch die Erfüllbarkeit aller anderen Anforderungen erreicht werden (siehe Kapitel 5.3). Daher ist es offensichtlich, dass VO-Monitoring Anforderungen eingehalten wären, falls das Monitoringspotenzial der zwei verschiedenen Toolsgruppen kombiniert wäre. Im nächsten Kapitel wird ein Konzept erstellt, das VOMS und MDS4 kombiniert nutzt, um einen VO-Monitor im Sinne dieser Arbeit zu realisieren.
Das zu erstellende Konzept nutzt VOMS als VO-Management System. Nach der Konzeptbeschreibung wird im Kapitel 5.5 gezeigt, wie das VOMS durch andere Systeme wie Shibboteth und CAS für das Monitoringkonzept ersetzt werden kann.

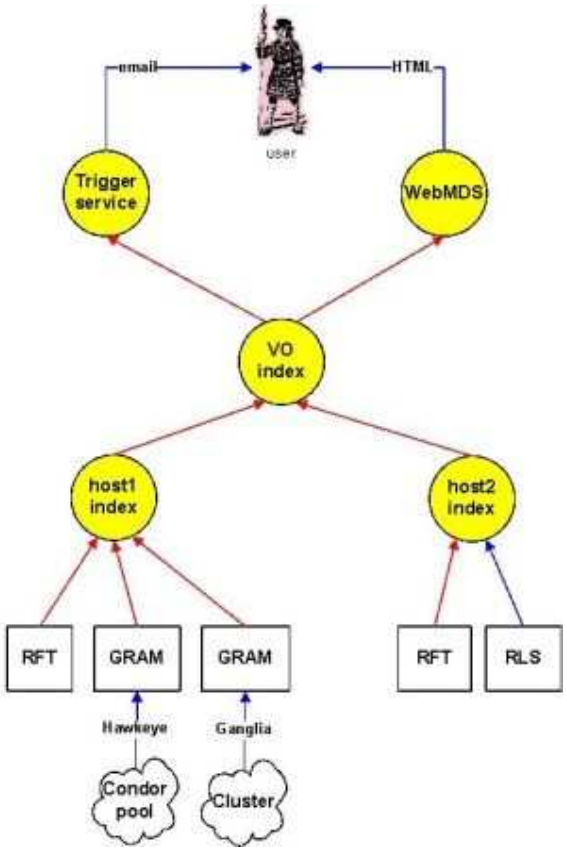


Abbildung 4.5: Bildung eines VO-Index mit MDS4 [GT4]

	CAS	GridShib/Shibboleth	VOMS/VOMRS	MDS4
Anforderung 1	✓	✓	✓	
Anforderung 2	✓	✓	✓	
Anforderung 3				✓
Anforderung 4				✓
Anforderung 5				

Tabelle 4.1: Bewertung der existierenden Tools

5 Erstellung eines VO-Monitoringkonzepts

Ziel dieses Kapitels ist die Entwicklung eines Monitoringkonzepts für virtuelle Organisationen. In den vorangegangenen Kapiteln wurden die Ziele und die Anforderungen dieses Konzepts formuliert. Einige eingesetzte Tools für den Aufbau und das Management von VOs wurden ebenso vorgestellt und bewertet. Aus dieser Bewertung kamen wir zum Schluss, dass diese Tools (alleine betrachtet) die Anforderungen für einen VO-Monitor nicht komplett erfüllen (siehe Tabelle 4.1). Es kristallisierte sich jedoch die Idee heraus, die Funktionalität dieser Tools zu kombinieren, um die erwünschten Anforderungen zu erfüllen. Diese Tools teilen sich in zwei Gruppen: Tools für das Management von VO-Mitgliedern (VOMS, CAS, Shibboleth) und Tools für das Monitoring von Ressourcen (MDS4). Eine logische Folgerung der Betrachtungen in Kapitel 4 ist, den VO-Monitor auf der gemeinsamen Basis der Funktionalitäten von VOMS und MDS4 zu konzeptionieren, da VOMS die Anforderung 1 und 2 und MDS4 die Anforderungen 3 und 4 erfüllt. Der Index Service von MDS4 stellt die Ressourceneigenschaften als XML-Daten zur Verfügung. Im ersten Teil dieses Kapitel werden daher die Struktur dieser XML-Dokumente und die Service Schnittstellen (WSDL) ausführlich beschrieben. Dabei werden Daten aus einem Beispielszenario genommen und gezeigt, wie Ressourceneigenschaften und Jobinformationen ausgewertet werden. Ebenso wird der VOMS Admin Service beschrieben.
In zweiten Teil wird ein Plugin entwickelt, der eine Verbindung zwischen den Daten aus dem Index Service und den Daten aus dem VOMS Admin herstellt. Dieses Plugin soll zur Erfüllung der letzten funktionalen Anforderung (Anforderung 5) dienen. Im letzten Teil wird eine prozess-orientierte Architektur vorgestellt, die das Konzept realisiert.

5.1 Grid Resource Monitoring Service

MDS4 basiert auf dem vorher schon beschriebenen Aggregator Framework. Im diesem Abschnitt wird die Funktionsweise des Index Services und die Struktur der Daten beschrieben, die durch diesen Dienst erschlossen werden.

5.1.1 Index Service Funktionsweise

Der Index Service ist ein Verzeichnisdienst, in dem alle Eigenschaften der registrierten Ressourcen in einer VO gesammelt werden. Er nutzt die Mechanismen des Aggregator Frameworks, um Ressourcen zu registrieren und ihre Eigenschaften und Zustände zur Verfügung zu stellen. Die Funktionsweise des Aggregator Frameworks wurde bereits in Kapitel 4.2 beschrieben. Nun wird zusammenfassend vorgestellt, wie Ressourceneigenschaften zugänglich gemacht werden und welche Komponenten bei diesem Prozess eine Rolle spielen.
Die Abbildung 5.1 zeigt den Informationsfluss bei der Publikation der Ressourceneigenschaften (Resource Properties).

Der Index Service besteht aus zwei Schnittstellen:

- Die erste Schnittstelle dient zum Hinzufügen der Resource Properties (*siehe Ziffer 1 in Abbildung 5.1*). Die Resource Properties werden durch eine Aggregator Source (Execution Source, Query Aggregator Source) verarbeitet und zu dem Index gesendet. Der Index bekommt die Resource Properties und speichert sie in einer hierarchischen Struktur als *Entries*. Ein *Entry* repräsentiert genau eine Ressource (WS-Ressource - siehe Kapitel 4.2) und enthält Informationen (Ressourceneigenschaften, Ressourceadresse, etc.) über genau diese Ressource. Aktuelle Ausprägungen von Resource Properties werden von einem *Information Provider* bereitgestellt. Der Information Provider kann ein WSRF Service wie RFT und WS-GRAM oder eine spezielle Software sein. Dieser liefert Informationen über die Ressourcen, die unter einem Grid-Dienst laufen. Beispielsweise bekommt der WS-GRAM Status-Informationen (z.B. die

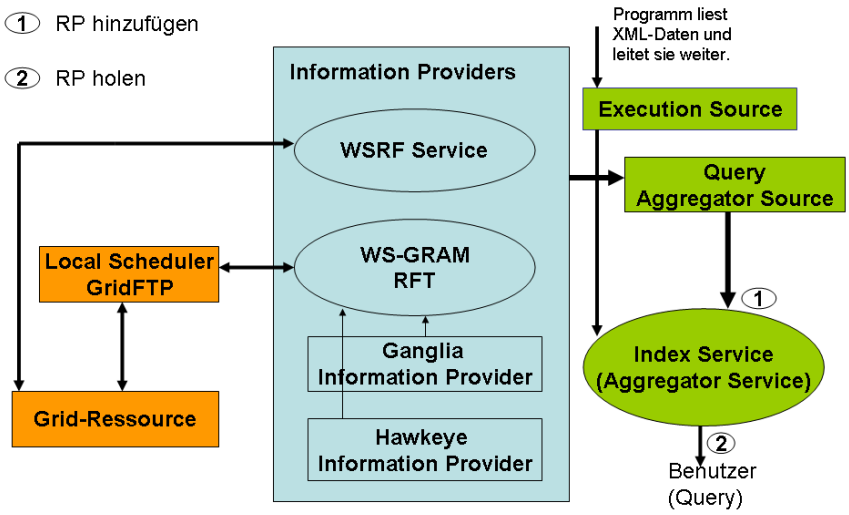


Abbildung 5.1: Informationsfluss bei der Publikation der Resource Properties zum Index Service

Anzahl laufender Jobs, Speicher-Statistik, Jobwarteschlange, Anzahl verfügbarer oder freier CPUs) vom Local Job Scheduler. Es gibt auch spezielle Information Providers *Hawkeye Information Provider, Ganglia Information Provider*)[1], die eingesetzt werden, wenn externe Job Scheduler wie *Condor*[2] [TTL 02] verwendet werden. Die Rolle und die Funktionsweise von Job Schedulern wurde bereits in Kapitel 3.3 erklärt. Die Hawkeye- und Ganglia-Information Provider nutzen meistens den WS-GRAM Service, um die Ressourceninformationen zu publizieren. Die so publizierten Informationen verwenden ein spezielles XML Schema (das *GLUE Schema*) [GLUE]. Dieses Schema ist geeignet für die Speicherung der Resource Properties, wenn sich unter einem Grid-Dienst (WS-Resource) beispielsweise ein Cluster befindet. Durch das *Computing Element (CE)* des GLUE Schemas können Job-Statusinformationen von einem beliebigen Job Scheduler gespeichert werden. Die Elemente *Cluster* und *SubCluster* ermöglichen ansonsten die Speicherung aller Informationsarten wie Hostname, Prozessorinformationen, Name des Betriebssystems und andere Informationen bezogen auf den Cluster. Mehr über das GLUE Schema ist im Anhang D zu finden.

- Die zweite Schnittstelle wird zur Abfrage und zum Holen der Resource Properties benötigt (*siehe Ziffer 2 in Abbildung 5.1*). Diese wird von jedem Benutzer verwendet, um nach Informationen über eine bestimmte Ressource zu suchen. Sie bietet außerdem folgende Operationen zur Abfrage des Index Services an:
  - `GetResourceProperty` sucht nach einer einzigen Resource Property unter Verwendung des Resource Property-Namen.
  - `GetResourceProperties` sucht nach mehreren Resource Properties.
  - `QueryResourceProperties` führt eine XPath-Abfrage [XPathv1.0] auf ein Resource Properties-Dokument aus.
  - Der Registrierungs-/ Benachrichtigungsmechanismus (*notification/subscription mechanism*) registriert eine Ressource beim Index Service und sendet neue Entries nur, wenn sich die Ressourceneigenschaften geändert haben. Bei den anderen zwei Operationen werden die Daten nach einer festgelegten Intervallzeit wiederholt gesendet und aktualisiert.

[1] `http://www.globus.org/toolkit/docs/4.0/info/providers/`
[2] `http://www.cs.wisc.edu/condor/`

Die Operation `QueryResourceProperties` wird im nächsten Abschnitt detailliert beschrieben, da sie später benötigt wird.

5.1.2 Semantik der Query API und Struktur der Daten

In diesem Abschnitt wird anhand des in Kapitel 3.3 beschriebenen Testbetts gezeigt, wie Resource Properties abgefragt werden und wie die zurückgegebenen Daten aussehen.

Da der Index Service nach der WS ServiceGroup Spezifikation [WS-SGr] konzipiert wurde, werden hier zunächst die wichtigsten Begriffe der WS ServiceGroup erklärt.

WS-ServiceGroup

Die WS-ServiceGroup Spezifikation ist ein Teil der WSRF-Spezifikation. Andere Spezifikationen, die zum WSRF gehören, sind:

- *WS-Resource* Spezifikation [WS-Res] (definiert, wie zustandbehaftete Ressourcen mittels Web Services realisiert werden)
- *WS-ResourceProperties* Spezifikation [WS-RP] (definiert, wie Ressourceneigenschaften angelegt, publiziert, modifiziert, gelöscht, entdeckt und abgefragt werden können)
- *WS-ResourceLifeTime* Spezifikation [WS-ResLt] (definiert Mechanismen, die beispielweise das Zerstören einer Ressource planmäßig, also nach einer bestimmten vorgesehenen Zeit, ermöglichen)
- *WS-Notification* Spezifikation [WSN] (definiert Modelle für die Benachrichtigung)
- *WS-BaseFauls* Spezifikation [WS-Bf] (definiert Mechanismen, die über Fehler auf WS-Ressourcen berichten).

Die WS-ServiceGroup Spezifikation beschreibt auf welche Weise WS-Ressourcen und Web Services aggregiert oder gruppiert werden können, damit ein bestimmtes Ziel erreicht wird. Das UML Diagramm in der Abbildung 5.2 stellt die wichtigsten Schnittstellen der ServiceGroup dar. Zur Begriffsklärung:

- `ServiceGroup`: Ein Web Service, der eine Gruppe von anderen Web Services oder WS-Ressourcen enthält. Beispielsweise ist der Index Service eine ServiceGroup.
- `Member`: Ein Web Service oder WS-Ressource, der Mitglied einer ServiceGroup ist.
- `ServiceGroupEntry (Entry)`: Ein XML-Element, das die Beziehung zwischen einer WS-Ressource (`Member`) und einer ServiceGroup definiert. Für jede WS-Ressource gibt es eine oder mehrere ServiceGroupEntries, die genau spezifizieren, zu welcher ServiceGroup die WS-Ressource gehört und sie enthalten zudem die Ressourceneigenschaften der WS-Ressource (repräsentiert hier als `Content`-Elemente). (Siehe Struktur im Listing 5.1). Eine ServiceGroup publiziert ihre ServiceGroupEntries. Durch die ServiceGroup können somit Clients zu den Ressourceneigenschaften gelangen.
- `ServiceGroupRegistration`: Eine Schnittstelle, die das Hinzufügen neuer Mitglieder (`Member`) in eine Gruppe (`ServiceGroup`) ermöglicht.
- `MemberEPR (Member End Point Reference)`: Die Adresse eines Web Services oder einer WS-Resource. Diese Adresse wird nach der WS-Addressing Spezifikation [WS-Addr] dargestellt.

Mehrere ServiceGroups können in einer hierarchischen Struktur so angeordnet werden, dass manche ServiceGroups wiederum als Member in ServiceGroups der höheren Ebene sind.

Der Zusammenhang zwischen ServiceGroups und Index Services ist folgender:

- Ein Index Service ist eine ServiceGroup. Es können (wie oben gezeigt) mehrere Index Services (ServiceGroups) zu einem Index Services gehören. Somit kann eine hierarchische Struktur von Index Services gebildet werden (siehe Abbildung 4.5 im Kapitel 4.2).
- Die Member des Index Services sind die WS-Ressourcen, die sich zum Index Service registriert haben.

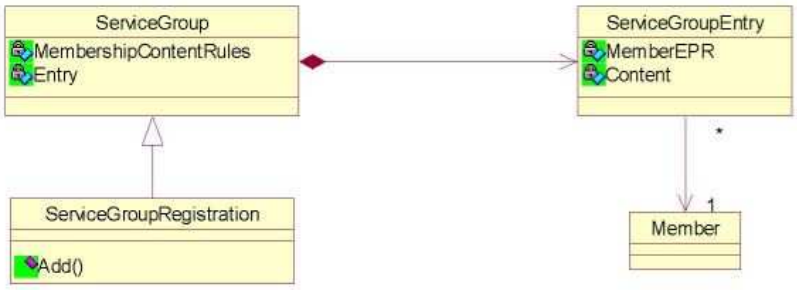


Abbildung 5.2: WS ServiceGroup UML Diagramm [WS-SGr]

- Ein Entry (Index Entry) stellt den ServiceGroupEntry des Index Services dar. Dieses Entry spezifiziert die Beziehung zwischen der WS-Ressource, dem Index Service und den Ressourceneigenschaften.

Das Listing 5.1 zeigt die Struktur eines Entry (wobei die Präfixe *wssg* und *wsa* jeweils die Namensräume (*Namespaces*) der WS-ServiceGroup/WS-Addressing Spezifikationen darstellen):

Listing 5.1: Struktur eines Entry-Elements

```
<wssg:Entry>
   <wssg:ServiceGroupEntryEPR>
      wsa:EndpointReferenceType
   </wssg:ServiceGroupEntryEPR>
   <wssg:MemberShipEPR>
      wsa:EndpointReferenceType
   </wssg:MemberShipEPR>
   <wssg:Content> {any} </wssg:Content>?
</wssg:Entry>
```

Das `<wssg:ServiceGroupEntryEPR>`-Element gibt die Adresse der ServiceGroup an. Das `<wssg:MemberShipEPR>`-Element gibt die Adresse der WS-Resource an. Das `<wssg:Content>`-Element enthält die Resource Properties. Beispiele dieses Entry-Elements sind im nächsten Abschnitt zu finden.

Semantik der Query Operation

Die Web Service Operation `QueryResourceProperties` ist eine Standard-Operation der WS-Ressource Properties-Spezifikation. Das heißt, alle Web Services, die den `ResourceProperties-PortType` implementieren, bieten automatisch diese Funktionalität. Hier wird nicht beschrieben, wie die SOAP-Nachrichten definiert sind. SOAP-Nachrichten sind eine Kapselung der realen Daten. Wie SOAP die Daten kapselt, kann in der entsprechenden Literattur [SOAPv1.2] nachgelesen werden.

Eine gesendete Nachricht (*request message*) muss folgende Struktur haben. (Das Präfix *wsrp* stellt dabei den Namensraum der WS-Resource Properties Spezifikation dar.):

Listing 5.2: Struktur des Request Message

```
<wsrp:QueryResourceProperties>
   <wsrp:QueryExpression dialect="URI">
     xsd:any
   </wsrp:QueryExpression>
</wsrp:QueryResourceProperties>
```

Das Element `</wsrp:QueryExpression>` enthält einen XPath Ausdruck (*XPath Expression*). Sein Attribut *dialect=¨URI¨* definiert, welche Version von XPath verwendet wird. In dieser Arbeit wird die Version 1.1 der XPath Spezifikation [XPathv1.0] eingesetzt. Die neueste Version XPath 2.0 [XPathv2.0] kann theoretisch zwar auch verwendet werden, Inkompatibilitätsprobleme mit dem MDS4 sind aber nicht auszuschließen. Wir verwenden in dieser Arbeit deshalb die Version 1.1.

Die zurückgegebene Nachricht (*response message*) besitzt die folgende Struktur:

Listing 5.3: Struktur des Response Message

```
<wsrp:QueryResourcePropertiesResponse>
   {any}
</wsrp:QueryResourcePropertiesResponse>
```

Diese Nachricht kann beliebige XML-Daten enthalten (angedeutet durch das `any`-Element). Es hängt von der Abfrage und der Art der Resource Properties ab. Die folgende Beispielabfrage (aus D-Grid) verdeutlicht die Konzepte noch einmal:

Die Beispielsabfrage soll einen Index Service (*https://astrogrid-mds.aip.de:8443/wsrf/services/DefaultIndexService*) durchsuchen und alle Ressourceneigenschaften holen.

Listing 5.4: Beispielsabfrage der `QueryResourceProperties`-Operation

```
<wsrp:QueryResourceProperties>
<!-----------------------------------------------------
Dieser XPath Ausdruck deutet darauf hin, dass das ganze
Resource Properties-Dokument zurückgegeben wird.
Statt der Wert "/*", kann beispielsweise durch den Wert
"count(//*[local-name()='Entry'])" nur die Anzahl aller
Entries zurückgegeben werden.
----------------------------------------------------->

   <wsrp:QueryExpression dialect="http://www.w3.org/TR/1999/REC-xpath-19991116">
     /*
   </wsrp:QueryExpression>

</wsrp:QueryResourceProperties>
```

Hier ist ein Auszug von der erhaltenen Antwort. Es handelt sich um zwei Entries. Der erste Entry liefert die Eigenschaften eines RFT-Services und der zweite liefert die Eigenschaften eines WS-GRAM Services:

Listing 5.5: Beispiele von `Entry`-Elementen

```
<wsrp:QueryResourcePropertiesResponse>
    <ns0:IndexRP>
       ........................................
   <!---------------------------------------
   Erster Entry: RFT Service Eigenschaften
    --------------------------------------->
    <ns16:Entry>
<!-------------------------------------------------------------------------
 Die Adresse der ServiceGroup. Es handelt sich hier um einen Entry des Default
     Index Services. Der Default Index Service ist also die ServiceGroup, bei
     der der RFT Service registriert ist. Die genaue Position dieses Entry
     innerhalb der ServiceGroup-Hierarchie wird durch das Element <
     ReferenceProperties> angegeben.
 ------------------------------------------------------------------------->
        <ns16:ServiceGroupEntryEPR>
            <ns30:Address>
                https://octopus-aip.gac-grid.org:8443/wsrf/services/
   DefaultIndexServiceEntry
            </ns30:Address>
        <ns31:ReferenceProperties>
            <ns17:ServiceGroupEntryKey>
                <ns18:GroupKey>22589165</ns18:GroupKey>
                <ns19:EntryKey>27035453</ns19:EntryKey>
            </ns17:ServiceGroupEntryKey>
        </ns31:ReferenceProperties>
        <ns32:ReferenceParameters/>
```

```
        </ns16:ServiceGroupEntryEPR>
         <!-- Die Adresse der WS-Ressource. Es handelt sich hier um einen RFT
    Service -->
         <ns16:MemberServiceEPR>
            <ns33:Address>
                    https://octopus-aip.gac-grid.org:8443/wsrf/services/
    ReliableFileTransferFactoryService
            </ns33:Address>
            <ns34:ReferenceProperties/>
            <ns35:ReferenceParameters/>
        </ns16:MemberServiceEPR>
    <ns16:Content xsi:type="ns20:AggregatorContent">
          <!-----------------------------------------------------------------
          Konfiguration der Aggregator Source. Hier wird eine Query Aggregator
              Source verwendet. Es wird ansonsten die Intervallzeit für die
              Aktualisierung der Resource Properties-Ausprägungen festgelegt.
          ----------------------------------------------------------------->
           <ns20:AggregatorConfig>
            <ns20:GetMultipleResourcePropertiesPollType>
                <!--Specifies that the index should refresh informationevery
     60000 milliseconds (once per minute) -->
                <ns20:PollIntervalMillis>60000</ns20:PollIntervalMillis>
                <!--specifies that all Resource Properties should be collected
    from the RFT factory -->
                <ns20:ResourcePropertyNames>rft:TotalNumberOfBytesTransferred</
    ns20:ResourcePropertyNames>
                <ns20:ResourcePropertyNames>rft:TotalNumberOfActiveTransfers</
    ns20:ResourcePropertyNames>
                <ns20:ResourcePropertyNames>rft:RFTFactoryStartTime</ns20:
    ResourcePropertyNames>
                <ns20:ResourcePropertyNames>rft:ActiveResourceInstances</ns20:
    ResourcePropertyNames>
                <ns20:ResourcePropertyNames>rft:TotalNumberOfTransfers</ns20:
    ResourcePropertyNames>
            </ns20:GetMultipleResourcePropertiesPollType>
        </ns20:AggregatorConfig>
        <!-----------------------------------------------------------------
        Ein Beispiel von Resource Properties für einen RFT Service. Liefert
            Informationen wie Anzahl von aktiven Transfers, etc.
        ----------------------------------------------------------------->
        <ns20:AggregatorData>
            <ns1:TotalNumberOfBytesTransferred>3192663</ns1:
    TotalNumberOfBytesTransferred>
            <ns2:TotalNumberOfActiveTransfers>0</ns2:TotalNumberOfActiveTransfers
    >
            <ns3:RFTFactoryStartTime>2007-04-06T17:35:34.636Z</ns3:
    RFTFactoryStartTime>
            <ns4:ActiveResourceInstances>0</ns4:ActiveResourceInstances>
            <ns5:TotalNumberOfTransfers>3</ns5:TotalNumberOfTransfers>
        </ns20:AggregatorData>
    </ns16:Content>
</ns16:Entry>
 ..............................
.......................

<!-----------------------------------------------
  Zweiter Entry: WS-GRAM Service Eigenschaften
  ----------------------------------------------->
<ns11:Entry>
   <!-------------------------------------------------------------------------
    Die Adresse der ServiceGroup. Es handelt sich hier um ein Entry des Default
```

```
        Index Services. Der Default Index Service ist also die ServiceGroup, bei
        der der WS-GRAM Service registriert ist. Die genaue Position dieses Entry
         innerhalb der ServiceGroup-Hierarchie wird durch das Element <
        ReferenceProperties> angegeben.
    ------------------------------------------------------------------------>
    <ns11:ServiceGroupEntryEPR>
        <ns24:Address>
            https://hydra.ari.uni-heidelberg.de:8443/wsrf/services/
    DefaultIndexServiceEntry
        </ns24:Address>
    <ns25:ReferenceProperties>
        <ns12:ServiceGroupEntryKey>
        <ns13:GroupKey>28649942</ns13:GroupKey>
        <ns14:EntryKey>10348889</ns14:EntryKey>
        </ns12:ServiceGroupEntryKey>
    </ns25:ReferenceProperties>
    <ns26:ReferenceParameters/>
    </ns11:ServiceGroupEntryEPR>
    <!------------------------------------------------------------------------
     Die Adresse der WS-Ressource. Es handelt sich hier um einen WS-GRAM Service
        . Das Element <ResourceID> gibt an, welcher Job Scheduler verwendet wird
        . Andere Beispiele von ResourceIDs sind "Fork", "Condor", "LSF", etc.
     ------------------------------------------------------------------------>
    <ns11:MemberServiceEPR>
        <ns27:Address>
            https://hydra.ari.uni-heidelberg.de:8443/wsrf/services/
    ManagedJobFactoryService
        </ns27:Address>
        <ns28:ReferenceProperties>
            <ns1:ResourceID>Multi</ns1:ResourceID>
        </ns28:ReferenceProperties>
        <ns29:ReferenceParameters/>
    </ns11:MemberServiceEPR>
    <ns11:Content xsi:type="ns15:AggregatorContent">
        <!------------------------------------------------------------------
        Konfiguration der Aggregator Source. Hier wird spezifiert, dass die
            abgefragten Daten das Computing Element (CE) des GLUE Schemas
            verwenden sollen. Die Resource Properties werden in diesem Beispiel
            von einem Information Provider (Ganglia) zur Verfügung gestellt.
       ------------------------------------------------------------------->
        <ns15:AggregatorConfig>
            <ns15:GetResourcePropertyPollType>
                <!-- Specifies that the index should refresh information
every 60000 milliseconds (once per minute) -->
                <ns15:PollIntervalMillis>60000</ns15:PollIntervalMillis>
                <!- specifies the resource property that should be aggregated,
    which
in this case is the GLUE clusterand scheduler information RP -->
                <ns15:ResourcePropertyName>glue:GLUECE</ns15:ResourcePropertyName
    >
            </ns15:GetResourcePropertyPollType>
        </ns15:AggregatorConfig>
        <!------------------------------------------------------------------
        Die eigentlichen Ressourceneigenschaften. Unter diesem WS-GRAM Dienst
            läuft ein Cluster, und die Eigenschaften dieses Clusters sind hier zu
             sehen.
        ------------------------------------------------------------------->
        <ns15:AggregatorData>
            <ns1:GLUECE>
                <ns1:Cluster ns1:Name="ARI-ZAH␣Hydra␣Cluster" ns1:UniqueID="ARI-
    ZAH␣Hydra␣Cluster">
```

```
                    <ns1:SubCluster ns1:Name="main" ns1:UniqueID="main">
                        <ns1:Host ns1:Name="hya04.ether" ns1:UniqueID="hya04.
    ether">
                            <ns1:Processor ns1:CacheL1="0" ns1:CacheL1D="0" ns1:
    CacheL1I="0" ns1:CacheL2="0"
ns1:ClockSpeed="2196" ns1:InstructionSet="x86"/>
                            <ns1:MainMemory ns1:RAMAvailable="1536" ns1:RAMSize="
    2025" ns1:VirtualAvailable="5762"
ns1:VirtualSize="6275"/>
                            <ns1:OperatingSystem ns1:Name="Linux" ns1:Release="
    2.6.13-15.8-smp"/>
                            <ns1:Architecture ns1:SMPSize="2"/>
                             <ns1:FileSystem ns1:AvailableSpace="66804" ns1:Name=
    "entire-system" ns1:ReadOnly="false"
ns1:Root="/" ns1:Size="70737"/>
                             <ns1:NetworkAdapter ns1:IPAddress="10.1.1.4" ns1:
    InboundIP="true" ns1:MTU="0"
ns1:Name="hya04.ether" ns1:OutboundIP="true"/>
                             <ns1:ProcessorLoad ns1:Last15Min="100" ns1:Last1Min=
    "100" ns1:Last5Min="100"/>
                        </ns1:Host>
                    </ns1:SubCluster>
                </ns1:Cluster>
                <!------------------------------------------------------------
              Das Element <Computing Element> liefert Informationen über die
                  Anzahl der Jobs, den Status der Jobs und die Jobbesitzer.
              ------------------------------------------------------------>
                <ns1:ComputingElement ns1:Name="default" ns1:UniqueID="default">
                    <ns1:Info ns1:GRAMVersion="4.0.3" ns1:LRMSType="Fork" ns1:
    LRMSVersion="1.0" ns1:TotalCPUs="2"/>
                    <ns1:State ns1:EstimatedResponseTime="0" ns1:FreeCPUs="2" ns1
    :RunningJobs="0" ns1:Status="enabled"
ns1:TotalJobs="0" ns1:WaitingJobs="0" ns1:WorstResponseTime="0"/>
                    <ns1:Policy ns1:MaxCPUTime="-1" ns1:MaxRunningJobs="-1" ns1:
    MaxTotalJobs="-1"
ns1:MaxWallClockTime="-1" ns1:Priority="0"/>
                </ns1:ComputingElement>
            </ns1:GLUECE>
        </ns15:AggregatorData>
    </ns11:Content>
</ns11:Entry>
.........................
</ns0:IndexRP>
</wsrp:QueryResourcePropertiesResponse>
```

Das Beispiel oben zeigt Daten für einen WS-GRAM Service und einen RFT Service. Für die Beschreibung ihrer Struktur siehe Kapitel 5.1.2.
Durch dieses Beispiel ist es offensichtlich, dass die Struktur der Ressourceneigenschaften stark von dem verwendeten Information Provider und der Art der WS-Ressource abhängt. Beispielsweise liefert der RFT Service Informationen wie Anzahl der Datentransfers, während der WS-GRAM Service andere Eigenschaften wie Cluster-Status, etc. liefert. Eine meist verwendete Form für die Kapselung der Daten (Resource Properties) ist das GLUE CE Schema. Die komplette Beschreibung dieses Schemas ist hier [GLUE], Anhang D zu finden. Manche Information Providers liefern detaillierte Informationen über die Jobs, falls unter dem Grid-Service Jobs laufen.
Durch die aufgeführten Beispiele haben wir gesehen, dass es durch den Index Service möglich ist, Eigenschaften wie Anzahl von Datentransfers, Anzahl laufender Jobs, Status des Cluster, etc. anzufragen. Damit ist nochmal bewiesen, dass MDS4 durch den Index Service die Anforderungen 3 und 4 erfüllt.

5.2 VO Membership Administration Service (VOMSAdmin)

In diesem Kapitel wird die VOMSAdmin-Schnittstelle vorgestellt. Es wird außerdem gezeigt, welche für VO-Monitoring-Zwecke relevanten Informationen zur Verfügung gestellt werden.

Das VOMSAdmin ist eine SOAP-Schnittstelle, die von einem VO-Manager verwendet werden kann. VOMS liefert eine Web- und eine Command Line-Applikation, die eine komfortable Nutzung dieser Schnittstelle ermöglichen. Durch VOMSAdmin kann ein VO-Manager unter anderem in einer VO Gruppen anlegen und Mitglieder in Gruppen einteilen. Der VO Manager kann auch Administrationsrechte an andere Mitglieder vergeben. Für jede Applikation, die die Schnittstelle nutzen möchte, sind hinreichende Zugriffsrechte erforderlich. Für die Entwicklung des VO-Monitors werden beispielsweise nur Leseoperationen benötigt, die nachfolgend aufgeführt sind. Die komplette WSDL-Beschreibung des VOMSAdmin befindet sich in der beigefügten CD (siehe Anhang A.1).

- *VOMSAdmin*
  - **getUser** gibt Informationen über einen Benutzer (Zertifikatsinformationen).
  - **getVOName** gibt den VO-Namen.
  - **listMembers** gibt die Liste der Mitglieder einer Gruppe in einer VO.
  - **listUsersWithRole** gibt eine Liste von Benutzern, die über eine bestimmte Rolle verfügen.
  - **listUsersWithCapability** gibt eine Liste von Benutzern, die über eine bestimmte Fähigkeit verfügen.
  - **getGroupPath** gibt den Pfad einer Gruppe (z.B „*/VO1/Mitarbeter/Leiterabteilung*“) in einer VO.
  - **listSubGroups** gibt eine Liste der Untergruppen einer Gruppe in einer VO. Für die VO im obigen Beispiel werden „*/VO1/Mitarbeter*“ und „*/VO1/Mitarbeter/Leiterabteilung*“ zurückgegeben.
  - **listGroups** gibt eine Liste aller Gruppen für einen gegebenen Benutzer in einer VO.
  - **listRoles** gibt eine Liste aller Rollen in einer VO.
  - **listRolesOfUser** gibt eine Liste aller Rollen eines Benutzers in einer VO.
  - **listCapabilities** gibt eine Liste aller Fähigkeiten in einer VO.
  - **listCapabilitiesOfUser** gibt eine Liste aller Fähigkeiten eines Benutzers in einer VO.
  - **listCAs** gibt einer Liste der Certificate Authorities (die Organisationen, die an einer VO beteiligt sind).

Mit diesen Operationen werden Informationen bereitgestellt, die Auskunft über die Struktur und Mitglieder einer VO geben und so die Anforderungen 1 und 2 erfüllen.

5.3 Integration von VOMS und Resource Monitoring Service für Monitoringszwecke

An dieser Stelle sind nun die wichtigsten Schnittstellen beschrieben, die für die Entwicklung des Monitoringkonzepts benötigt werden. Vor der eigentlichen Konzeptentwicklung werden zunächst die Daten, die von diesen Schnittstellen geliefert werden, analysiert. Aus dieser Analyse wird dann ein Konzept vorgestellt, das zur Erfüllung der Anforderung 5 führen soll. Die Anforderung 5 besagt, dass der Zusammenhang zwischen Informationen aus VOMS und denen aus dem Index Service durch den VO-Monitor erfasst werden muss. In der Einführung dieses Abschnitts wird erst gezeigt warum die Anforderung 5 derzeit nicht erfüllt ist und anschließend wird das neue Konzept vorgestellt, das die Integration der Daten aus dem VOMS und dem Index Service ermöglicht. Die Beschreibung der entwickelten VO-Monitoring-Architektur folgt in den Kapiteln 5.4 und 5.5.

5.3.1 Problemstellung

In den vorangegangenen Kapiteln (Kapitel 4, 5.1 und 5.2) wurde gezeigt, dass die existierenden Tools bereits die Anforderungen 1, 2, 3 und 4 erfüllen. Dabei haben wir auch gesehen, dass VOMS die Anforderungen 1 und 2 erfüllt, und MDS die Anforderugen 3 und 4. Das Monitoringkonzept soll dann die Funktionalität der beiden Tools kombinieren, um die ersten vier funktionalen Anforderungen zu erfüllen.
Im Folgenden wird durch ein Beispiel gezeigt, dass die Anforderung 5 im aktuellen Stand nicht erfüllt ist. Es soll vor allem gezeigt werden, dass allein die von VOMSAdmin und dem Index Service gelieferten Informationen nicht ausreichen, um die in Anforderung 5 erläuterten Beziehung und Interaktionen komplett zu erfassen.

Zur Erleichterung des Verständnisses wird anhand eines kurzen Beispiels gezeigt, welche Probleme bei der Aggregation der beiden Datenmengen auftreten können.
Die Abbildung 5.3 zeigt ein typisches Beispiel, wo Ressourcen bei zwei verschiedenen VO-Index Services registriert sind. Es gibt drei Organisationen (Organisation A aus New York, Organisation B aus Paris und Organisation C aus Garching), die Teile ihrer Ressourcen virtuellen Organisationen zur Verfügung stellen. Beispielsweise stellt die Organisation B die Ressource RB1 in der VO1 und die Ressource RB2 in der VO2 zur Verfügung. Wie in den Kapiteln 2.5 und 4 erläutert, findet die Bereitstellung einer Ressource in einer VO statt, indem der Resource Provider (also die Organisation) Ressourcenzugriffsmechanismen definiert, die Mitglieder aus dieser VO autorisieren können. Die Festlegung der Zugehörigkeit einer Ressource zu einer VO wird also von der Organisation (in diesem Beispiel von den Organisationen A, B und C) unternommen. Im Beispiel der Abbildung 5.3 werden die Ressourcen in einem VO-Index registriert, damit die VO-Mitglieder diese finden können.
Beispielsweise möchte ein Mitglied aus der VO2 den VO-Index 2 nutzen, um nach verfügbaren Ressourcen in VO2 zu suchen. Durch diesen Index erkennt er die Ressourcen RB1, RB2 und RC1. Möchte dieses Mitglied die Ressource RB1 nutzen, wird er erst bei der Autorisierung feststellen, dass er diese Ressource nicht verwenden darf, weil sich die Ressource RB1 nicht in der VO2, sondern in der VO1 befindet. In realen Situationen erfahren VO-Mitglieder im vorab vom VO-Manager oder Resource Provider, welche Ressource oder Ressourcengruppe sie nutzen dürfen.

Betrachtet man andererseits einen einfachen VO-Monitor (Abbildung 5.4), der VOs erfasst, dann kann es geschehen, dass falsche Informationen geliefert werden. Der VO-Monitor kennt nicht die genaue Position einer Ressource und verlässt sich deswegen nur auf den Index Service. Die *Position einer Ressource* definiert sich im Rahmen dieser Arbeit durch die VOs und die Gruppen, in denen die Ressource bereitgestellt wurde. Anders ausgedrückt bestimmen die Einträge in der Zugriffskontrollliste die Position einer Ressource in einer VO (siehe Kapitel 5.3.2). Im Beispiel der Abbildung 5.4 erfasst der VO-Monitor die Mitglieder und die Ressourcen der VOs (VO1 und VO2). Er ordnet aber manche Ressourcen (RB2 und RB1) der falschen VO zu. Allerdings gehört RB2 *nur* zur VO2 und RB1 *nur* zur VO1. Vor diesem Hintergrund kann man feststellen, dass die Zugriffsrechte auf die Ressourcen aus Monitoring-Sicht nicht bekannt sind. Es kann daher keine Beziehung zwischen den Ressourcen und den VO-Mitgliedern erfasst werden. Die Anforderung 5 ist somit nicht erfüllt.

Es wird hier deshalb ein Verfahren benötigt, das diese Lücke schließt. Damit ein VO-Monitor die Beziehung zwischen Ressourcen und VO-Mitgliedern herstellen und darüber hinaus die Position dieser Ressourcen herausfinden kann, müssen auch die Autorisierungsinformationen der Ressourcen zum Index Service mitgeschickt werden. Der nächste Abschnitt beschreibt diesen Ansatz im Detail.

5.3.2 Integrationskonzept

Das Grundkonzept für das Monitoring von virtuellen Organisationen ist, das MDS so zu erweitern, dass die Ressourcezugriffsinformationen (*in der Zugriffskontrollliste*) auch als Ressourceneigenschaften durch den Index Service verfügbar sind. Dieses Konzept beruht auf folgende Annahmen:

- Der Zugriff auf Ressourcen wird durch Zugriffskontrolllisten beschränkt. Eine Zugriffskontrollliste enthält die Liste der Benutzer oder Gruppen von Benutzern, die Zugriff haben.
- Eine Ressource gehört zu einer VO oder zu einer Gruppe, sobald mindestens ein Mitglied oder eine Gruppe von Mitgliedern aus dieser VO bzw. Gruppe die Ressource nutzen darf. Daher muss in der Zu-

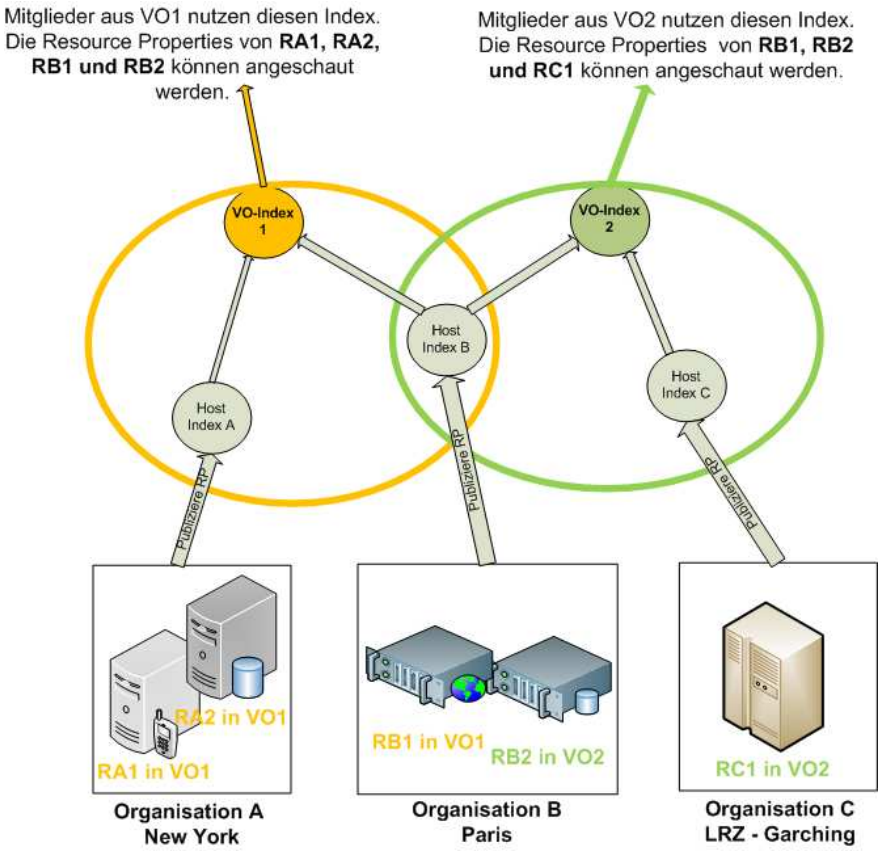


Abbildung 5.3: Bildung und Verwendung von VO-Indexen

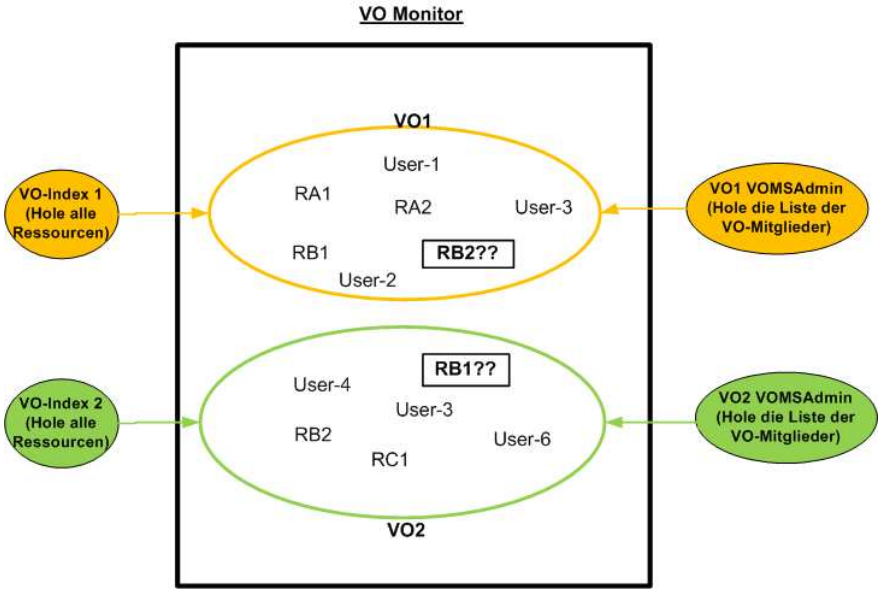


Abbildung 5.4: Einfacher VO-Monitor

griffskontrollliste mindestens ein Eintrag existieren, der auf ein VO-Mitglied verweist. Die Zugehörigkeit einer Ressource in einer VO ist also unmittelbar abhängig von den Einträgen der Zugriffskontrollliste.

- Die Zugriffskontrollliste einer Ressource kann jederzeit von dem Resource Provider geändert oder gelöscht werden. Dadurch ändert sich auch die VO-Zugehörigkeit der Ressource. Wird die Zugriffskontrollliste gelöscht, wird damit indirekt auch die Zugehörigkeit der Ressource zur VO beendet.

Aus diesen Annahmen und Feststellungen folgt, dass die Zugriffskontrollliste (ACL) die Zugehörigkeit einer Ressource zu einer VO bestimmt. Daraus ergibt sich die Tatsache, dass, wenn ein Benutzer überhaupt eine Ressource in einer VO nutzen darf, diese Ressource dann zu der gleichen Gruppe wie der Benutzer gehört (aber nicht exklusiv). Die genaue Position der Ressource kann also durch die Position ihres autorisierten Benutzers festgestellt werden. Wegen der hierarchischen Struktur der VO gehört eine Ressource der Gruppe A unmittelbar zu den Untergruppen dieser Gruppe, und kann daher von allen Mitgliedern dieser Untergruppen (bis auf besondere Restriktionen) verwendet werden.

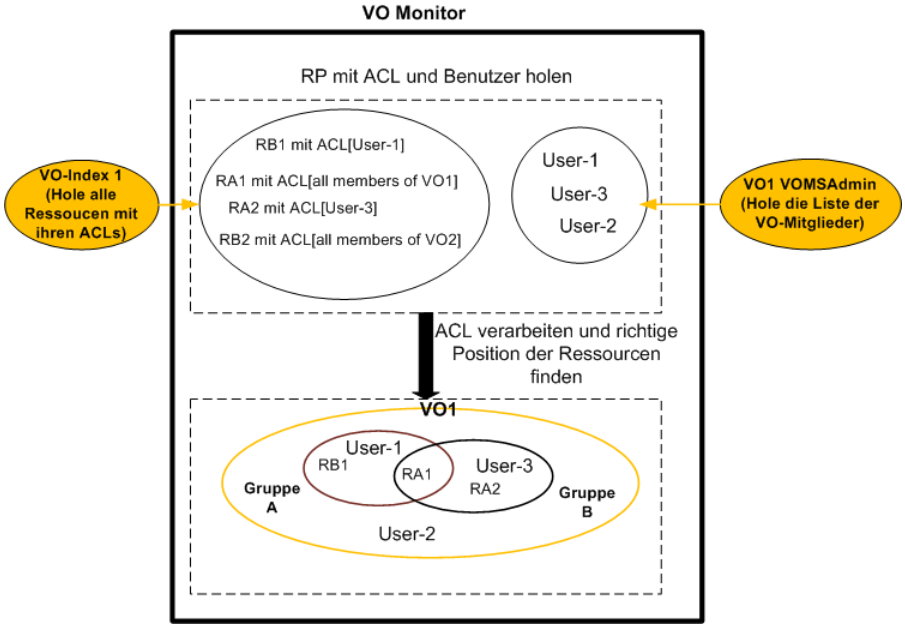


Abbildung 5.5: VO Monitoring Konzept

Diesen Annahmen zufolge kann also ein Integrationskonzept konzipiert werden, das das oben genannte Problem löst. Die Abbildung 5.5 fasst die Prinzipien dieses Konzepts zusammen.
Im ersten Schritt werden einerseits Informationen über VO-Mitglieder und VO-Mitgliederrechte (VOMS Attribute, die auch die Position der Mitglieder in der VO angeben) aus dem VOMSAdmin verwendet. Anderseits werden aus dem Index Service Informationen über die Ressourcen mit ihren Zugriffskontrolllisten genutzt. Der VO-Monitor kann dann diese Informationen genau überprüfen und die Ressourcen in der richtigen Position in der VO einordnen. In der Abbildung 5.5 gehört beispielsweise die Ressource RA1 zu den Gruppen A und B, weil sie von mindestens einem Mitglied aus diesen Gruppen verwendet werden darf. Vergleicht man dies mit dem Szenario aus der Abbildung 5.4, stellt man fest, dass der VO-Monitor nicht mehr fälschlicherweise die Ressource RB2 in der VO1 einordnet. Dieses Konzept erfüllt die Anforderung 5 (siehe Tabelle 5.1). Es sind nun alle Anforderungen erfüllt, um ein VO-Monitoringkonzept zu entwickeln.

	CAS	GridShib/Shibboleth	VOMS/VOMRS	MDS4	Integrationskonzept
Anforderung 1	✓	✓	✓		
Anforderung 2	✓	✓	✓		
Anforderung 3				✓	
Anforderung 4				✓	
Anforderung 5					✓

Tabelle 5.1: Bewertung der existierenden Tools mit Integrationskonzept

Das nächste Kapitel befasst sich mit der Realisierung des Integrationskonzepts. Das VO-Monitoringkonzept folgt in Kapitel 5.4.

5.3.3 Plugin zur Integration von VOMS und Grid Monitoring Service

In diesem Abschnitt wird eine Architektur vorgestellt, die das eben vorgestellte Integrationskonzept im Kontext der Globus Middleware realisiert.

Plugin Architektur

Das Aggregator Framework (siehe Kapitel 4.2) von MDS4 ermöglicht die Publikation von Daten im Index Service, die von einer externen Applikation erstellt wurden. Es kann also eine Applikation entwickelt werden, die diese Möglichkeit nutzt, um die Einträge aus den Zugriffskontrolllisten zu publizieren.
Abbildung 5.6 zeigt die Architektur dieser Applikation (Plugin). Zuerst muss die Applikation als Dienst bereitgestellt und im Index Service registriert werden. Der bereitgestellte Dienst wird gebraucht, um die Daten der Applikation als Entry zu kapseln und zu identifizieren. Die Applikation benötigt als Parameter eine XML-Konfigurationsdatei, die angibt, welche Zugriffskontrolllisten auf welche Ressourcen angewendet werden (*Mapping*). Die Struktur dieser Konfigurationsdatei wird im nächsten Abschnitt besprochen. Das Skript (das im Hintergrund eine Java Applikation aufruft) extrahiert aus der Konfigurationsdatei die lokale Adresse der ACL-Datei. Es öffnet anschließend die ACL-Datei, prüft den Inhalt und generiert aus den Einträgen ein XML-Dokument, das die Verbindung zwischen Ressourcen und Zugriffsinformationen in einer einfachen Form darstellt. Die Struktur dieser Datei wird ebenfalls in den nächsten Abschnitten besprochen. Schließlich enthalten die vom Index Service geholten Daten zusätzlich zu den normalen Entries auch ein Entry, das die vom Plugin generierten Daten enthält. Der VO-Monitor kann diese Daten auswerten und damit die Beziehung zwischen einer Ressource und ihren potenziellen Benutzern herstellen.
Im Folgenden sind die Ausführungsschritte des Plugins aus Abbildung 5.6 zusammengefasst:

1. Der Resource Provider schreibt eine Konfigurationsdatei. Er teilt dadurch mit, welche ACL für welche Ressource zuständig ist.
2. Das Plugin-Skript liest die Konfigurationsdatei.
3. Sobald das Skript die Verweise auf die ACLs hat, greift er auf diese Dateien zu, liest deren Inhalte und konvertiert sie.
4. Die Execution Source liest die konvertierten XML-Daten vom Skript, findet heraus, in welcher ServiceGroup der Plugin registriert ist, und legt einen neuen Entry in dieser ServiceGroup an.
5. Die Execution Source publiziert den neu erstellten Entry zum Index Service.
6. Jedes Mitglied oder jede Applikation in der VO kann den Index Service abfragen und dadurch erfahren, welche Zugriffsrechte für welche Ressourcen festgelegt wurden.

Struktur der Konfigurationsdatei

Die Konfigurationsdatei ist eine XML-Datei, die eine feste Struktur hat. Diese Datei muss von dem Resource Provider verwaltet werden. Die komplette Struktur in XSD (XML Schema Definition) [XSD] ist in der beigefügten CD zu finden (siehe Anhang A.1). Hier soll nur ein Auszug gezeigt werden.

Bevor auf die Syntaxbeschreibung eingegangen wird, zunächst eine kurze Vorstellung der verwendeten Nomenklatur:
`<Ein XML Tag>?` bedeutet, dass das Element maximal einmal oder gar nicht vorkommen kann.
`<Ein XML Tag>+` bedeutet, dass das Element mindestens einmal vorkommen muss.
`<Ein XML-Tag>*` bedeutet, dass das Element beliebig oft auftritt.
`<Ein XML-Tag EinAttribut=¨Wert1|Wert2|Wert3¨>`: Das Zeichen `'|'` steht hier für den logischen Operator „`oder`“ .

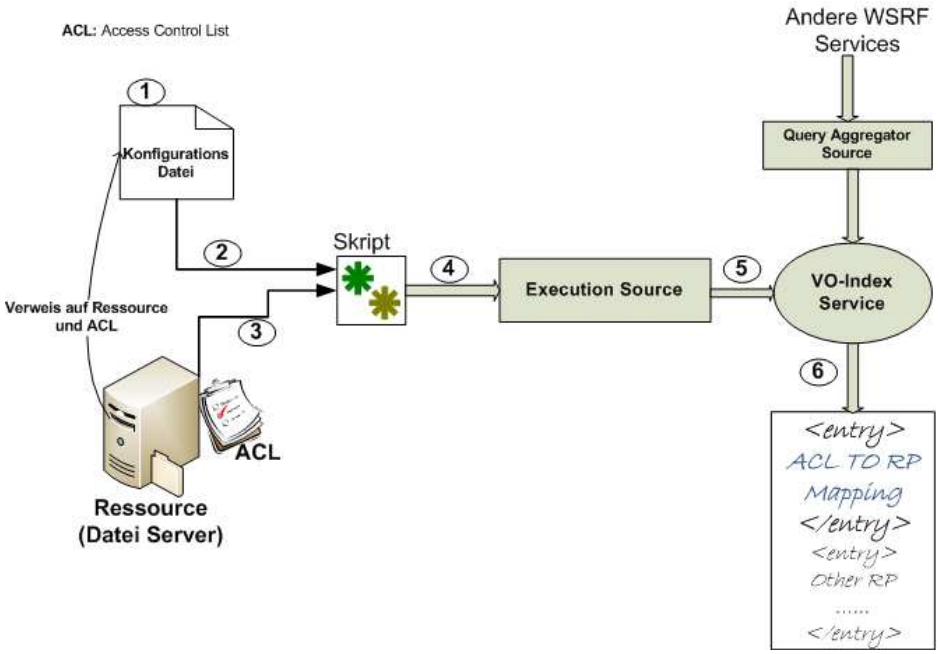


Abbildung 5.6: Plugin Architektur

Wenn vor dem Element nichts steht, bedeutet dies, dass das Element nur einmal vorkommen muss, natürlich solange das Elternelement existiert.

Listing 5.6: XML Schema der Konfigurationsdatei

```
<voresourcesConfiguration>
   <voresourceEntry>+
      <resource host="host-URL" serviceType="GRAM-Multi|GRAM-Fork|GRAM-PBS|GRAM-
          Condor|
GRAM-Loadleveler|GRAM-SGE|GRAM-NQS|RFT|Service-URI"␣/>
      <gridmap-users>?
         <gridmap-user value="unique␣subject-name"/>*
         <gridmap-file value="absolute␣path␣to␣file"/>*
      </gridmap-users>
      <voms-attributes>?
         <voms-attribute value="voms-attribute"/>*
         <vomsAttrAuthzFile value="absolute␣path␣to␣file"/>*
      </voms-attributes>
   </voresourceEntry>
<voresourcesConfiguration>
```

Der Zweck dieser Konfigurationsdatei ist, für eine gegebene Ressource alle Autorisierungsinformationen oder einen Verweis darauf anzugeben. Diese Informationen werden später ausgewertet, um herauszufinden, in welcher VO sich eine Ressource befindet. Zur Syntaxbeschreibung:

- Das Element `<voresourcesConfiguration>` enthält alle Einträge. Es existiert maximal einmal.
- Das Element `<voresourceEntry>` enthält genau eine Ressource (siehe `<resource>`-Element) und keine oder mehrere Autorisierungsinformationen. Falls unter einem Host mehrere Grid-Dienste (WS-Ressourcen) angeboten werden, dann kann dieses Element mehrmals vorkommen.
- Das Element `<resource>` gibt eine WS-Ressource an. Es muss genau einmal im Element `<voresourceEntry>` vorkommen. Das Attribut `host` gibt den Host (Beispiel: *grid3.nm.ifi.lmu.de* oder *http://grid3.nm.ifi.lmu.de:8443*) an, auf dem die WS-Ressource bereitgestellt wurde. Das Attribut `ServiceType` gibt an, welche WS-Resource auf diesem Host verfügbar ist. Beispielweise bedeutet GRAM-Condor, dass es sich um einen WS-GRAM Service handelt und dass der darunter laufende Job

Scheduler der Condor-Scheduler ist. Abhängig von den Schedulern, die zur Verfügung stehen, kann die `ServiceType`-Liste erweitert werden. Durch diese beiden Attribute ist die Eindeutigkeit einer Ressource global garantiert.

- Das Element `<gridmap-users>` enthält eine Liste der Benutzer (`<gridmap-user>`), die autorisierten Zugriff auf die Ressource (`<resource>`) haben. Es enthält außerdem eine Liste von Gridmap-Files. Gridmap-Files enthalten eine Liste von Benutzern.
- Das Element `<gridmap-user>` enthält genau einen Benutzer (repräsentiert durch seinen X.509-Zertifikat Distinguished Name).
- Das Element `<gridmap-file>` enthält einen Verweis auf genau eine Gridmap-File. Diese enthält eine Liste von Distinguished Names.
- Das Element `<voms-attributes>` enthält VOMS-Attribute. Wie in Kapitel 2.5 beschrieben, repräsentieren VOMS-Attribute eine Gruppe von VO-Mitgliedern.
- Durch das Element `<voms-attribute>` wird genau ein VOMS-Attribut angegeben.
- Das Element `<vomsAttrAuthzFile>` enthält einen Verweis auf eine VOMS-Attribute-Datei, die eine Liste von VOMS-Attributen enthält. Diese VOMS-Attribute repräsentieren alle Mitglieder, die für den Ressourcenzugriff autorisiert sind.

Das Plugin unterstützt damit zwei Arten von Zugriffskontrolllisten, nämlich Gridmap-Files und VOMS-Attribute Dateien.

Struktur der erstellten Daten

Das Plugin bekommt eine Konfigurationsdatei als Parameter, validiert diese Datei gegen das XML Schema (siehe Anhang A.1) und generiert eine neue XML-Datei mit der folgenden Struktur:

```
<voresources>
  <voresource>+
      <resource host="host-URL" serviceType="GRAM-Multi|GRAM-Fork|GRAM-PBS|GRAM-
          Condor|GRAM-Loadleveler
|GRAM-SGE|GRAM-NQS|RFT|Service-URI"␣/>
      <gridmap-users>?
            <gridmap-user>subject-name<gridmap-user>*
      </gridmap-users>
      <voms-attributes>?
            <voms-attribute>voms-attribute</voms-attribute>*
      </voms-attributes>
   </voresource>
<voresources>
```

Die Strutur der generierten Datei unterscheidet sich nur wenig von der Konfigurationsdatei. Die Inhalte der im Element `<gridmap-file>` angegebenen Datei werden extrahiert und jeweils als `<gridmap-user>` in die neue Datei geschrieben. Ebenso werden die Inhalte von der in `<vomsAttrAuthzFile>` angegebenen Datei extrahiert und als `<voms-attribute>`-Elemente in die neue XML-Datei geschrieben.
Das generierte XML-Dokument wird dann von einem VO-Monitor verwendet, um die Beziehung zwischen einer Ressource und ihren potenziellen Benutzern herzustellen.

Beispiel

In diesem Abschnitt sollen anhand des in Kapitel 3.3 vorgestellten Testbetts einige Beispieldaten gezeigt werden. In diesem Fall wurde das Plugin nachträglich installiert.
In der Testbettbeschreibung haben wir gesehen, dass ein RFT Service auf dem Rechner Grid3 bereitgestellt ist. Dieser RFT Service wird für die Verwaltung von GridFTP-Transfers benötigt. Es wird zusätzlich durch

Zugriffskontrolllisten festgelegt, wer diesen Dienst aufrufen darf. Da wir im Testbett das VOMS als VO Management System verwenden, werden daher VOMS-Attribute-Dateien für die Zugriffskontrolle eingesetzt. Es können außerdem Gridmap-Files eingesetzt werden, weil der VOMS PDP (Policy Decision Point) auch X.509-Zertifikaten Subject Name erkennt.

Beispiele von Zugriffskontrolldateien sind:

- Inhalt des Gridmap-Files

```
"/C=DE/O=GridGermany/OU=Leibniz-Rechenzentrum/CN=Herve␣Amikem␣Chemeza"
    amikem
"/C=DE/O=GridGermany/OU=Leibniz-Rechenzentrum/CN=Michael␣Schiffers" schiffer
```

- Inhalt des vomsAttribute-Files

```
"/VO1/LRZ/MNM/BueroA/Role="Administration"/Capability=NULL"
"/VO1/LRZ/MNM/BueroB/Role="Management"/Capability=NULL"
```

Die Konfigurationsdatei wird für den RFT Service so aussehen:

```
<voresourcesConfiguration>
   <voresourceEntry>
      <resource host="https://grid3.nm.ifi.lmu.de:8443" serviceType="RFT" />
      <gridmap-users>
         <gridmap-file value="grid-mapfile"/>
      </gridmap-users>
      <voms-attributes>
         <vomsAttrAuthzFile value="vomsAttributeFile"/>
      </voms-attributes>
   </voresourceEntry>
<voresourcesConfiguration>
```

Das Plugin-Skript bekommt diese Konfigurationsdatei als Parameter und generiert daraus die folgende Datei:

```
<voresources>
   <voresource>
      <resource host="https://grid3.nm.ifi.lmu.de:8443" serviceType="RFT" />
      <gridmap-users>
         <gridmap-user>/C=DE/O=GridGermany/OU=Leibniz-Rechenzentrum/CN=Herve
             Amikem Chemeza</gridmap-user>
         <gridmap-user>/C=DE/O=GridGermany/OU=Leibniz-Rechenzentrum/CN=Michael
             Schiffers</gridmap-user>
      </gridmap-users>
      <voms-attributes>
         <voms-attribute>/VO1/LRZ/MNM/BueroA/Role="Administration"/Capability=
             NULL
</voms-attribute>
         <voms-attribute>/VO1/LRZ/MNM/BueroB/Role="Management"/Capability=NULL
</voms-attribute>
      </voms-attributes>
   </voresource>
<voresources>
```

Die vom Skript generierten Daten werden daraufhin von der Execution Source zum Index Service publiziert, dabei in einem einzigen Entry gekapselt, mit welchem der VO-Monitor die Beziehungen zwischen Ressourcen und VO-Mitgliedern herstellen kann. Ein Beispiel-Entry sieht so aus:

```
<ns16:Entry>
      <ns16:ServiceGroupEntryEPR>
         <ns30:Address>
```

```
            https://grid3.nm.ifi.lmu.de:8443/wsrf/services/DefaultIndexService
         </ns30:Address>
      <ns31:ReferenceProperties/>
      <ns32:ReferenceParameters/>
      </ns16:ServiceGroupEntryEPR>
       <ns16:MemberServiceEPR>
         <ns33:Address>
               https://grid3.nm.ifi.lmu.de:8443/wsrf/services/
                   VOResourceMappingService
         </ns33:Address>
         <ns34:ReferenceProperties/>
         <ns35:ReferenceParameters/>
      </ns16:MemberServiceEPR>
   <ns16:Content xsi:type="ns20:AggregatorContent">
        <ns20:AggregatorConfig/>
      <ns20:AggregatorData>
                <!-- Hier befinden sich genau die vom Plugin generierten XML-
                    Daten. -->
         <voresources>
            <voresource>
               <resource host="grid3.nm.ifi.lmu.de:8443" serviceType="RFT" />
                  <gridmap-users>
                  <gridmap-user>/C=DE/O=GridGermany/OU=Leibniz-Rechenzentrum/CN=
                      Herve Amikem Chemeza</gridmap-user> <gridmap-user>/C=DE/O=
                      GridGermany/OU=Leibniz-Rechenzentrum/CN=Michael Schiffers</
                      gridmap-user>
                  </gridmap-users>
                  <voms-attributes>
                     <voms-attribute>/VO1/LRZ/MNM/BueroA/Role="Administration"/
                         Capability=NULL</voms-attribute>
                     <voms-attribute>/VO1/LRZ/MNM/BueroB/Role="Management"/
                         Capability=NULL</voms-attribute>
                  </voms-attributes>
            </voresource>
         <voresources>
      </ns20:AggregatorData>
   </ns16:Content>
</ns16:Entry>
```

In diesem Entry sind die Adresse der Ressource (durch das `<ns16:MemberServiceEPR>`-Element) und die Adresse des Index Services (durch das `<ns16:ServiceGroupEntryEPR>`-Element), bei der sich das Plugin registriert hat, dargestellt. Das Element `<ns20:AggregatorData>` enthält das vom Plugin generierte XML-Dokument. Dieses Dokument zeigt dass auf dem Grid3 (*grid3.nm.ifi.lmu.de*) ein RFT Service bereitgestellt ist. Es zeigt auch dass die Benutzer („*Herve Amikem*“ und „*Schiffers Michael*“) und alle Mitglieder der VO-Gruppe „*/VO1/LRZ/MNM/BueroA*“ mit der Rolle „*Administrator*“ den Zugriff auf den bereitgestellten RFT-Dienst haben. Genau diese Information möchte der VO-Monitor wissen, um die richtige Position der Ressourcen in der VO zu bestimmen.

5.4 Prozess-orientiertes VO-Monitoring

Vor diesem Hintergrund ist nun das Ziel dieses Abschnittes die Entwicklung eines Konzepts für VO-Monitoring. In den vorangegangenen Kapiteln wurden bereits erste Ideen zum Konzept dargestellt. Es wurde gezeigt, dass aus dem VOMSAdmin und dem Index Service genügend Informationen extrahiert werden können, um die Anforderungen eines VO-Monitors zu erfüllen. Das zu entwickelnde Konzept folgt einer *Prozess-orientierten Architektur (POA)*. Eine solche Architektur ermöglicht die Bildung von Prozessketten aus verfügbaren Diensten, was eine flexiblere Ausrichtung der IT an den Geschäftsprozessen erlaubt. Der so gebildete Prozess kann im Falle des VO-Monitorings alle verfügbaren Daten aus den jeweiligen Diensten extrahieren und

sie derart verarbeiten und aggregieren, dass am Ende alle Informationen über eine VO durch einen einzigen Dienst zu erreichen sind. Bevor die eigentlichen Konzepte genauer beschrieben werden, folgt zunächst eine kurze Erläuterung der Grundkonzepte der POA. Die Modellierung des Konzepts erfolgt durch UML *(Unified Modeling Language)* [UMLv1.4.2].

5.4.1 Einführung in die Prozess-orientierte Architektur (POA)

Die Flexibilität, die Unternehmen benötigen, hängt wesentlich von ihren DV-Systemen ab. Bei der Ausrichtung einer IT-Strategie auf aktuelle Herausforderungen bietet eine POA eine Reihe von Vorteilen. Unter anderem ermöglicht sie die Unabhängigkeit von Technologien, Plattformen, Systemen und Sprachen und damit eine leichtere und flexiblere Integration von hoch effizienten Systemen innerhalb eines Unternehmens oder organisationsübergreifend. Prozess-orientierte Architekturen ermöglichen die Komposition von Prozessen, um die Ziele von Geschäftsprozessen zu unterstützen.
Abbildung 5.7 zeigt ein Beispiel für eine POA. Auf der linken Seite ist eine dreisichtige J2EE-Laufzeitumgebung zu sehen, im rechten Bereich befindet sich ein Legacy-System. Zentral ist ein Prozess-Management-System. In einem Prozess-Management-System werden Prozesse instanziiert und ausgeführt. Abbildung 5.7 zeigt außerdem, dass die Kommunikation zwischen den beiden Systemen auf drei verschiedene Weisen stattfinden kann. In der untersten Schicht kann dies über die Datenbank oder ein Messagingsystem geschehen, in der Mittelschicht bieten sich J2EE-Standards an. Um eine hohe Flexibilität und Unabhängigkeit zu erreichen, ist es aber sinnvoll, Funktionen als Dienste zu entkoppeln. Das findet in der obersten Schicht (Activity Layer) statt, in der Prozesse zur Realisierung bestimmter Aufgaben eingesetzt werden können. POA ist Teil einer SOA (Service Oriented Architecture), solange der Zugang zu den Systemen über Web Services stattfindet.

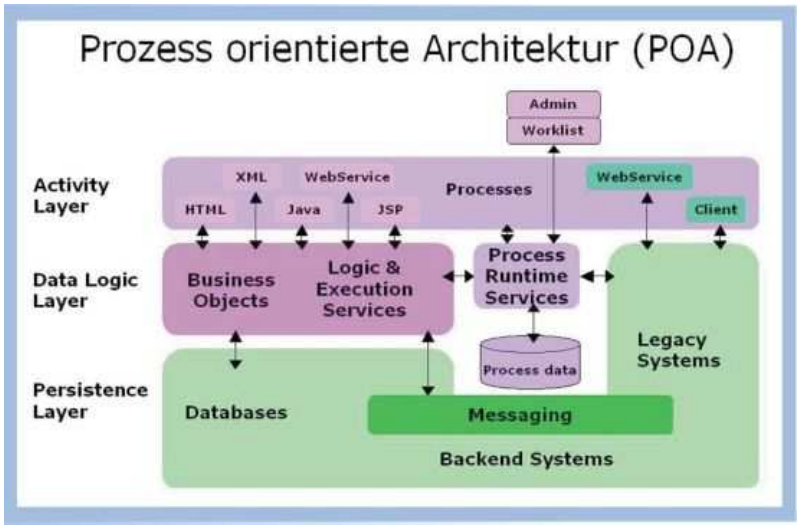


Abbildung 5.7: Beispiel einer prozess-orientierten Architektur [Fran 03]

Das wichtigste POA-Konzept ist der Prozess (*Business Process*). Ein Prozess ist eine Aktivität oder eine Komposition von Aktivitäten. Die Aufgabe eines Prozesses kann beispielsweise eine Kombination von Web Services für Transaktionen, Applikationen oder administrativen Tätigkeiten sein. Die Web Services können weltweit auf verschiedene Orte verteilt oder selbst Prozessen sein. Zu den wichtigsten Anforderungen an einen Prozess gehört die einfache Modellierbarkeit. Es müssen Modellierungswerkzeuge existieren, die es z.B. einem Projektmanager, ohne tiefere Kenntnisse über die vorhandenen Systeme, ermöglichen, die gewünschten Geschäftsprozesse zu definieren. Prozesse müssen auch in der Lage sein, die zugrunde liegende Funktionalität eines Systems zu kapseln. Wenn ein Prozess mit einem oder mehreren *DBMS (Database Management System)* kommuniziert, so muss es auch aus der Prozesssicht Mechanismen für Transaktionen, Benutzersynchronisation und Recovery geben. Da Prozesse hier Web Services nutzen, muss es wegen dieser losen Kopplung Mechanismen zur Fehlerbehandlung geben.
Zusammengefasst ermöglichen Prozesse das automatische Ausführen von Aktivitäten über mehrere Standorte

durch die Nutzung von Standards.
Prozesse können wie folgt komponiert werden:

- *Orchestrierung*. Bei der Orchestrierung werden Dienste mit Hilfe einer Prozessbeschreibungssprache angeordnet und deren Ablauf genau festgelegt. Innerhalb des ausführbaren Geschäftsprozesses wird vorab geplant, wann und unter welchen Bedingungen ein Web Service aufgerufen wird. Bei den Web Services kann es sich sowohl um interne (firmeneigene) als auch externe (eventuell von Geschäftspartnern verwaltete) handeln. Die Orchestrierung sieht eine kontrollierende Instanz (Koordinator) vor, die den gesamten Prozessablauf, also die Aktivitäten, die zur Zielerreichung notwendig sind, überwacht. Somit entspricht die Orchestrierung eher den in Workflow-Systemen üblichen Prozessabläufen. Für die Orchestrierung von Web Services kann WS-BPEL (Web Services Business Process Execution Language) [BPELv2.0] verwendet werden. Mehr über WS-BPEL ist im Anhang B zu finden.

- *Choreographie*. Im Gegensatz zur Orchestrierung bezieht sich die Choreographie auf eine verteilte Art der Aufgabenbearbeitung. Hierbei wird die Abfolge der Nachrichten von allen Seiten verfolgt und keine Seite ist für die Koordination des Prozessablaufs zuständig, sondern jede angesprochene Instanz ist für den weiteren Verlauf der Bearbeitung zur Zielerreichung verantwortlich. Die Aufgabe der Choreographie besteht somit darin, den Ablauf zwischen Client und Web Service zu regeln. Bei dieser Kommunikation spielt es keine Rolle, ob der Web Service zur Zielerreichung selbst als Client in Erscheinung tritt oder nicht. Für die Choreographie werden Sprachen wie *WS-CDL (Web Services Choreography Description Language)* [WS-CDL] eingesetzt.

- *Kollaboration*: Hier wird der Prozess als eine Kollaboration zwischen den Geschäftspartnern (Business Partner) betrachtet.

In dieser Arbeit werden Grid Monitoring Dienste und VOMS Dienste orchestriert, um alle benötigten Daten bereitzustellen. Dafür ist die Prozessbeschreibungsprache WS-BPEL gut geeignet. WS-BPEL betrachtet den VO-Monitor als einen Prozess, der ausgeführt wird und dabei Zwischenzustände in Variablen speichert, Daten aus dem VOMSAdmin und dem VO-Index holt, sie verarbeitet und zur Verfügung stellt.
Das zu entwickelnde Konzept nutzt VOMS als VO Management System. Wegen der Vorteile der POA wie lose Kopplung können andere VO Management Systeme wie CAS und GridShib eingesetzt werden. Wie so eine Anpassung stattfindet, wird in Kapitel 5.6 gezeigt.

5.4.2 Modellierung von Business Prozessen (Workflow)

Die Modellierung von Businessprozessen erfordert eine Diskussion zweier Aspekte: das Business Anwendungsfall Modell (*business use case model*) und das Business Objekt Modell (*business object model*). [TUT1]

- Ein Business Anwendungsfall beschreibt Prozesse. Diese Prozesse werden als eine Menge von Aktionen betrachtet, die den Geschäftsteilnehmern (Geschäftspartner, business actor) sichtbare Werte zeigen, um eine Interaktion zu ermöglichen. Ein Business Anwendungsfall kann mit UML Anwendungsfall--Diagrammen (*UML use case diagram*) modelliert werden. Anwendungsfall-Diagramme, die auch während der Anfoderungsanalyse verwendet werden können, beschreiben die Funktionalitäten (von außen betrachtet) eines Systems. Sie beschreiben also im Falle eines Prozesses wie die Akteure und das System (Prozess) interagieren, um die Geschäftsziele zu erreichen. Das Business Anwendungsfall Modell liefert einen groben Überblick aus Geschäftsteilnehmer-Sicht.

- Während ein Business Anwendungsfall Modell beschreibt, was ein Prozess macht, zeigt das Business Objekt Modell, wie der Geschäftsprozess ausgeführt wird. Es stellt detailliert dar, wie die Geschäftspartner mit dem Prozess interagieren, um die Geschäftsziele zu erreichen. Ein UML Aktivitätsdiagramm (*UML activity diagram*) kann hier eingesetzt werden. Ein Aktivitätsdiagramm beschreibt den Workflow des Geschäftsprozesses. Es stellt einerseits dar, welche Aktivitäten ausgeführt werden und andererseits, welche Geschäftspartner welche Aktivitäten ausführen.

Im folgenden Abschnitt wird anhand des Business Modells das VO-Monitoringkonzept vorgestellt.

5.4.3 Das VO-Monitoringkonzept

Ziel dieses Abschnittes ist die detaillierte Beschreibung des VO-Monitoringkonzepts dieser Arbeit.
In der ersten Hälfte dieses Kapitel wurden die wichtigsten Komponenten des VO-Monitors vorgestellt. Es wurde beschrieben, wie die Komponenten funktionieren und welche Art von Daten aus diesen Komponenten abgerufen werden können. Es wurde auch eine Integration zwischen den Daten aus den jeweiligen Komponenten konzipiert und realisiert. Die Integration sollte die Entwicklung eines VO-Monitoringkonzepts ermöglichen. Das VO-Monitoring instanziiert eine POA. Das ist möglich, da alle Komponenten (VOMSAdmin und Index Service) Web Services Schnittstellen anbieten. Die Definition und die Vorteile der POA wurden bereits erläutert. Anhand der im vorherigen Abschnitt formulierten Beschreibungsmittel werden nun die einzelnen VO-Monitoringkonzepte diskutiert.

Das VO-Monitoring Anwendungsfall Modell

Das in Abbildung 5.8 dargestellte Anwendungsfall-Diagramm zeigt das VO-Monitoringkonzept. Das Konzept besteht aus zwei Systemen: das VO-Monitoring System (oder der VO-Monitor) und das Frontend. Der VO-Monitor ist ein Prozess, der mit den wichtigsten Akteuren (Web Diensten) kommuniziert, Daten über die Inhalte der VO sammelt und sie zur Verfügung stellt. Das Frontend repräsentiert eine Benutzerapplikation, die Monitoringdaten aus einer Datenbank liest, um sie in einfacherer lesbarer Form darzustellen. Das Frontend kann eine Web-Applikation oder eine graphische Applikation sein. Im Folgenden werden die verschiedenen Akteure und die Anwendungsfälle näher beschrieben.

Akteure:

- *VOMSAdmin Service*: Siehe Kapitel 5.2
- *Index Service*: Siehe Kapitel 5.1
- *Datenbank Service*: Ein Web Service, der alle Operationen für die Speicherung sämtlicher VO-Daten wie Ressourcen, Mitglieder, Gruppen, Jobs, etc anbietet. In Kapitel 6.2 wird dieser Dienst näher beschrieben.
- *Client*: Ein beliebiger Benutzer, der den VO-Monitor nutzen kann, um den Lebenszyklus einer bestimmten VO zu betrachten. Die gelesenen Daten werden aus einer Datenbank geholt, welche das einzige Bindeglied zwischen dem Frontend und dem VO-Monitor ist.

Anwendungsfälle:

- *Schreiben in die Datenbank*: Dieser Anwendungsfall kapselt eine Menge von Operationen, die Zugriff auf eine entfernte Datenbank ermöglichen. Alle anderen Anwendungsfälle nutzen diesen Anwendungsfall, um ihre Daten in der Datenbank zu speichern.
- *Hole VO-Struktur und VO-Mitglieder*: Hier werden aus dem VOMSAdmin alle Informationen bezüglich der Mitglieder und der Gruppen in der VO erfasst. Dies alles wird direkt in der Datenbank durch den Datenbank Service gespeichert, und somit sind die Daten dann sofort von dem Client abrufbar, d.h. die VO-Daten aus dem Monitoring System sind zeitnah abrufbar.
- *Hole Ressourceneigenschaften*: Aus dem Index Service werden aktuelle Informationen über Ressourcen geholt (siehe Kapitel 5.1). Diese Informationen enthalten auch die Plugin-Daten, die eine Verbindung zwischen den Ressourcen und den VO-Mitgliedern herstellen soll. Siehe Kapitel 5.3.
- *Hole Plugin Informationen*: In den aus dem Index Service abgerufenen Daten sind auch Plugin Informationen vorhanden (siehe Kapitel 5.3).
- *Finde Beziehung Benutzer-Ressource* ist der eigentliche Anwendungsfall, der sich um die Herstellung der Beziehung zwischen Ressourcen und VO-Mitgliedern kümmert. Er verwendet einen speziellen Algorithmus für diesen Zweck. Mehr über den Algorithmus ist in Kapitel 6.2 zu finden.
- *Fehler Behandlung*: Der Fehlerbehandlung-Mechanismus gehört zu den Konzepten der POA. Falls ein Fehler (z.B. ein Web Service wird unverfügbar) im System auftritt, soll eine Fehlerbehandlungsroutine gestartet werden. Diese Routine protokolliert den Fehler in der Datenbank (durch den Datenbank Service) und stoppt den Monitor-Prozess, je nach Schwierigkeitsgrad des Fehlers.

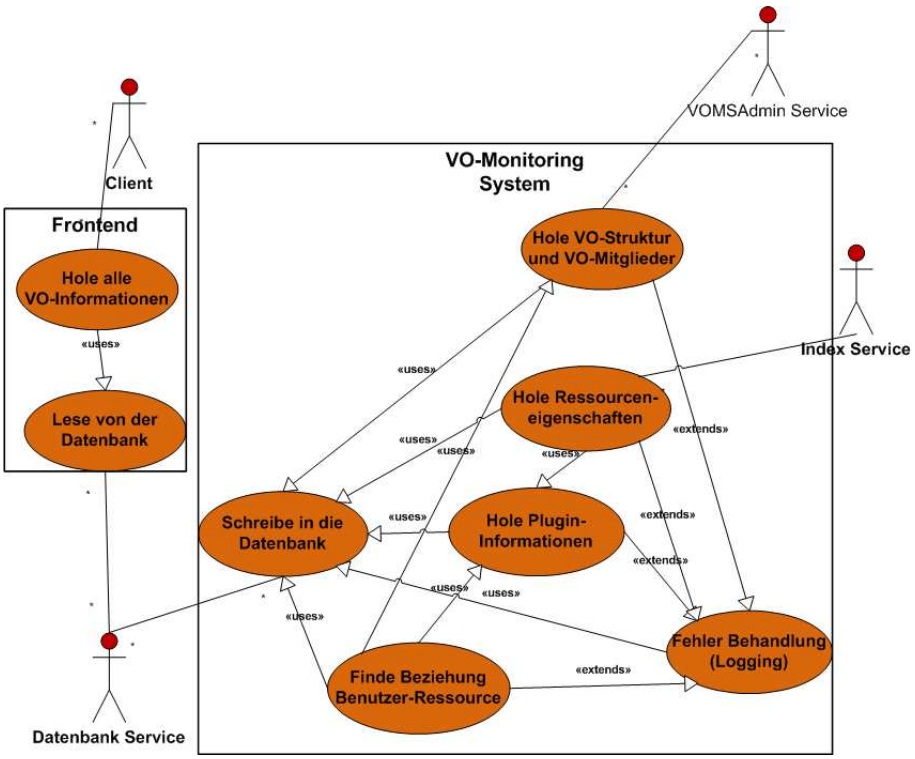


Abbildung 5.8: VO Monitoring Anwendungsfall-Diagramm

Das VO-Monitoring Objekt Modell

Das Business Aktivitätsdiagramm in Abbildung 5.9 beschreibt im Detail die verschiedenen Aktivitäten des VO-Monitors. Das Frontend wird hier nicht beschrieben, da seine Aktivitäten trivial sind. Das Diagramm zeigt zusätzlich, welche Akteure in welche Aktivitäten involviert sind. Beispielsweise ist der Datenbank Service bei jedem Datenbankzugriff angesprochen.
Der VO-Monitor wird als zyklischer Prozess realisiert. Nach jedem Zyklus wartet er eine vordefinierte Zeit, bevor er erneut startet.
Zusammengefasst werden die Aktivitäten in drei Gruppen klassifiziert:

- Die Aktivitäten in der ersten Gruppe befassen sich mit dem VOMSAdmin Service und der Datenbank. Es werden alle VO-Mitglieder erfasst und in der Datenbank gespeichert. Dabei müssen auch die Gruppen und die Rollen dieser Mitglieder miterfasst werden.

- Die zweite Gruppe der Aktivitäten befasst sich mit dem Index Service und der Datenbank. Alle verfügbaren VO-Indexe werden abgefragt, die Ressourcen mit ihren Eigenschaften erfasst und in die Datenbank geschrieben.

- Da der Monitor noch evaluieren soll, zu welcher VO die erfassten Ressourcen eigentlich gehören (siehe Kapitel 5.3), müssen noch die in den anderen zwei Aktivitätsgruppen erfassten Daten analysiert werden. Die Analyse nutzt die durch das Plugin publizierten Daten, um die Beziehung zwischen VOs und Ressourcen herzustellen. Nach diesem Schritt werden die Daten in der Datenbank aktualisiert und der Zyklus beendet. Nach Beenden des Zyklus sind die Daten aktuell. Ein Benutzer kann dann durch das Frontend auf diese Daten zugreifen und ansehen, welche Benutzer und Ressourcen in der VO vorhanden sind.

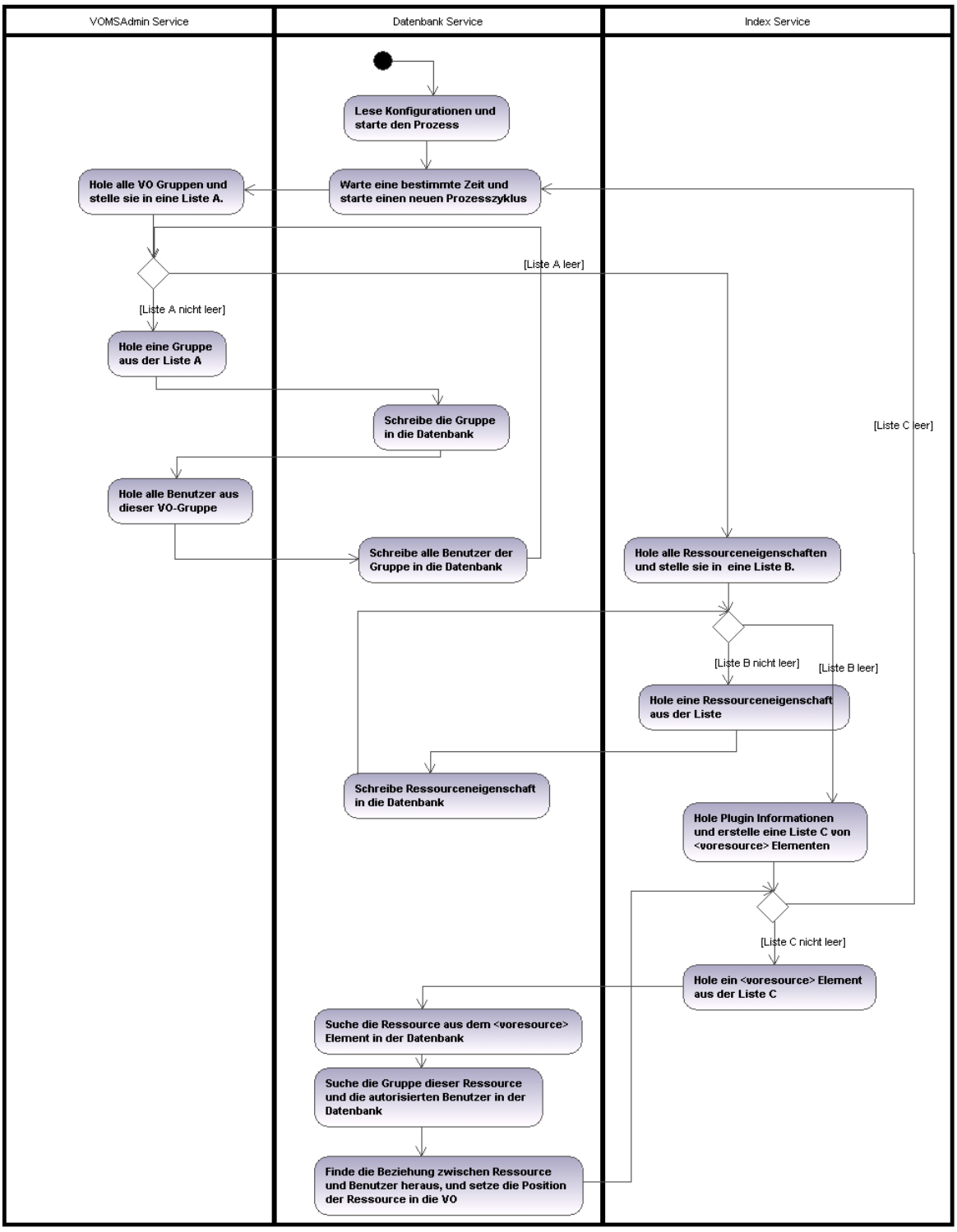


Abbildung 5.9: VO-Monitor Aktivitätdiagramm

5.5 Realisierung des Konzepts für andere VO-Management Systeme

Das oben beschriebene Konzept ist auf das Zusammenspiel von VOMS und MDS ausgelegt. In Kapitel 4 wurden andere Tools, insbesondere für das Management von Autorisierungen in VOs, vorgestellt. Da diese Tools auch in vielen Grid-Infrastrukturen angewendet werden, soll hier kurz dargestellt werden, wie das VO-Monitoringkonzept für diese Tools angepasst werden kann.
Abbildung 5.10 ist eine modifizierte Version von Abbildung 5.8. Sie zeigt, welche Anwendungsfälle geändert werden müssen, im Falle des Einsatzes von CAS oder Shibboleth. In Kapitel 4 wurde gezeigt, dass beide Tools auch Web Services Schnittstellen für die VO-Administration anbieten. Sie können daher leicht in das Monitoringkonzept integriert werden. Da aber die Autorisierungsinformationen in einer anderen Form (z.B. SAML Assertions) vorliegen, muss die Struktur der Datenbank geändert werden. Das oben vorgestellte Monitoringkonzept betrachtet nur VOMS-Attribute. Insofern ist auch das zugrunde liegende Datenbankschema stark von der Struktur der VOMS-Attribute abhängig. Für den Einsatz des VO-Monitors mit Shibboleth muss deshalb zunächst das Datenbankschema umstrukturiert werden, so dass es die Struktur vom SAML Assertions wiederspiegelt. Ebenso muss auch das Plugin leicht modifiziert werden. Die Struktur der publizierten Zusatzinformationen muss geändert und für SAML Assertions angepasst werden. In der Abbildung 5.10 kann man sehen, dass die Anwendungsfälle (mit Hintergrundfarbe) *Hole VO-Struktur /VO Mitglieder*, *Hole Plugin-Informationen*, *Schreibe in die Datenbank* und *Finde Beziehung Benutzer-Ressourcen* anzupassen sind. Ebenso ist der Anwendungsfall *Lese von der Datenbank* im Frontend zu ändern, weil sich die Datenbank-Struktur geändert hat. Der Index Service bleibt unverändert. Er braucht auch nicht ausgetauscht zu werden, da das Globus Toolkit als Grid-Middelware verwendet wird.

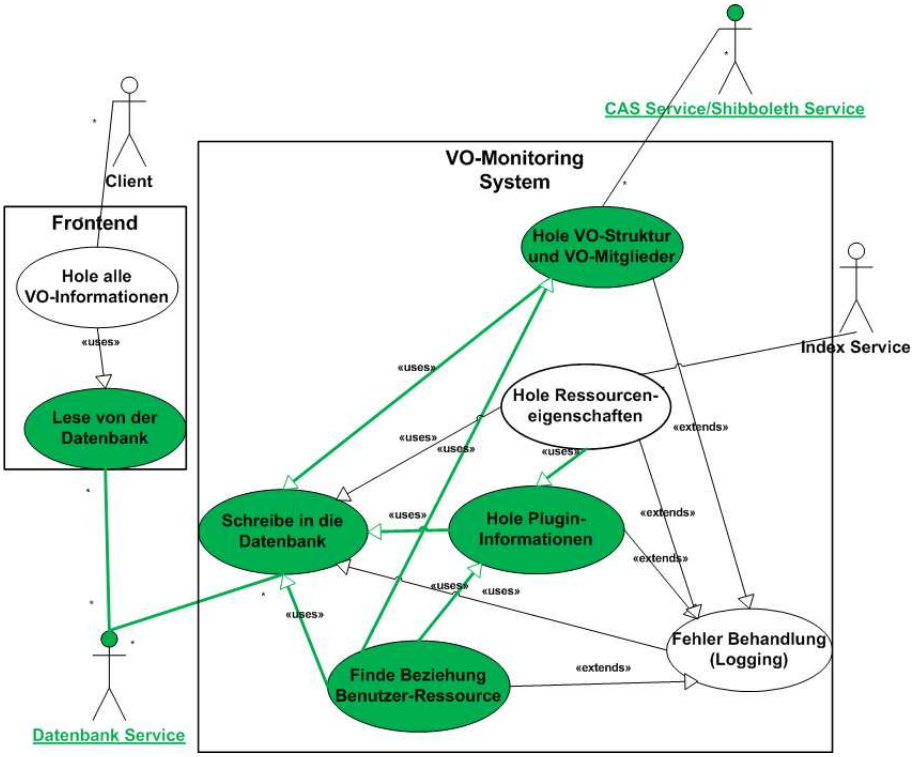


Abbildung 5.10: VO-Monitoring Anwendungsfalldiagramm angepasst für CAS oder Shibboleth.

5.6 Zusammenfassung

Dieses Kapitel hat sich mit der Entwicklung des VO-Monitors befasst. Bevor das eigentliche Konzept vorgestellt wurde, wurden zunächst die Komponentenschnittstellen beschrieben. Es wurde anschließend ein Plugin vorgestellt, das zusätzliche Informationen über die Ressourcen bereitstellte. Es nutzte dafür die Mechanismen des MDS4, die publizierten Daten die genaue Lokalisierung der Ressourcen in einer VO unterstützt haben. Der VO-Monitor selbst folgt einer prozess-orientierten Architektur. Er verarbeitet die Daten, die er vom VOMS und Index Service bekommt und legt sie in einer Datenbank ab.

Im Rahmen dieser Arbeit muss das erstellte Konzept auch implementiert werden, was zur Tragfähigkeit des Konzepts dienen soll. Das nächste Kapitel befasst sich mit dieser Implementierung des VO-Monitors. Es wird zuerst eine Implementierungsarchitektur vorgeschlagen und daraufhin ein Dienst für den Datenbankzugriff entwickelt, durch den die Monitoringdaten in der Datenbank abgelegt werden. Anhand dieser Implementierung soll schließlich die Bewertung des Konzepts im Kapitel 7 stattfinden.

6 Konzeptimplementierung

Dieses Kapitel befasst sich mit der Umsetzung des Konzeptes, das in Kapitel 5 beschrieben wurde. Es wird zunächst eine Implementierungsarchitektur vorgestellt. Dann werden die neuen entwickelten Komponenten (z.B. Datenbank Service) und anschließend der BPEL-Prozess beschrieben.

6.1 VO-Monitor Architektur

Die Abbildung 6.1 zeigt die Monitor-Architektur. In dieser Abbildung sind die Komponenten in Bereiche unterteilt:

- *Der Bereich A* besteht aus den Hauptkomponenten, die in dieser Arbeit entwickelt werden. Er entspricht genau dem VO-Monitoring System (siehe Abbildung 5.8)
- *Der Bereich B* besteht aus bereits existierenden Komponenten (VOMS und MDS). Diese Komponenten bieten die nötigen Dienste für VO-Management und Resource Discovery.
- *Der Bereich C* besteht ebenfalls aus neuen Komponenten. Er entspricht genau dem Frontend System (siehe Abbildung 5.8). Das Frontend, das eine benutzerfreundliche Darstellung der Monitor-Daten ermöglicht, wird jedoch in dieser Arbeit nicht implementiert.

6.1.1 Das VO-Monitoring System

Das VO-Monitoring System besteht aus einem Server und einem Datenbank Service.

Der *Server (VO-Monitor Server)* ist eine BPEL Engine, die den Monitor-Prozess bereitstellt, instanziiert und ausführt. Im Rahmen dieser Arbeit wurde die *ActiveBPEL Engine*[1] gewählt. ActiveBPEL ist eine robuste und ausgereifte BPEL-Engine. Sie ist unter einer Open-Source-Lizenz verfügbar. Sie implementiert die aktuellste BPEL-Spezifikation (Version 2.0) [BPELv2.0] vollständig und kann somit einen BPEL-Prozess direkt ausführen. Sie kann BPEL-Prozessdefinitionen und dazugehörige WSDL-Definitionen laden und die entsprechenden Web-Services Operationen zum Starten des Prozesses öffentlich zugänglich machen. Sobald eine eingehende Nachricht die Startaktivität eines Prozesses aufruft, wird eine Prozessinstanz erstellt und gestartet. Die Engine kontrolliert dann die vollständige Ausführung der Prozessinstanz, u.a. durch das Bearbeiten von Ereignissen, Warteschlangen und Alarmen. Zudem kommt ActiveBPEL mit einem Tool, dem ActiveBPEL Designer, das die Modellierung von BPEL-Prozessen mit einer graphischen Notation ermöglicht. Mit diesem Tool, einem Eclipse[2]-Plugin, wird der BPEL-Quellcode automatisch generiert und die Prozesse können während der Entwicklung bereits ausgetestet werden (*Debugging*). Wegen ihrer Open-Source-Lizenz und ihrer erweiterbaren Funktionalität können Entwickler die Engine für ihre eigenen Projekte anpassen. Der Datenbank Service ist ein Web Service, der den Zugriff auf eine entfernte Datenbank ermöglicht. Dieser Dienst bietet Operationen für die Speicherung der VO-Daten an. Unter diesem Dienst wird eine *MySQL-Datenbank*[3] betrieben. Die Datenbank Schema- und Service-Beschreibung erfolgt in Kapitel 6.2.

[1] `http://www.active-endpoints.com/active-bpel-engine-overview.htm`
[2] `http://www.eclipse.org/`
[3] `http://www.mysql.de/`

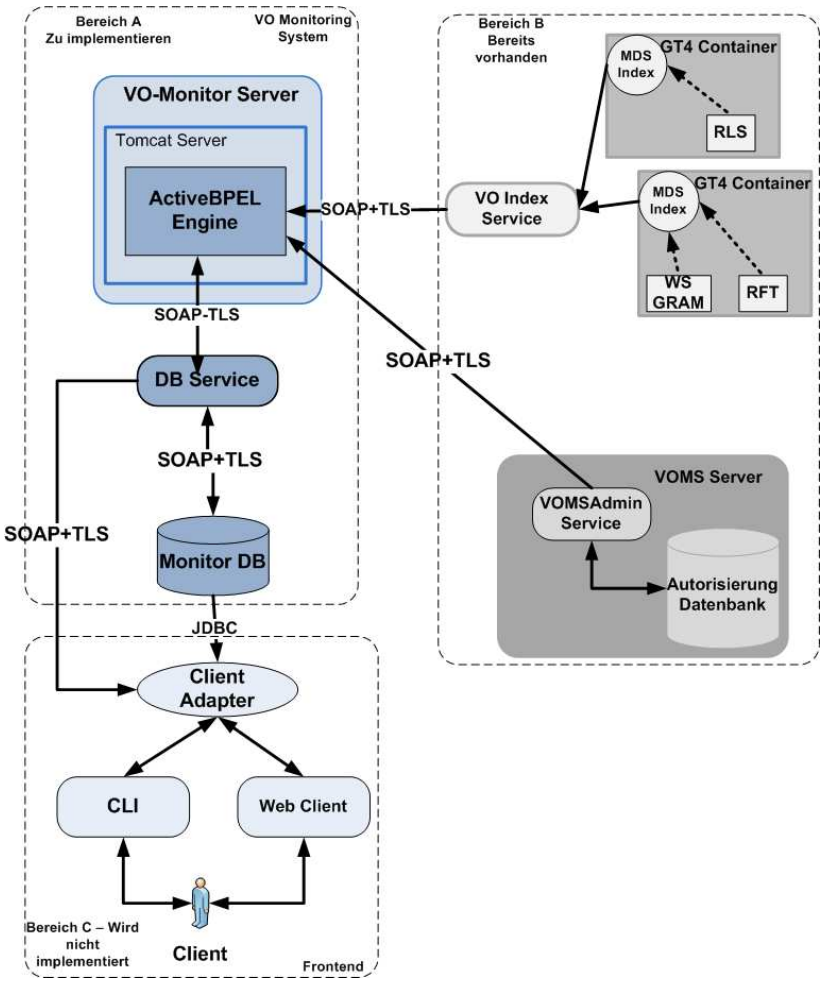


Abbildung 6.1: VO-Monitor Architektur

6.1.2 Das Frontend System

Wie bereits in Kapitel 5 erwähnt, ermöglicht das Frontend die gespeicherten VO-Daten anzusehen. Da ein großer Teil der Daten in XML-Dokumenten gespeichert wird, wird eine Komponente benötigt, die die Struktur dieser Daten verstehen und entsprechend in eine einfachere Form transformieren kann. Diese Komponente ist der Client Adapter. Er definiert eine API (*Application Programmable Interface*), die an einer Benutzer-Applikation (Command-Line oder Web-Applikation) gebunden werden muss. Sie kommuniziert mit der Datenbank über den Web Service oder direkt über *JDBC (Java Database Connectivity)*[4]. Das Frontend System wird in dieser Arbeit nicht implementiert.

6.2 Datenbank Service

In diesem Abschnitt soll der neue entwickelte Datenbank Service beschrieben werden. Bevor auf die Web Service-Operationen eingegangen wird, wird erst das Datenbankschema vorgestellt, das vom VO-Monitor verwendet wird.

[4] `http://java.sun.com/javase/technologies/database/`

6.2.1 Datenbank Schema

In dieser Arbeit wurde als DBMS MySql ausgewählt. Die Auswahl ist vor allem auf die Popularität und die Open-Source-Lizenz von MySql zurückzuführen. Aus diesem Grund lehnt sich der Typ der gespeicherten Daten stark an MySql an. Sollte ein anderes DBMS eingesetzt werden, müssten unter Umständen die Typen einiger Daten angepasst werden.
Abbildung 6.2 zeigt einen Überblick über das Datenbankschema. Die Pfeile in der Abbildung repräsentieren 1:N-Beziehungen (in Richtung der Pfeile). Betrachtet man beispielsweise den Pfeil der Tabelle `vo` zu `indexservice`, dann bedeutet er, dass die Tabelle `indexservice` eine Referenz auf die Tabelle `vo` hat (*1:N Beziehung*).
Im Folgenden wird die Struktur der verschiedenen Tabellen und deren Rollen beschrieben.

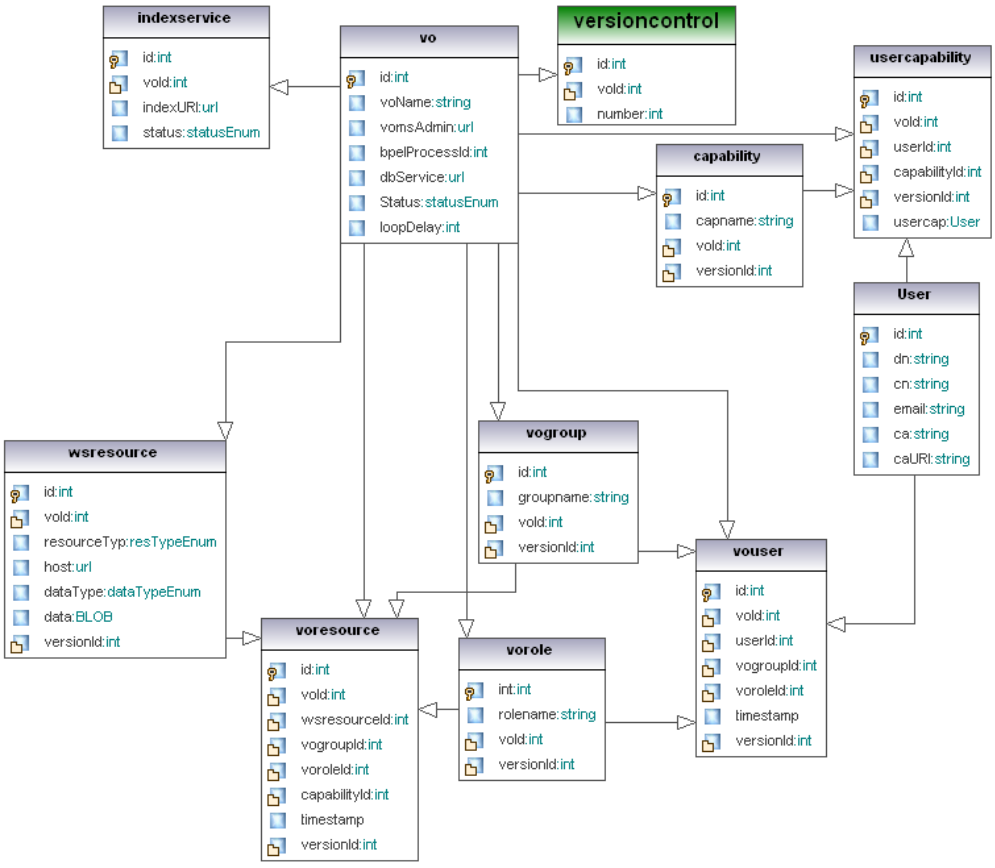


Abbildung 6.2: Datenbankschema

Konfigurationstabellen

Die Tabellen `vo` und `indexservice` werden für die Konfiguration des Monitors verwendet. Da der VO-Monitor gleichzeitig mehrere verschiedene VOs überwachen kann, dienen diese Tabellen vor allem dazu, die verschiedenen Prozesse zu unterscheiden.
Die Tabelle `indexservice` speichert den URL eines VO-Dienstes (Spalte `indexURI`). Sie enthält auch einen Fremdschlüssel auf die Tabelle `vo` (Spalte `voId`). Auf diese Weise können eine oder mehrere VO-Indizes für das Monitoring der Ressourcen innerhalb einer einzigen VO verwendet werden (siehe Kapitel 5.3.1). Die Daten in diesen Tabellen werden beim Prozessstart eingetragen.
Die Tabelle `vo` enthält Konfigurationsinformationen über die verschiedenen Prozesse. Ein Eintrag in dieser Tabelle entspricht genau einem Prozess, der für eine VO zuständig ist. Die Tabelle enthält außerdem folgende Spalten:

- `voName`: Der Name der VO
- `vomsAdmin`: Ein URL auf das VOMSAdmin, das für eine VO zuständig ist.
- `bpelAdminProcessId`: Eine eindeutige ID von dem BPEL-Prozess, der für die VO zuständig ist. Da mehrere Prozesse in der gleichen BPEL-Engine für verschiedene VOs ausgeführt werden können, kann durch diese ID der richtige BPEL-Prozess gefunden werden.
- `dbService`: Ein Verweis auf den Datenbank Service, der für diese VO zuständig ist.
- `status`: Der Status dient dazu, den Zustand des Prozesses zu erfahren. Der Zustand wird selbst vom BPEL-Prozess gesetzt. Falls ein Fehler auftritt, wird entsprechend der Status geändert. Mögliche Werte sind `RUNNING, FAILED, STOPPED`.
- `loopDelay`: Definiert die Intervallzeit zwischen Prozesszyklen.

Versionsverwaltung der Daten

In Kapitel 5.4.3 wurde durch ein Aktivitätsdiagramm gezeigt, dass der für eine VO verantwortliche Prozess unendlich ausgeführt werden soll, falls keine Fehler auftreten. Zur Vermeidung von Overhead bestimmt die Intervallzeit die Dauer zwischen dem Ende eines Prozesszyklus und dem Anfang des nächsten Zyklus. Nach jedem Zyklus werden alle Daten in die Datenbasis neu geschrieben. Um einen Überblick über die Daten aus den verschiedenen Prozesszyklen zu behalten, wurde eine Versionsverwaltung eingeführt. Die Tabelle `versioncontrol` enthält für jeden Prozess eine eindeutige Versionsnummer für die Daten, die von verschiedenen Prozesszyklen abgelegt wurden. Für einen gegebenen Prozess enthält die Spalte `number` eine eindeutige Nummer für jede Datenversion. Die Version mit der höchsten Nummer ist die aktuellste Version. Alle anderen Tabellen in der Datenbank haben einen Fremdschlüssel auf diese Tabelle. Somit ist es möglich, durch eine Versionsangabe entsprechende Daten aus den verschiedenen Tabellen abzurufen.

Tabellen für VO-Daten

Außer der in den zwei vorherigen Abschnitten vorgestellten Tabellen, dienen alle anderen Tabellen zur Speicherung der VO-Daten.

- `vogroup`: In dieser Tabelle werden die VO-Gruppen abgelegt. Der Gruppenname hat die gleiche Syntax wie VOMS-Attribute. Beispielsweise „*/VO1/Mitarbeiter*“ steht für die Gruppe „*Mitarbeiter*“ in der „VO1“.
- `vorole`: Genauso wie bei der `vogroup` werden hier alle verfügbaren Rollen (z.B. „*Role=Manager*“) in der VO gespeichert.
- `capability`: Hier werden alle Capabilities (z.B. „*capability=EDV-Skills*“) gespeichert, die in der VO verfügbar sind.
- `user`: In dieser Tabelle werden alle Benutzer abgelegt, die vom Monitor erfasst wurden. Ein Benutzer wird hauptsächlich durch seinen X.509 Subject Name identifiziert. Durch die Tabelle `vouser` kann für einen beliebigen Benutzer herausgefunden werden, in welcher VO und in welcher Gruppe er sich befindet.
- `usercapability`: Die Tabelle stellt eine Beziehung zwischen einem Benutzer und einer Capability (Fähigkeit) her. Sie enthält einen Fremdschlüssel auf die Tabelle `user`, wo die Benutzerdaten liegen. Durch diese Tabelle kann eine Client-Applikation erfahren, welche Benutzer welche Capabilities besitzen.
- `vouser`: Die Einträge in dieser Tabelle repräsentieren die VO-Mitglieder. Die Tabelle `user` enthält die Benutzerinformationen, die Tabelle `vouser` dagegen die Attribute oder Rechte von diesen Benutzern in der VO. Sie beinhaltet Fremdschlüssel auf die Tabellen `vogroup`, `vorole` und `user`. Somit ist es aus dieser Tabelle ableitbar, zu welcher Gruppe ein Benutzer in einer VO gehört und welche Rolle, Fähigkeit oder Rechte er besitzt.

- `wsresource`: In dieser Tabelle werden Informationen über WS-Resourcen gespeichert. Die WS-Ressourcen sind Grid-Dienste, die Zugriff auf physische Ressourcen ermöglichen. Wie in den vorangehenden Kapiteln bereits erwähnt, bietet das Globus Toolkit Standard Dienste wie das WS-GRAM oder den RFT. Hinter einem WS-GRAM Dienst können verschiedene Job-Schedulers wie Fork, PBS, etc. liegen. In der Spalte `resourceType` wird der Typ der WS-Resource angegeben. Mögliche Werte sind GRAM-Fork, GRAM-PBS, GRAM-Condor, etc. (siehe Kapitel 5.3.3). Die Spalte `host` gibt einen Verweis (URL) auf die WS-Ressourcen an. Beispielsweise hat ein WS-GRAM Dienst den Host *https://hydra.ari.uniheidelberg.de:8443/wsrf/services/ManagedJobFactoryService*. In der Spalte `dataType` wird der Typ der gespeicherten Daten angegeben. Diese entsprechen den Inhalten des Elements `<aggregatorData>`. Das `<aggregatorData>`-Element ist ein Kind des `Content`-Elements in einem Index-Eintrag (Entry-Element) (siehe Kapitel 5.1.2 und 5.1.2). Mögliche Werte der `dataTyp`-Spalte sind `GLUECE`, `RFT-DATA` oder `UNKNOWN`. Die Ressourceneigenschaften, die durch einen WS-GRAM Service abgerufen werden, sind meistens durch ein spezielles XML-Schema, das *GLUE CE Schema* (siehe Anhang D.1), gekapselt. Der Wert `GLUECE` in der Spalte `dataType` bedeutet also, dass die Ressourceneigenschaften durch diesen GLUE CE-Datentyp dargestellt sind. `RFT-DATA` bedeutet, dass es sich um Ressourceneigenschaften handelt, die aus dem RFT Service abgerufen wurden. Die Ressourceneigenschaften selbst werden in der Spalte `data` als *BLOB (Binary Large Object)* gespeichert. Die Typangabe der Daten ist vor allem für einen Client wichtig, damit er aus dem VO-Monitor die Informationstypen richtig interpretieren kann.
- `voresource`: Diese Tabelle enthält nur Fremdschlüssel auf die Tabellen `vorole`, `vogroup`, `wsresource` und `capability`. Aus den Informationen, die durch das Integration-Plugin geliefert wurden, kann die genaue Position (siehe Kapitel 5.3.1) einer gegebenen WS-Resource in einer VO festgestellt werden. Diese Position (die eigentlich den Zugriffsinformationen entspricht, die in der Zugriffskontrollliste stehen) besteht aus einer Gruppe, einer Rolle oder einer Capability (z.B. */VO1/OrganisationX/Role=Management/Capability=NULL*) in einer gegebenen VO.

6.2.2 Web Services Beschreibung

Der Datenbank Service ist ein Web Service, der Zugriffsoperationen für das oben vorgestellte Datenbankschema bietet. Diese Operationen lehnen sich stark an das verwendete Datenbank Schema an. Im Folgenden werden die von diesem Service verwendeten Datentypen in *XSD (XML Schema Definition)* [XSD] vorgestellt. Anschließend werden die Dienstoperationen vorgestellt. Eine komplette Auflistung der WSDL ist in der beigefügten CD (siehe Anhang A.1) zu finden.

Typbeschreibung:
In der folgenden Auflistung sind die Typen definiert, die von den Nachrichten des Datenbank Services verwendet werden. Der Typ `dbAnyType` wird beispielsweise von Nachrichten verwendet, die die Ressourceneigenschaften beinhalten. Die Eigenschaften werden dann in der Spalte `data` der Datenbanktabelle `wsresource` gespeichert. Der Typ `resourceDataType` definiert die verschiedenen Datentypen. Diese Auflistung definiert konzeptionell eine Abbildung der verschiedenen Nachrichtentypen auf die Datenbanktabellentypen des Datenbankschemas (siehe Abbildung 6.2). Der Nachrichtentyp `User` ist beispielsweise für die Datenbanktabelle `user` bestimmt. Die Beziehungen können gefunden werden, indem die Tabellenbeschreibungen im Abschnitt 6.2.1 mit den hier aufgelisteten Typen verglichen werden.

Listing 6.1: Typenbeschreibung des Datenbank-Services

```
<types>
 <xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema"
      targetNamespace="http://nm.ifi.lmu.de/VO-Monitor/DBService"
      xmlns:soapenc="http://schemas.xmlsoap.org/soap/encoding/"
      xmlns:wsdl="http://schemas.xmlsoap.org/wsdl/">
   <xsd:import namespace="http://schemas.xmlsoap.org/soap/encoding/" />
   <xsd:complexType name="dbAnyType" mixed="true">
       <xsd:sequence>
         <xsd:any processContents="skip" minOccurs="0" maxOccurs="unbounded"/>
       </xsd:sequence>
   </xsd:complexType>
```

```
<xsd:complexType name="DBServiceExceptionType">
    <xsd:sequence>
      <xsd:element name="description" type="xsd:string" />
      <xsd:element name="timestamp" type="xsd:dateTime" />
      <xsd:element name="details" type="xsd:string" />
    </xsd:sequence>
</xsd:complexType>
 <xsd:element name="DBServiceException" type="tns:DBServiceExceptionType"></
     xsd:element>
<xsd:complexType name="VOIdentifierType">
   <xsd:all>
     <xsd:element name="voName" nillable="false" type="xsd:string"/>
     <xsd:element name="vomsAdminURI" nillable="false" type="xsd:anyURI"/>
     <xsd:element name="bpelProcessID" nillable="false" type="xsd:int"/>
   </xsd:all>
</xsd:complexType>
 <xsd:complexType name="ResourceIdentifierType">
   <xsd:all>
     <xsd:element name="host" nillable="false" type="xsd:anyURI"/>
     <xsd:element name="type" nillable="false" type="tns:resourceType"/>
   </xsd:all>
</xsd:complexType>
<xsd:complexType name="User">
   <xsd:sequence>
      <xsd:element name="CA" nillable="true" type="xsd:string"/>
      <xsd:element name="CN" nillable="true" type="xsd:string"/>
      <xsd:element name="DN" nillable="true" type="xsd:string"/>
      <xsd:element name="certUri" nillable="true" type="xsd:string"/>
      <xsd:element name="mail" nillable="true" type="xsd:string"/>
   </xsd:sequence>
</xsd:complexType>
<!-- Wird eingesetzt für die Speicherung von mehreren Users als Array (Liste)
    -->
<xsd:complexType name="ArrayOfUser">
   <xsd:complexContent>
       <xsd:restriction base="soapenc:Array">
           <xsd:attribute ref="soapenc:arrayType" wsdl:arrayType="tns:User[]"
               />
       </xsd:restriction>
   </xsd:complexContent>
</xsd:complexType>
<!-- Representiert eine Liste von Zeichenketten-->
<xsd:complexType name="ArrayOfString">
   <xsd:complexContent>
      <xsd:restriction base="soapenc:Array">
          <xsd:attribute ref="soapenc:arrayType" wsdl:arrayType="xsd:string[]"
              />
       </xsd:restriction>
   </xsd:complexContent>
</xsd:complexType>
<xsd:simpleType name="resourceType">
   <xsd:restriction base="xsd:string">
      <xsd:enumeration value="GRAM-Multi"/>
      <xsd:enumeration value="GRAM-Fork"/>
      <xsd:enumeration value="GRAM-PBS"/>
      <xsd:enumeration value="GRAM-Condor"/>
      <xsd:enumeration value="GRAM-Loadleveler"/>
      <xsd:enumeration value="GRAM-SGE"/>
      <xsd:enumeration value="GRAM-NQS"/>
      <xsd:enumeration value="RFT"/>
      <xsd:enumeration value="Service-URI"/>
```

```
      </xsd:restriction>
   </xsd:simpleType>
   <xsd:simpleType name="resourceDataType">
      <xsd:restriction base="xsd:string">
         <xsd:enumeration value="GLUECE"/>
         <xsd:enumeration value="RFT-DATA"/>
         <xsd:enumeration value="UNKNOWN"/>
      </xsd:restriction>
   </xsd:simpleType>
   <xsd:complexType name="WS-ResourceType">
      <xsd:sequence>
         <xsd:element name="resource" type="tns:resourceType" />
         <xsd:element name="host" type="xsd:anyURI" />
         <xsd:element name="resourceData" type="tns:resourceDataType" />
         <xsd:element name="data" type="tns:dbAnyType" />
      </xsd:sequence>
   </xsd:complexType>
   <xsd:simpleType name="URIStatusType">
      <xsd:restriction base="xsd:string">
         <xsd:enumeration value="NOTAVAILABLE"/>
         <xsd:enumeration value="AVAILABLE"/>>
      </xsd:restriction>
   </xsd:simpleType>
   <xsd:simpleType name="ProcessStatusType">
      <xsd:restriction base="xsd:string">
         <xsd:enumeration value="RUNNING"/>
         <xsd:enumeration value="STOPPED"/>
         <xsd:enumeration value="FAILED"/>>
      </xsd:restriction>
   </xsd:simpleType>
   <!-- Mögliche Fehler-Flags, die bei der Fehlerbehandlungsroutine verwendet
       werden.-->
   <xsd:simpleType name="LogLevelType">
      <xsd:restriction base="xsd:string">
         <xsd:enumeration value="ERROR"/>
         <xsd:enumeration value="FATAL"/>
      </xsd:restriction>
   </xsd:simpleType>
 </xsd:schema>
</types>
```

Schnittstellenbeschreibung:
Der Datenbank Service, dessen Dienstname `DBServiceSOAP` ist, bietet die unten gelisteten Operationen. Alle diese Operationen geben eine Nachricht von Typ `DBServiceException` (siehe XSD im Listing 6.1) zurück, falls ein Fehler während des Datenzugriffs auftritt. Jede Fehlersituation resultiert im Start einer entsprechenden Fehlerbehandlungsroutine. Das WSDL-Dokument ist in der beigefügten CD (siehe Anhang A.1) zu finden.

`DBServiceSOAP`

- `getLoopIntervalTime` gibt die Intervallzeit zwischen jedem Prozesszyklus zurück.
- `setProcessStatus` setzt den Prozessstatus.
- `writeLogEntry` wird verwendet für die Protokollierung des Prozesses.
- `initMonitorConfiguration` schreibt die Prozesskonfigurationsparameter in die Datenbank.
- `increaseVersionControl` inkrementiert die Version der von einem Prozess abgelegten Daten.
- `insertGroup` speichert eine neue VO-Gruppe in der Datenbank.

- `insertGroups` speichert eine Liste von VO-Gruppen in der Datenbank.
- `insertRole` speichert eine neue VO-Rolle.
- `insertRoles` speichert eine Liste von VO-Rollen.
- `insertUser` speichert einen neuen Benutzer.
- `insertUsers` speichert eine Liste von Benutzern.
- `insertSubGroup` definiert für eine gegebene VO-Gruppe eine neue Untergruppe.
- `insertCapability` schreibt eine neue Capability in die Datenbank.
- `insertCapabilities` schreibt eine Liste von Capabilities in die Datenbank.
- `setVOUserGroup` setzt die VO-Gruppe eines bereits gespeicherten Benutzers.
- `setVOUserGroups` setzt die VO-Gruppen eines bereits gespeicherten Benutzers.
- `setVOUserRole` setzt die VO-Rolle eines bereits gespeicherten Benutzers.
- `setVOUserRoles` setzt die VO-Rollen eines bereits gespeicherten Benutzers.
- `setUserCapability` setzt die Capability eines bereits gespeicherten Benutzers.
- `setUserCapabilities` setzt die Capabilities eines bereits gespeicherten Benutzers.
- `setResource` speichert eine Ressource und ihre Eigenschaften in der Datenbank.
- `getResourceId` sucht eine Ressource in der Datenbank und liefert ihre Eigenschaften zurück.
- `setVOResource` setzt die VO-Zugehörigkeit einer bereits gespeicherten Ressource.
- `getPath4User` gibt für einen gegebenen Benutzer deren VOMS-Attribute zurück.
- `isPathExists` testet, ob ein gegebenes VOMS-Attribut in einer VO existiert.

6.3 BPEL-Prozess-Beschreibung

Dieser Kapitel ist der Beschreibung des VO-Monitor-Prozesses (ein BPEL-Prozess) gewidmet. Eine Einführung in BPEL und die BPEL-Syntax ist im Anhang B zu finden.
Zur Beschreibung des BPEL-Prozesses wird auf das VO-Monitor Aktivitätsdiagramm (in Abbildung 5.10) zurückgegriffen, da es die wesentlichen Aktivitäten des Prozesses enthält. Die Abbildung 6.3 (eine Variante der Abbildung 5.10) enthält Nummern für alle Aktivitäten. Manche Aktivitäten bestehen aus Subaktivitäten. Beispielsweise besteht die Aktivität 3 aus vier verschiedenen Subaktivitäten (3.1, 3.2, 3.3 und 3.4).
In diesem Abschnitt werden für die Aktivitäten 2, 3, 4 und 5 die BPEL-Quellcodes und die dahinter stehenden Algorithmen kurz vorgestellt. Die Aktivität 1, die für die Konfiguration einer Prozessinstanz zuständig ist, ist für das VO-Monitoringkonzept nicht relevant und wird hier nicht weiter betrachtet.
Bevor in die Beschreibung des Prozesses eingegangen wird, werden andere Aspekte des BPEL-Prozesses wie der Prozess Service, die Kommunikationsbeziehung zwischen den Gesprächspartnern und der Fehlerbehandlungsmechanismus kurz vorgestellt.

Monitor Service
BPEL stellt selbst einen Service dar. Somit werden BPEL-Prozesse wie alle anderen Web Services bereitgestellt und ihre Schnittstellen lassen sich durch WSDL-Dokumente beschreiben.
Das folgende WSDL-Dokument beschreibt den Monitor Service. Der Monitor Service ist ein BPEL-Prozess Service. Er bietet Operationen für das Starten von VO-Monitoring-Prozessen. Das WSDL definiert eine Request-Operation (`startMonitorInstance`). Durch diese Operation werden die Konfigurationsparameter dem BPEL-Prozess übergeben. Es wird dem BPEL-Prozess also mitgeteilt, welche VO überwacht werden soll, welche Datenbank verwendet werden soll und welche VO-Indizes für die VO zuständig sind.

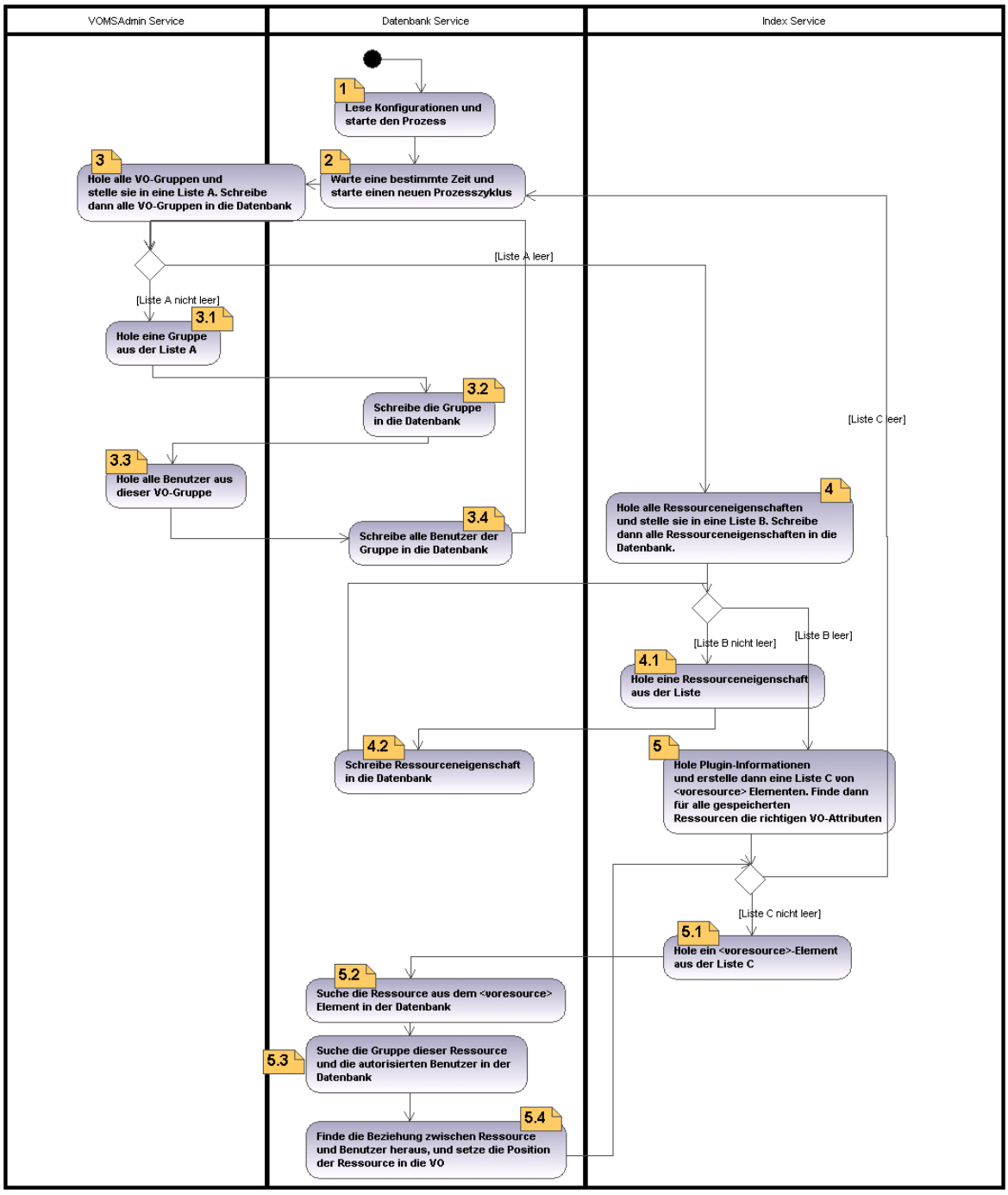


Abbildung 6.3: Aktivitätsdiagramm des VO-Monitors (annotierte Version)

Listing 6.2: WSDL des Monitor-Services

```
<wsdl:definitions .....name="VO-Monitor" targetNamespace="http://nm.ifi.lmu.de/VO
    -Monitor"
 xmlns:p="http://schemas.xmlsoap.org/wsdl/http/"
xmlns:plnk="http://docs.oasis-open.org/wsbpel/2.0/plnktype">

<wsdl:types>
 <xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema"
targetNamespace="http://nm.ifi.lmu.de/VO-Monitor">
```

```
      <xsd:complexType name="initParameterType">
      <xsd:all>
        <xsd:element name="dbService" nillable="false" type="xsd:anyURI"/>
        <xsd:element name="indexService" nillable="false" type="xsd:anyURI"/>
        <xsd:element name="loopDelaySecs" nillable="false" type="xsd:int"/>
        <xsd:element name="voName" nillable="false" type="xsd:string"/>
        <xsd:element name="vomsAdminURI" nillable="false" type="xsd:anyURI"/>
      </xsd:all>
   </xsd:complexType>
 </xsd:schema>
</wsdl:types>

<wsdl:message name="startMonitorRequest">
      <wsdl:part name="startMonitorRequest" type="tns:initParameterType"/>
</wsdl:message>

   <wsdl:portType name="VO-Monitor">
            <wsdl:operation name="startMonitorInstance">
          <wsdl:input message="tns:startMonitorRequest" name="startMonitorRequest"
              />
            </wsdl:operation>
</wsdl:portType>
..............
  <plnk:partnerLinkType name="VO-MonitorLT">
            <plnk:role name="process" portType="tns:VO-Monitor" />
</plnk:partnerLinkType>

</wsdl:definitions>
```

Kommunikationsbeziehung

Das Starten eines Monitoring-Prozesses erfolgt über den Monitor Service, der beispielsweise durch einen Monitor-Administrator auf der Basis entsprechender Parameter ausgeführt wird. Eine Client-Applikation, die das Starten und die entfernte Verwaltung der Prozesse ermöglicht, ist in der beigefügten CD (siehe *console-app* im Anhang A.1) zu finden. Das `<partnerLink>`-Element mit dem Name `process` im folgenden Listing definiert die Kommunikationsbeziehung zwischen dem BPEL-Prozess und einer Client-Applikation, die auf den Monitor Service zugreifen möchte. Das `<partnerLink>`-Element allgemein teilt dem BPEL-Prozess mit, mit welchen Web Services er interagieren kann. Die weiteren `<partnerLink>`-Elemente definieren die Verbindungen zu den Web Services (Index Service, VOMSAdmin, Datenbank Service), mit denen der BPEL-Prozess kommunizieren soll. Mehr über die hier verwendete Syntax ist im Anhang B zu finden.

Listing 6.3: PartnerLinks Definition

```
<bpel:partnerLinks>
      <!-- PartnerLink BPEL Prozess und Prozess Service -->
      <bpel:partnerLink myRole="process" name="MonitorProcessPartnerLink"
          partnerLinkType="ns15:VO-MonitorLT"/>

      <!-- PartnerLink BPEL Prozess und Index Service -->
      <bpel:partnerLink name="IndexServicePartnerLink" partnerLinkType="ns2:
          RPServiceLT" partnerRole="provider"/>

      <!-- PartnerLink BPEL Prozess und VOMSAdmin Service -->
      <bpel:partnerLink name="VOMSAdminPartnerLink" partnerLinkType="voms:
          VOMSAdminLT" partnerRole="VOMSAdmin"/>

      <!-- PartnerLink BPEL Prozess und Datenbank Service -->
      <bpel:partnerLink name="DatabasePartnerLink" partnerLinkType="db:
          DBServiceLT" partnerRole="Database"/>
</bpel:partnerLinks>
```

Fehlerbehandlungsmechanismus

Der VO-Monitor behandelt und protokolliert erwartete (z.B. Fehler beim Zugriff auf VOMSAdmin und den Datenbank Service) und unerwartete Fehler. Jede Prozessaktivität benutzt einen Fehlerbehandlungsmechanismus abhängig von den Services, die in dieser Aktivität abgerufen werden. Das Aktivitätsdiagramm in Abbildung 6.3 gibt einen Überblick über die verwendeten Services für jede Aktivität. Eine Fehlerbehandlungsroutine wird in BPEL in einem `<faultHandlers>`-Element gekapselt. Dieses Element enthält dann das `catch`-Element oder das `catchAll`-Element. Das `<catch>`-Element definiert eine Fehlerbehandlungsroutine für einen bestimmten Fehler, der bekannt ist und unter Umständen zu erwarten ist. Das `<catchAll>`-Element definiert eine Routine für unerwartete Fehler. Die Fehlerbehandlungsroutine im Monitor-Prozess besteht aus drei verschiedenen Aktivitäten, die einzeln oder alle gemeinsam auftreten können:

- `dbWriteLogEntry` ist eine `BPEL-<invoke>`-Aktivität, die einen Fehler protokolliert. Es können zwei Fehlerarten (`DBException` und `VOMSException`) protokolliert werden. `VOMSException` kann beim Aufruf des VOMSAdmin Services auftreten, während `DBException` beim Aufruf des Datenbank Services.
- `dbSetProcessStatus` ist ebenso eine `BPEL-<invoke>`-Aktivität, die den Prozess-Status in der Datenbank protokolliert. Falls der Fehler gravierend ist (z.B. wenn ein Dienst nicht mehr verfügbar ist) und der Prozess beendet werden muss, wird der Status zu `STOPPED` gesetzt und anschließend die Aktivität `<exit>` ausgeführt.
- `<exit>`: eine BPEL-Aktivität zum Beenden einer Prozessinstanz.

Falls bekannte Fehler wie `DBException` und `VOMSException` auftretten, wird der Fehler protokolliert und die Prozessausführung im nächsten Zyklus fortgeführt. Anderenfalls wird durch die `<catchAll>`-Aktivität der Prozess beendet. Das folgende Listing zeigt einen Ausschnitt der Fehlerbehandlungsroutine im VO-Monitor-Prozess.

Listing 6.4: Fehlerbehandlungsroutine

```
<bpel:faultHandlers>
              <bpel:catch faultElement="db:DBServiceException" faultVariable="
                  DBException">
                <bpel:sequence>
                  <bpel:assign name="initDbWriteLogEntryRequest">
                    <bpel:copy>
                      <bpel:from variable="voID"/>
                      <bpel:to>$dbWriteLogEntryRequest.voId</bpel:to>
                    </bpel:copy>
                    <bpel:copy>
                      <bpel:from>$DBException/description</bpel:from>
                      <bpel:to>$dbWriteLogEntryRequest.message</bpel:to>
                    </bpel:copy>
                    <bpel:copy>
                      <bpel:from>$DBException/timestamp</bpel:from>
                      <bpel:to>$dbWriteLogEntryRequest.date</bpel:to>
                    </bpel:copy>
                    <bpel:copy>
                      <bpel:from>$DBException/details</bpel:from>
                      <bpel:to>$dbWriteLogEntryRequest.stacktrace</bpel:to>
                    </bpel:copy>
                    <bpel:copy>
                      <bpel:from>
                        <bpel:literal>ERROR</bpel:literal>
                      </bpel:from>
                      <bpel:to>$dbWriteLogEntryRequest.logLevel</bpel:to>
                    </bpel:copy>
                  </bpel:assign>
                  <bpel:invoke inputVariable="dbWriteLogEntryRequest"
name="dbWriteLogEntry" operation="writeLogEntry" partnerLink="DatabasePartnerLink
    "
portType="db:DBService"/>
```

```
                    <bpel:empty/>
                  </bpel:sequence>
                </bpel:catch>
                <bpel:catch faultName="voms:VOMSException">
                   ..............
                </bpel:catch>

                 <bpel:catchAll>
                   <bpel:sequence>
          <bpel:assign name="initOps">
            <bpel:copy>
              <bpel:from variable="voID"/>
              <bpel:to>$dbWriteLogEntryRequest.voId</bpel:to>
            </bpel:copy>
            <bpel:copy>
              <bpel:from>
                <bpel:literal>Unexpected Error:Maybe bad process configuration</
                    bpel:literal>
              </bpel:from>
              <bpel:to>$dbWriteLogEntryRequest.message</bpel:to>
            </bpel:copy>
            <bpel:copy>
              <bpel:from>
                <bpel:literal>FATAL</bpel:literal>
              </bpel:from>
              <bpel:to>$dbWriteLogEntryRequest.logLevel</bpel:to>
            </bpel:copy>
            <bpel:copy>
              <bpel:from variable="voID"/>
              <bpel:to>$dbSetProcessStatusRequest.voId</bpel:to>
            </bpel:copy>
            <bpel:copy>
              <bpel:from>
                <bpel:literal>FAILED</bpel:literal>
              </bpel:from>
              <bpel:to>$dbSetProcessStatusRequest.status</bpel:to>
            </bpel:copy>
            <bpel:copy>
              <bpel:from>
                <bpel:literal>null</bpel:literal>
              </bpel:from>
              <bpel:to>$dbWriteLogEntryRequest.stacktrace</bpel:to>
            </bpel:copy>
            <bpel:copy>
              <bpel:from>number(0)</bpel:from>
              <bpel:to>$dbWriteLogEntryRequest.date</bpel:to>
            </bpel:copy>
          </bpel:assign>
          <bpel:invoke inputVariable="dbWriteLogEntryRequest" name="
              dbWriteLogEntry"
operation="writeLogEntry" partnerLink="DatabasePartnerLink" portType="db:
    DBService"/>
          <bpel:invoke inputVariable="dbSetProcessStatusRequest" name="
              dbSetProcessStatus"
operation="setProcessStatus" partnerLink="DatabasePartnerLink" portType="db:
    DBService"/>
          <bpel:exit/>
        </bpel:sequence>
                 </bpel:catchAll>
</bpel:faultHandlers>
```

6.3.1 Beschreibung der Prozessaktivitäten

Vor diesem Hintergrund werden nun die jeweiligen Aktivitäten des Monitor-Prozesses kurz beschrieben. Aus Übersichtlichkeits- und Platzgründen wird in diesem Abschnitt auf eine Beschreibung der BPEL-Syntax verzichtet. Für die Beschreibung der Algorithmen der verschiedenen Aktivitäten wird hier eine einfache Pseudocode-Notation verwendet. Da BPEL-Konstrukte denen einer normalen Programmiersprache wie Java ähneln, können die gleichen Notationen wie *for each, while, if else, etc.* im Pseudocode verwendet werden. Eine vollständige Beschreibung der BPEL-Konstrukte ist in [BPELv2.0] zu finden. Für jede Aktivität wird das entsprechende BPEL-Element angegeben.

Die Aktivitäten des in Abbildung 6.3 dargestellten Diagramms werden hier beschrieben:

- **Aktivität 2:** In dieser Aktivität wird der Prozess eine bestimmte Zeit blockiert, bevor mit der Ausführung fortgefahren werden kann. Nach dieser Aktivität beginnt ein neuer Prozesszyklus.

  BPEL-Element: (Siehe Listing C.1 im Anhang C.)

  ```
  <bpel:wait name="WaitTheDelay"> ........</bpel:wait>
  ```

- **Aktivität 3:** Diese Aktivität unterteilt sich in mehrere Sub-Aktivitäten. Die verwendeten Web Services sind VOMSAdmin und der Datenbank Service. Aus VOMSAdmin werden alle VO-Mitglieder, VO-Gruppen, VO-Rollen und VO-Capabilities erfasst. Die erfassten Daten werden dann in der Datenbank durch den Datenbank Service abgelegt.

  Pseudocode: `vo` ist eine BPEL-Variable, die eine VO identifiziert. `VOMSAdmin` steht für den VOMSAdmin Service und `DBService` für den Datenbank Service. Eine kurze Beschreibung der Service-Operationen ist jeweils in den Kapitel 5.2 und 6.2.2 zu finden. `var` deklariert eine BPEL-Variable.

  ```
  var groupname-list := VOMSAdmin.listSubGroups(vo);
  while groupname-list is not empty {
    var group := groupname-list.getFirstElement();
    DBService.insertGroup(vo,group);
    groupname-list.add(VOMSAdmin.listSubGroups(group))
  }
  var role-list := VOMSAdmin.listRoles(vo);
  DBService.insertRoles(vo,role-list);
  var user-list := VOMSAdmin.listMembers(vo);
  DBService.insertUsers(vo,user-list);
  for each user in user-list {
    DBService.setVOUserGroups(vo,user,VOMSAdmin.listGroups(vo,user));
    DBService.setVOUserRoles(vo,user,VOMSAdmin.listRolesOfUser(vo,user));
    DBService.setUserCapabilities(vo,user,VOMSAdmin.listCapabilitiesOfUser(vo,
        user));
  }
  var cap-list := VOMSAdmin.listCapabilities(vo);
  DBService.insertCapabilities(vo,cap-list);
  ```

  BPEL-Element: (Siehe Listing C.2 im Anhang C.)

  ```
  <bpel:scope name="VOMS">............</bpel:scope>
  ```

- **Aktivität 4:** Diese Aktivität besitzt ebenfalls Sub-Aktivitäten. Die verwendeten Services sind der Index Service und der Datenbank Service. In dieser Aktivität werden alle verfügbaren Ressourceneigenschaften aus dem VO-Index geholt und anschließend in die Datenbank geschrieben.

  Pseudocode: `IndexService` steht für den Index Service.

  ```
  var allResourceProperties = IndexService.QueryResourceProperties();
  ```

```
/**Durch einen Xpath-Ausdruck**/

var indexServiceEntryCount = allResourceProperties.getEntryCount();
for (i=0 --> indexServiceCount){

      /**Durch einen Xpath-Ausdruck **/

   var entry = allResourceProperties.getEntryAtIndex(i)
   DBService.setResource(entry);
}
```

BPEL-Element: (Siehe Listing C.3 im Anhang C.)

```
<bpel:scope name="MDS-IndexService"> .............</bpel:scope>
```

- **Aktivität 5:** Diese Aktivität verwendet den Datenbank Service. Da nun alle Mitglieder und Ressourcen einer VO in einer Datenbank gespeichert sind, müssen jetzt noch auf Basis der durch das Plugin gelieferten Daten die Zugehörigkeiten der Ressourcen zu VOs bestimmt werden (siehe Kapitel 5.3).

Pseudocode:

```
/**Nutze die Variable "IndexService" aus der vorherigen Aktivität
** (Aktivität 4)
*/
var allResourceProperties = IndexService.QueryResourceProperties();

/**Durch einen XPath-Ausdruck. Teste ob das Element <voresource>
** in den Entries vorhanden ist.
*/
if (vo-res-mapping supported) {

/**Durch einen XPath-Ausdruck. Hole alle vorhandenen <voresource>-Elementen
** aus den Entries und stellt sie in eine Liste.
*/
  voResourceList = allResourceProperties.getVOResourceList()

  for each (voResourceElt in voResourceList) {

/**Ein <voresource>-Element enthält den WS-Ressource-URL(Kapitel 5.3).
** Durch diesen URL werden die Ressourceneigenschaften in der Datenbank
** gesucht.Das zurückgegebene "wsResource" enthält alle Eigenschaften
** einer Ressource. Falls die Ressource in der Datenbank nicht
** existiert, dann wird nichts zurückgegeben.
*/

wsResource:= DBService.getResourceId(voResourceElt.getWSResourceHost());

/**Hier werden die "gridmap-user" und die "voms-attribute" aus dem
** <voresource>-Element extrahiert (siehe Kapitel 5.3). Für die
** extrahierten "gridmap-user" werden von der Datenbank die
** VO-Zugriffsrechte geholt. Die Rechte eines VO-Mitglieds
** (hier gridmap-user) bestimmen unmittelbar die Position der Ressource
** (die, dieses Mitglied verwenden darf) in der VO.
*/
    for each (gridmapUser in voResourceElt.grimapUsers) {
      voms-path := DBService.getPath4User(vo,gridmapUser);
      if (voms-path not null) then
        DBService.setVOResource(vo,wsResource,voms-path);
    }
```

```
/**Falls "voms-attribute" durch den Plugin publiziert wurden, dann
** bestimmen diese Attribute die Position der Ressourcen in der VO.
*/
     for each(voms-path in voResourceElt.vomsAttribtues) {
       if (DBService.isPathExists(vo, voms-path) then
          DBService.setVOResource(vo,wsResource,voms-path);
     }
      }
}
```

BPEL-Element: (Siehe Listing C.3 im Anhang C.)

```
<bpel:if name="IF-VOResMappingSupported">........</bpel:if>
```

6.4 Zusammenfassung

Dieses Kapitel hat sich mit der Implementierung des Monitoringkonzepts befasst. Es wurde zunächst eine Implementierungsarchitektur vorgestellt, dann wurden die Aktivitäten des Monitor-Prozesses beschrieben. Ein vollständiges Listing der Aktivitäten befindet sich im Anhang C. Im nächsten Kapitel wird anhand der gestellten VO-Monitoring-Anforderungen das implementierte Monitoringkonzept bewertet.

7 Bewertung

Im Kapitel 3 wurden Anforderungen für ein VO-Monitoring System gestellt. Diese Anforderungen unterteilen sich in funktionale und nicht-funktionale Anforderungen. Auf Grundlage der funktionalen Anforderungen wurden aktuelle Ansätze und Tools in Kapitel 4 bewertet und mit Hilfe dieser Ergebnissen in Kapitel 5 ein Monitoringkonzept entwickelt. In diesem Kapitel soll die in Kapitel 6 vorgestellte Lösung für die Realisierung des Konzepts anhand der Monitoring-Anforderungen bewertet werden.
Bei dieser Bewertung werden funktionale und den nicht-funktionale Aspekten betrachtet.

7.1 Bewertung der funktionalen Anforderungen

Die in Abbildung 6.1 vorgestellte Monitoring Architektur nutzt eine relationale Datenbank für die Speicherung der Daten. Diese Daten werden durch einen BPEL-Prozess gesammelt und in die Datenbank geschrieben. Die Auswertung und die Darstellung der Daten sollen eigentlich *nur* beim Frontend erfolgen, weil das Frontend nur mit der Datenbank kommuniziert, in der die VO-Daten abgelegt sind. Da wir aber in dieser Arbeit das Frontend nicht implementiert haben, wird deswegen das Datenbankschema für die Bewertung verwendet. Andere Aspekte wie Zuverlässigkeit, Effizienz oder Sicherheit werden im nächsten Abschnitt bewertet.
In der folgenden Diskussion hat die verwendete Datenbank das Schema, das in der Abbildung 6.2 dargestellt ist.

Anforderung 1
Der VO-Monitor muss hier die Änderungen einer VO-Struktur verfolgen können. Die Tabellen `vogroup` und `vorole` liefern die aktuellen Gruppen und Rollen in einer VO. Sollte eine Änderung in der VO vorkommen, dann wird der Prozess alle Änderungen in einen neuen Datensatz schreiben. Ebenso liefern die Tabellen `vouser`, `voresource` Informationen über die VO-Mitglieder und die Ressourcen.

Anforderung 2
Hier muss der VO-Monitor alle verfügbaren Informationen über VO-Mitglieder liefern können. Die Tabelle `vouser` enthält alle Mitglieder einer VO. Sie liefert durch ihre Spalten `vogroupId` und `voroleId` jeweils die Gruppen und die Rollen eines Mitglieds. Die Spalte `userId` verweist auf die Tabelle `user`, in der die Basisinformationen (Zertifikatinformationen, Heimatorganisation, Email, etc) über den Benutzer, dessen Mitgliedschaft in einer VO durch die Tabelle `vouser` repräsentiert ist, liegen. Durch die Tabelle `usercapability` können die Capabilities (Fähigkeiten) eines bestimmten Benutzers abgefragt werden.
Somit haben wir für jedes VO-Mitglied vollständige VOMS-Attribute. Es ist zu beachten, dass ein Mitglied in mehreren Gruppen sein oder über mehrere verschiedene Rollen in einer gleichen VO verfügen kann. Das lässt sich durch mehrere VOMS-Attribute repräsentieren. Die Tabelle `vouser` kann auch für einen einzigen Benutzer mehrere Datensätze über seine Mitgliedschaft haben.

Anforderung 3
Diese Anforderung erfordert, dass alle in einer VO vorhandenen Ressourcen erfasst werden können. Neben den Ressourceneigenschaften sind auch die Zugriffskontrollinformationen auf diese Ressourcen von großer Bedeutung. Diese Zugriffskontrollinformationen (in der VOMS-Attribute-Notation) bestimmen die Position der Ressourcen in der VO. Die Tabelle `wsresource` liefert die Ressourceneigenschaften und die Tabelle `voresource` bestimmt die Position einer Ressource. Diese Position wird durch die Gruppe (in der Spalte `groupId`), die Rolle (in der Spalte `roleId`) und die Capability (in der Spalte `capabilityId`) angegeben.

Anforderung 4
In dieser Anforderung wird vom VO-Monitor verlangt, Jobs oder Aktivitäten auf Ressourcen zu erfassen. Es soll auch für die jeweiligen Jobs die Identität des Aufrufers festgestellt werden.
Ressourceneigenschaften und Jobinformationen werden alle in derselben Tabelle `wsresource` gespeichert.

Jobinformationen gehören eigentlich zu den Ressourceneigenschaften. Sie werden von den Information Providers zur Verfügung gestellt. Die Struktur der gelieferten Informationen hängt vom Typ des Information Providers ab. Beispielsweise liefern der RFT Service und der WS-GRAM Service Informationen mit unterschiedlichen Strukturen. Der Typ der gespeicherten Daten wird in der Spalte `dataTyp` der Tabelle `wsresource` angegeben. Die Ressourceneigenschaften selbst (inklusive der Jobinformationen) sind in der Spalte `data` als XML-Dokument gespeichert. Die Auswertung dieses Dokuments (d.h. die Unterscheidung zwischen Jobs- und Ressourceneigenschaften) soll bei dem Frontend anhand des Datentyps stattfinden. Somit sind für alle erfassten Ressourcen auch Informationen über die darauf laufenden Jobs verfügbar.

Anforderung 5
In einer VO werden Ressourcen durch reale Organisationen bereitgestellt. Diese Organisationen legen die Zugriffsrechte auf ihre Ressourcen fest, und dies meistens ohne Kenntnis über die VO-Mitglieder zu haben. Auf der anderen Seite kann ein VO-Manager Mitglieder in eine Gruppe einordnen, ohne die Kenntnis über alle in dieser Gruppe bereitgestellten Ressourcen zu haben. Dies ist vor allem auf die Dynamik der VO zurückzuführen. Genau hier kommt die Anforderung 5 ins Spiel. Es sollen für jede Ressource auch die Zugriffsrechte als Ressourceneigenschaften geliefert werden. Somit kann die Position einer Ressource in der VO implizit durch die Position ihrer autorisierten Benutzer bestimmt werden. Die Tabelle `voressource` liefert diese Zugriffsinformationen. Zusammen mit den Daten aus der Tabelle `vouser`, ist es möglich die Beziehung zwischen Ressourcen und VO-Mitgliedern herzustellen. Der VO-Monitor oder die realen Organisation können beispielsweise diesen VO-Monitor verwenden, um zu den fehlenden Informationen zu gelangen.

7.2 Bewertung der nicht-funktionalen Anforderungen

In diesem Abschnitt wird der erstellte VO-Monitor nach den erstellten Nicht-Funktionalen Anforderungen (Anforderung 6) bewertet.
Für diese Bewertung wurde der VO-Monitor im Testbett (siehe Kapitel 3.3) getestet.

Leistungsverhalten/Effizienz:
Der VO-Monitor wurde für mehr als drei Wochen zum Einsatz gebracht. Bis dahin waren keine Ausfälle des Systems aufgetreten und der Monitor war immer verfügbar. Bei der Implementierung wurde viel Wert auf die Schnelligkeit gelegt. Monitoring-Systeme werden gewöhnlich gebraucht, um möglichst zeitnah alle Änderungen in einem System zu erfassen. Die Antwort- und Verarbeitungszeit sowie der Durchsatz stellten sich als angemessen heraus. Besonders zufriedenstellend waren die Effizienz und dieSchnelligkeit, mit denen der BPEL Prozess mit großen XML-Daten umgehen konnte und die Transportzeit dieser Daten durch SOAP-Nachrichten. Die ActiveBPEL Engine trug erheblich dazu bei.

Zuverlässigkeit:
Die Zuverlässigkeit definiert die Fähigkeit eines Systems, das Leistungsniveau unter festgelegten Bedingungen über einen festgelegten Zeitraum aufrecht zu erhalten.
In Kapitel 6.3 haben wir gesehen, dass der VO-Monitor über spezielle Routinen für Fehlerbehandlung verfügt. Es wurde auch gezeigt, dass zwischen kritischen und nicht kritischen Fehlern unterschieden wird. Nicht kritische Fehler treten beispielsweise auf, wenn die verwendeten Dienste (VOMSAdmin oder der VO-Index) falsche oder unvollständige Daten liefern. In solchen Fällen wird der VO-Monitor diese Fehler protokollieren, stoppen und nach einer gewissen Zeit wieder neu starten. Somit wurde eine gute Reife und Qualität erreicht und die Versagenfähigkeit durch Fehlerzustände erheblich reduziert.

Skalierbarkeit:
Die Skalierbarkeit definiert die Fähigkeit eines Systems mit steigenden Nutzerzahlen und Datenaufkommen zurechtzukommen. Der VO-Monitor wurde konzipiert, um mehrere VOs gleichzeitig überwachen zu können. Dabei bieten sich die Möglichkeiten, für die zu überwachenden VOs jeweils einen eigenen BPEL-Prozess zu starten und eine eigene Datenbank für die Speicherung der VO-Daten zu verwenden. Dieser Aspekt konnte leider nicht getestet werden, da das Testbett in erster Linie für die Tragfähigkeit des Konzepts realisiert wurde.

Sicherheit:
Das Globus Toolkit stellt mit seinem Sicherheitwerkzeug (GSI - siehe Kapitel 2.4) ein mächtiges Tool für den Aufbau sicherer Grid-Anwendungen dar. Alle Grid-Dienste (wie z.B. der Index Service) verwenden das

GSI. Das VOMS und das ActiveBPEL verwenden das TLS-Protokoll für den Datentransport (siehe Abbildung 6.1). Das VOMS, der Index Service und die ActiveBPEL Engine können auch die Authentifizierung und die Autorisierung implementieren. Somit ist der VO-Monitor gegen Angriffe gesichert und ermöglicht auch den Zugriff nur für die autorisierten Benutzer.

Übertragbarkeit:
Die Abbildung 6.1 zeigt, dass der VO-Monitor hauptsächlich aus einer Datenbank und einer BPEL-Engine besteht. Dafür wurden MySql und ActiveBPEL ausgewählt. Beide sind Open-source Technologien und können auf allen Betriebsystemen installiert und betrieben werden.

Benutzbarkeit:
Diese Anforderung soll durch das Frontend erfüllt werden. Das Frontend wurde nicht implementiert. Es soll die Daten aus der Datenbank (zum großen Teil als XML-Dokument) in eine lesbare und verständliche Form bringen. Andere Aspekte wie Look&Feel sind zu beachten und lassen sich leicht z.B. mit einer Web-Applikation realisieren.

8 Zusammenfassung und Ausblick

8.1 Zusammenfassung

Grids stellen eine neuartige IT-Infrastruktur dar, die eine gemeinsame und koordinierte Nutzung von weltweit verteilten Ressourcen ermöglicht. So können verschiedene Unternehmen, Universitäten, Forschungszentren oder sogar einzelne Personen im Rahmen einer Kooperation Teile ihrer Infrastruktur (Datenbank, Mainframe, Speicherkapazität) einer gemeinsamen Nutzung zuführen. Diese Kooperation wird durch die Bildung einer VO ermöglicht. Die Größe einer virtuellen Organisation kann abhängig von ihren Zielen und über die Zeit stark variieren. Wegen des Ausmaßes und der Dynamik von VOs werden verlässliche Verfahren benötigt, die einen gesamten Überblick über die Teilnehmer und die vorhandenen Ressourcen in einer VO geben können. Mit genau dieser Fragestellung beschäftigte sich diese Arbeit. Daher war das Ziel der Diplomarbeit die Erstellung eines Monitoringskonzepts für virtuelle Organisationen in Globus-basierten Grids.

Um die Aufgabenstellung zu bearbeiten, wurden in Kapitel 2 zunächst wichtige Begriffe im Kontext des Grid Computing erläutert. Es wurden dabei die verschiedenen Ausprägungen heutiger Grid-Architekturen und Grid-Middleware dargestellt und wichtige Eigenschaften der VO-Monitoring- und Management Tools aufgezeigt. Es folgten im Kapitel 3 die Anforderungsanalyse und die Erstellung eines Anforderungskatalogs für ein VO-Monitoringkonzept. Dabei wurde zunächst ein Beispielszenario beschrieben, das für weitere Diskussionen in der Arbeit verwendet wurde, um die funktionalen sowie die nicht-funktionalen Anforderungen zu erstellen. Im Kapitel 4 wurde die Ist-Situation des VO-Monitoring dargestellt. Hierbei wurden vorhandene Ansätze des VO-Monitoring und Management vorgestellt und anschließend mit dem erstellten Katalog bewertet. Es handelte sich dabei um vorhandene Tools für das Monitoring von Grid-Ressourcen und das Management von VOs. Es hat sich daher gezeigt, dass die aktuellen Ansätze und Tools die Anforderungen nicht vollständig erfüllen können. Das neue Konzept erfasst im Vergleich zu diesen bestehenden Ansätzen alle möglichen Aktivitäten und Elementarten in einer VO. Zum Beispiel statt nur die Statusinformationen über Grid-Ressourcen zur Verfügung zu stellen, wie es der Fall bei MDS (siehe Kapitel 4.2) ist, wird das neue Konzept zusätzlich Informationen über die VO-Mitglieder und ihre Beziehungen zu den Grid-Ressourcen liefern.

Aus den Anforderungen aufbauend konnte dann im Kapitel 5 das Konzept für VO-Monitoring erarbeitet werden. Das Konzept, das als Integrationskonzept anzusehen ist, basiert hauptsächlich auf der Integration vorher analysierter bestehender Ansätze zum Erreichen der angestrebten Ziele. Für die Realisierung des Konzepts wurde ein prozess-orientierter Ansatz im Rahmen einer SOA-Architektur verwendet. Vorteil dieses Ansatzes ist, dass er die Möglichkeit bietet, Dienste zu orchestrieren. Diese Dienste stellen die Zugriffschnittstellen zu den Tools (die integriert werden sollen) dar.

Kapitel 6 befasste sich mit der Implementierung des Konzepts. Mit dieser Implementierung konnte dann das im Kapitel 3 vorgestellte Szenario getestet und bewertet werden. Die entwickelte Anwendung zeigte sich nach der Bewertung im Kapitel 7 zuverlässig, stabil und robust.

8.2 Ausblick

Die primär angestrebten Ziele der Arbeit waren das Prüfen für die Notwendigkeit eines VO-Monitoringkonzepts und danach die Realisierung dieses Konzepts unter der Bedingung, dass kein anderes Konzept die angestrebten Anforderungen erfüllte.
Das VO-Monitoringkonzept sollte zudem mit der neuen Grid-Architektur (OGSA) kompatibel sein und OGSA-Dienste unterstützen. Dies ist gelungen mit dem Einsatz von Globus Toolkit, das nach OGSA aufgebaut ist und derzeit als de facto Grid-Middleware gilt. Hierbei wurde das MDS, die Resource Management Komponente

von Globus Toolkit, verwendet. OGSA bietet noch keine Standard-Ansätze für das Management von VOs. Daher waren wir in dieser Arbeit frei bei der Auswahl eines Tools für VO-Management. VOMS wurde wegen seiner Einfachheit und seiner übersichtlichen Art der Zuteilung von VO-Rechten (VOMS-Attribute) gewählt. Das erstellte Konzept lehnt sich daher stark an die Struktur von VOMS an. Dieser Einsatz begrenzt den VO-Monitor, indem er nur VOs überwachen kann, die VOMS verwenden. Es wurde bereits im Kapitel 5.5 kurz erläutert, wie das VO-Monitoring für andere VO-Management Systeme angepasst werden kann.

Vor diesem Hintergrund könnte das Konzept erweitert werden, so dass alle VO-Management Systeme unterstützt werden und daher alle VOs, in denen die Ressourcen durch Globus Toolkit bereitgestellt wurden, durch den erstellten VO-Monitor überwacht werden können. Dies kann realisiert werden, indem:

- ein abstraktes Schema für die Zuteilung der VO-Rechte definiert wird. Der VO-Monitor wird dann nur dieses Schema erkennen, verwenden und darüber hinaus Ressourcen und Mitglieder in der VO richtig lokalisieren können. Hierbei werden die VOMS-Attribute durch dieses Schema ersetzt.
- ein Modul definiert wird, das die Autorisierungsinformationen anderer Tools (z.B. VOMS-Attribute oder die SAML-Attribute) in dieses abstrakte Schema konvertiert.

Zudem kann die entwickelte Anwendung im Rahmen anderer Arbeiten wie FoPras oder SEPs erweitert werden. Folgende Änderungen wären noch wünschenswert:

- Refactoring des Datenbankschemas und des Datenbank Services zum effizienteren gleichzeitigen Monitoring mehrerer VOs. Hierbei könnten beispielsweise kommerzielle leistungsfähige Datenbank Management Systeme (DBMS) eingesetzt werden.
- Implementierung eines Frontends zur Visualisierung der Monitordaten. Das könnte z.B. eine Web-Anwendung sein.
- Implementierung von Authentifizierung- und Autorisierungsverfahren.

A Installation und Bedienung der Applikation

A.1 Verzeichnisstruktur der beigefügten CD

Die beigefügte CD besteht aus folgenden Verzeichnissen:

plugin In diesem Verzeichnis liegen die Quellcodes und die Bibliotheken des Plugins. Es besteht aus folgenden Unterverzeichnissen:

- *src* Der Quellcode des Plugins
- *etc* Die XML-Schemas der Konfigurationsdatei und erzeugten Dokumente
- *lib-export* Eine Bibliothek zur Installation des Plugins in Globus Toolkit 4
- *lib* Zusätzliche Bibliotheken für die Installation des Plugins

bpel In diesem Verzeichnis liegen der BPEL-Prozess und die WSDLs anderer Services. Es besteht aus folgenden Unterverzeichnissen:

- *bpel* Der BPEL-Quellcode
- *deployment* Der BPEL-Deployment Descriptor für die ActiveBPEL Engine
- *new-schema* Die WSDLs der Monitor/Datenbank Services
- *existing-schema* Die WSDLs und XML-Schemas der Index/VOMSAdmin Services

dbservice In diesem Verzeichnis liegen das Datenbank Schema und die Datenbank-Service-Applikation. Es besteht aus folgenden Unterverzeichnissen:

- *database* Die komplette Datenbank in SQL-Datei
- *src* Der Quellcode des Datenbank-Services
- *webContent* Die Datenbank-Service Applikation

console-app In diesem Verzeichnis liegt eine Applikation (Command Line - CLI) zum Bedienen des VO-Monitors. Es hat folgenden Unterverzeichnissen:

- *src* Der Quellcode der CLI
- *etc* Die Konfigurationsdatei der CLI
- *lib* Bibliotheken für die CLI

A.2 Installationsanleitung

A.2.1 Das VO-Monitor System

In diesem Abschnitt werden die Installation und die Konfiguration des VO-Monitoring Systems (siehe Abbildung 6.1 im Kapitel 6.1) in einem Linux-System beschrieben.
Das VO-Monitoring System besteht aus einem Server und einem Datenbank Service. Beide Komponenten können auf derselben oder verschiedenen Maschinen installiert werden.

A.2.1.1 ActiveBPEL-Engine

Installation Für die Installation der ActiveBPEL Engine wird die Source-Distribution aus der ActiveEndpoints-Downloadseite (http://www.active-endpoints.com/active-bpel-engine-download.htm) heruntergeladen. In dieser Arbeit wurde die *Version 3.0.3* verwendet und auf Apache Tomcat 5.5 installiert. Apache Tomcat (`http://tomcat.apache.org/`) muss vorinstalliert sein. Nach der Installation und Konfiguration von Tomcat muss die Umgebungsvariable `$CATALINA_HOME` gesetzt werden. In dieser Variable wird der Pfad zum Tomcat-Verzeichnis gespeichert. Die Installationsroutine von ActiveBPEL wird dann diese Variable verwenden, um die Engine in Tomcat zu installieren. Die Engine wird mit dem Tomcat-Container mitgestartet.

Konfiguration Für die Bereitstellung des VO-Monitor-Prozesses (BPEL-Prozess) soll die Datei `CD/bpel/deployment/processDeploy.bpr` im Verzeichnis *bpr* der Tomcat-Installation kopiert werden. Dann muss die Engine wie in der Abbildung A.1 konfiguriert werden. Eine Dokumentation für die Konfiguration der Engine ist ausserdem auf der Seite `http://www.active-endpoints.com/open-source-documentation.htm` zu finden.

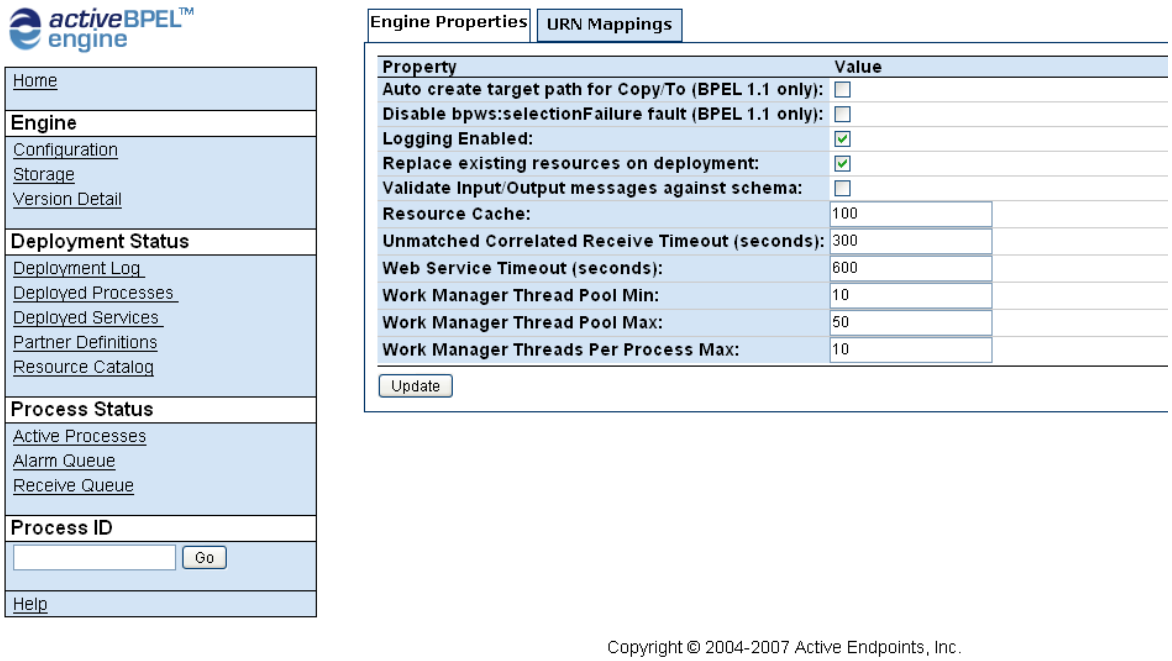


Abbildung A.1: ActiveBPEL Konfiguration

A.2.1.2 Datenbank Service

Installation Für die Installation des Datenbank Services werden folgende Software benötigt:

- `Apache Tomcat Version 5.5` (`http://tomcat.apache.org/download-55.cgi`)
- `Apache Axis Versions 1.4` (`http://ws.apache.org/axis/`)
- `MySQL Datenbank Version 5.0` (`http://dev.mysql.com/doc/refman/5.0/en/index.html`)
- `Hibernate Version 3.2` (`http://www.hibernate.org/5.html`)

Konfiguration :

- Das **Datenbank Schema** kann mit dem SQL-Code (`CD/dbservice/database/database-Setup.sql`) angelegt werden.
- Für die Einrichtung des Datenbank Services muss den kompletten Verzeichnis `CD/dbservice/-WebContent/` ins *webapps*-Verzeichnis von Tomcat kopiert werden. Zuvor müssen aber folgende Dateien konfiguriert werden:
  - `dbservice/WebContent/WEB-INF/classes/hibernate.cfg.xml` - Zugriff von Hibernate auf die Datenbank
  - `dbservice/WebContent/META-INF/context.xml` - Kontextkonfiguration des Datenbank Services

A.2.1.3 Globus Toolkit

Siehe [GLO05]

A.2.1.4 VOMS

Siehe [VOMS05]

A.2.2 MDS-Plugin

Installation Für die Installation des Plugins müssen einpaar Dateien wie folgt kopiert werden (Die Umgebungsvariable `$GLOBUS_LOCATION` speichert den Pfad zur Globus Installation):

- Kopieren von `CD/plugin/lib-export/VOResourcePlugin.jar` in Verzeichnis `$GLOBUS_LOCATION/lib`
- Kopieren von `CD/plugin/lib/*` in Verzeichnis `$GLOBUS_LOCATION/lib`
- Kopieren von `CD/plugin/etc/VOResMappingConfSchema.xsd` in Verzeichnis `$GLOBUS_LOCATION/libexec/aggrexec`
- Kopieren von `CD/plugin/etc/aggregator-exec-vomonitor.sh` in Verzeichnis `$GLOBUS_LOCATION/libexec/aggrexec`

Konfiguration und Bedienung Für die Konfiguration und die Bedienung des Plugins siehe die Execution Framework Dokumentation auf der Globus-Webseite:

- http://www.globus.org/toolkit/docs/4.0/info/aggregator/rn01re01.html
- http://www.globus.org/toolkit/docs/4.0/info/aggregator/Execution_Aggregator_Source.html

B WS-BPEL Spezifikation

BPEL ist für Prozess- und Workflow-Techniken das, was SQL für die relationalen Datenbanken ist, so steht es im aktuellen Präsentationsmaterial von Oasis.
Bei WS-BPEL (Web Services Business Execution Language) handelt es sich um eine ausführbare, XML-basierende Sprache zur Beschreibung von Geschäftsprozessen, deren Aktivitäten durch Web-Services implementiert werden und deren Nachrichtenaustausch über XML-Dokumente erfolgt. Das Besondere daran: WS-BPEL bietet eine homogene Syntax für die Ablaufbeschreibung und den Datenzugriff auf XML-Dokumente. Und es enthält Elemente, die speziell auf die Ablaufproblematik langlaufender Geschäftsprozesse mit mehreren Partnern zugeschnitten sind. So können die Aktivitäten eines Geschäftsprozesses in Scopes, das heißt in kontextorientierten, transaktionalen Einheiten zusammengefasst werden. Für den Fehlerfall, bei dem bereits abgeschlossene Scopes zurückgesetzt werden müssen - man spricht von Kompensation - , enthält WS-BPEL mächtige syntaktische Konstrukte. Hier ist WS-BPEL anderen Programmiersprachen klar überlegen - in der Theorie.
Dieses Kapitel enthält Auszüge der WS-BPEL 2.0-Spezifikation. Für eine detaillierte Beschreibung siehe [BPELv2.0].

Die Beschreibung eines Prozesses erfolgt innerhalb eines `process`-Elements, dessen allgemeine Grammatik gemäß WS-BPEL 2.0 [BPELv2.0] folgendermaßend aussieht:

```
<process name="NCName" targetNamespace="anyURI"
  queryLanguage="anyURI"?
  expressionLanguage="anyURI"?
  suppressJoinFailure="yes|no"?
  exitOnStandardFault="yes|no"?
  xmlns="http://docs.oasis-open.org/wsbpel/2.0/process/executable">


  <extensions>?
    <extension namespace="anyURI" mustUnderstand="yes|no" />+
  </extensions>


  <import namespace="anyURI"?
    location="anyURI"?
    importType="anyURI" />*


  <partnerLinks>?
    <!-- Note: At least one role must be specified. -->
    <partnerLink name="NCName"
      partnerLinkType="QName"
      myRole="NCName"?
      partnerRole="NCName"?
      initializePartnerRole="yes|no"?>+
    </partnerLink>
  </partnerLinks>


  <messageExchanges>?
    <messageExchange name="NCName" />+
  </messageExchanges>
```

```
  <variables>?
    <variable name="BPELVariableName"
      messageType="QName"?
      type="QName"?
      element="QName"?>+
      from-spec?
    </variable>
  </variables>


  <correlationSets>?
    <correlationSet name="NCName" properties="QName-list" />+
  </correlationSets>


  <faultHandlers>?
    <!-- Note: There must be at least one faultHandler -->
    <catch faultName="QName"?
      faultVariable="BPELVariableName"?
      ( faultMessageType="QName" | faultElement="QName" )? >*
      <!-- activity -->
    </catch>
    <catchAll>?
      <!-- activity -->
    </catchAll>
  </faultHandlers>


  <eventHandlers>?
    <!-- Note: There must be at least one onEvent or onAlarm. -->
    <onEvent partnerLink="NCName"
      portType="QName"?
      operation="NCName"
      ( messageType="QName" | element="QName" )?
      variable="BPELVariableName"?
      messageExchange="NCName"?>*
      <correlations>?
        <correlation set="NCName" initiate="yes|join|no"? />+
      </correlations>
      <fromParts>?
        <fromPart part="NCName" toVariable="BPELVariableName" />+
      </fromParts>
      <scope ...>...</scope>
    </onEvent>
    <onAlarm>*
      <!-- Note: There must be at least one expression. -->
      (
      <for expressionLanguage="anyURI"?>duration-expr</for>
      |
      <until expressionLanguage="anyURI"?>deadline-expr</until>
      )?
      <repeatEvery expressionLanguage="anyURI"?>
        duration-expr
      </repeatEvery>?
      <scope ...>...</scope>
    </onAlarm>
  </eventHandlers>
  <!-- activity -->
</process>
```

PartnerLinks:
Ein BPEL-Prozess nutzt Dienste in Form von Web Services. Die Dienste werden von Dritten - so genannten Partnern - zur Verfügung gestellt. Mit Hilfe des Elements `<partnerLinks>` wird definiert, wie ein Prozess mit seinen Partnern kommuniziert. Prinzipiell kann ein Partner mehrere Rollen zu einem anderen Partner einnehmen. Die Rolle hängt davon ab, welchen Zweck die Kommunikation erfüllt. Folgendes Listing stellt die Syntax des `<partnerLinks>`-Elements dar.
Jede Kommunikationsverbindung mit einem Partner muss über ein Element `partnerLink` definiert werden. Jede Verbindung erhält über das Attribut `name` einen eindeutigen Bezeichner. Für jede Rolle wird festgelegt welche Rolle der eigene Prozess (`myRole`) und welche der externe Partner (`partnerRole`) übernimmt. Die Rollen werden im entsprechenden `PartnerLinkType` definiert.

```
<partnerLinks>
      <partnerLink name="ncname" partnerLinkType="qname" myRole="ncname"?
          partnerRole="ncname"?>
      </partnerLink>+
</partnerLinks>
```

PartnerLinkType:
`PartnerLinkType` spezifiziert die Rollen, die in einer Beziehung zwischen zwei Services vorkommen. Zusätzlich wird zu jeder Rolle der `portType` spezifiziert [WSDLv1.1]. Partner Link Types sind in WSDL mit einem speziellen Namespace definiert:
http://schemas.xmlsoap.org/ws/2003/05/partner-link/.
Im folgenden ist ein Extrakt einer PartnerLink-Definition aus dem WSDL des Datenbank Services (siehe Kapitel 6.2) aufgelistet.

```
<plnk:partnerLinkType name="DBServiceLT">
            <!-- tns:DBService ist ein PortType aus dem DBService-WSDL -->
            <plnk:role name="Database" portType="tns:DBService"/>
</plnk:partnerLinkType>
```

FaultHandlers:

```
<faultHandlers>
  <catch faultName="QName"?
    faultVariable="BPELVariableName"?
    ( faultMessageType="QName" | faultElement="QName" )? >*
    <!-- activity -->
  </catch>
  <catchAll>?
    <!-- activity -->
  </catchAll>
</faultHandlers>
```

`<!-- activity -->` steht für die primitiven und strukturierten Aktivitäten. Primitiven Aktivitäten beschreiben die elementaren Schritte eines BPEL-Prozesses. Die strukturierten Aktivitäten beschreiben Kontrollkonstrukte und können rekursiv mehrere primitiven oder strukturierten Aktivitäten enthalten.

Primitive Activitäten in WS-BPEL 2.0:

- `<receive>*` Wartet, dass ein Client den Prozess durch Senden einer Nachricht aktiviert.
- `<reply>*` Antwort für eine synchrone Operation erzeugen.
- `<invoke>*` Aufruf eines anderen Web Services.
- `<assign>*` Manipulation von Variablenwerten.
- `<validate>` XML-Daten, die in Variablen liegen, validieren.
- `<empty>` Aktivität, die nichts tut.
- `<throw>*` Anzeigen von Fehlern oder Ausnahmen.

- `<rethrow>*` eine Ausnahme oder einen Fehler innerhalb einem Fault-Handler weiterleiten.
- `<exit>*` den Prozess beenden.
- `<wait>*` eine vorgegebene Zeit abwarten.
- `<compensate>` eine Kompensation-Aktivität aufrufen.
- `<extensionActivity>` Wrapper für WS-BPEL-Erweiterung.

Strukturierte Activitäten in WS-BPEL 2.0:

- `<flow>` Parallele Ausführung von Aktivitäten.
- `<sequence>*` Sequenz von Aktivitäten.
- `<while>*` Schleife.
- `<repeatUntil>*` die enthaltene Aktivitäten werden ausgeführt, bis eine bestimmte Bedingung erfüllt ist.
- `<pick>` Auswahl von Alternativen aufgrund externer Ereignisse (nicht-deterministische Verzweigungen).
- `<forEach>*` enthaltene Aktivitäten werden wiederholt ausgeführt, kontrolliert durch einen Zähler.
- `<if then else>*` Verzweigung.
- `<scope>*` Ausfuhrungskontext fur Aktivitaten mit der Moglichkeit, eigene Variablen, PartnerLinks, Fault, Compensation, Termination und Event Handler zu definieren.

Im Syntax der mit (*) gekennzeichneten Aktivitäten wird im Folgenden genauer betrachtet:

receive:

```
<receive partnerLink="NCName"
  portType="QName"?
  operation="NCName"
  variable="BPELVariableName"?
  createInstance="yes|no"?
  messageExchange="NCName"?
  standard-attributes>
  standard-elements
  <correlations>?
    <correlation set="NCName" initiate="yes|join|no"? />+
  </correlations>
  <fromParts>?
    <fromPart part="NCName" toVariable="BPELVariableName" />+
  </fromParts>
</receive>
```

reply:

```
<reply partnerLink="NCName"
  portType="QName"? operation="NCName"
  variable="BPELVariableName"?
  faultName="QName"?
  messageExchange="NCName"?
  standard-attributes>
  standard-elements
  <correlations>?
    <correlation set="NCName" initiate="yes|join|no"? />+
  </correlations>
  <toParts>?
    <toPart part="NCName" fromVariable="BPELVariableName" />+
```

```
  </toParts>
</reply>
```

invoke:

```
<invoke partnerLink="NCName"
  portType="QName"?
  operation="NCName"
  inputVariable="BPELVariableName"?
  outputVariable="BPELVariableName"?
  standard-attributes>
  standard-elements
  <correlations>?
    <correlation set="NCName" initiate="yes|join|no"?
      pattern="request|response|request-response"? />+
  </correlations>
  <catch faultName="QName"?
    faultVariable="BPELVariableName"?
    faultMessageType="QName"?
    faultElement="QName"?>*
    <!-- activity -->
  </catch>
  <catchAll>?
    <!-- activity -->
  </catchAll>
  <compensationHandler>?
    <!-- activity -->
  </compensationHandler>
  <toParts>?
    <toPart part="NCName" fromVariable="BPELVariableName" />+
  </toParts>
  <fromParts>?
    <fromPart part="NCName" toVariable="BPELVariableName" />+
  </fromParts>
</invoke>
```

assign:

```
<assign validate="yes|no"? standard-attributes>
  standard-elements
  (
  <copy keepSrcElementName="yes|no"? ignoreMissingFromData="yes|no"?>
    from-spec to-spec
  </copy>
  |
  <extensionAssignOperation>
    assign-element-of-other-namespace
  </extensionAssignOperation>
  )+
</assign>
```

assign from-spec:

```
<from variable="BPELVariableName" part="NCName"?>
  <query queryLanguage="anyURI"?>?
    queryContent
  </query>
</from>
<from partnerLink="NCName" endpointReference="myRole|partnerRole" />
<from variable="BPELVariableName" property="QName" />
<from expressionLanguage="anyURI"?>expression</from>
```

```
<from><literal>literal value</literal></from>
<from/>
```

assign to-spec:

```
<to variable="BPELVariableName" part="NCName"?>
  <query queryLanguage="anyURI"?>?
    queryContent
  </query>
</to>
<to partnerLink="NCName" />
<to variable="BPELVariableName" property="QName" />
<to expressionLanguage="anyURI"?>expression</to>
<to/>
```

throw:

```
<throw faultName="QName" faultVariable="BPELVariableName"?
  standard-attributes>
  standard-elements
</throw>
```

retrow:

```
<rethrow standard-attributes>
  standard-elements
</rethrow>
```

exit:

```
<exit standard-attributes>
  standard-elements
</exit>
```

wait:

```
<wait standard-attributes>
  standard-elements
  (
  <for expressionLanguage="anyURI"?>duration-expr</for>
  |
  <until expressionLanguage="anyURI"?>deadline-expr</until>
  )
</wait>
```

sequence:

```
<sequence standard-attributes>
  standard-elements
  <!-- activity -->+
</sequence>
```

while:

```
<while standard-attributes>
  standard-elements
  <condition expressionLanguage="anyURI"?>bool-expr</condition>
  <!-- activity -->
</while>
```

repeatUntil:

```
<repeatUntil standard-attributes>
  standard-elements
 <!-- activity -->
  <condition expressionLanguage="anyURI"?>bool-expr</condition>
</repeatUntil>
```

forEach:

```
<forEach counterName="BPELVariableName" parallel="yes|no"
  standard-attributes>
  standard-elements
  <startCounterValue expressionLanguage="anyURI"?>
    unsigned-integer-expression
  </startCounterValue>
  <finalCounterValue expressionLanguage="anyURI"?>
    unsigned-integer-expression
  </finalCounterValue>
  <completionCondition>?
    <branches expressionLanguage="anyURI"?
      successfulBranchesOnly="yes|no"?>?
      unsigned-integer-expression
    </branches>
  </completionCondition>
  <scope ...>...</scope>
</forEach>
```

if then else:

```
<if standard-attributes>
  standard-elements
  <condition expressionLanguage="anyURI"?>bool-expr</condition>
  <!-- activity -->
  <elseif>*
    <condition expressionLanguage="anyURI"?>bool-expr</condition>
    <!-- activity -->
  </elseif>
  <else>?
    <!-- activity -->
  </else>
</if>
```

scope:

```
<scope isolated="yes|no"? exitOnStandardFault="yes|no"?
  standard-attributes>
  standard-elements
  <variables>?
    ...
  </variables>
  <partnerLinks>?
    ...
  </partnerLinks>
  <messageExchanges>?
    ...
  </messageExchanges>
  <correlationSets>?
    ...
  </correlationSets>
  <eventHandlers>?
    ...
  </eventHandlers>
```

```
  <faultHandlers>?
    ...
  </faultHandlers>
  <compensationHandler>?
    ...
  </compensationHandler>
  <terminationHandler>?
    ...
  </terminationHandler>
  <!-- activity -->
</scope>
```

C BPEL-Prozess des VO-Monitor

In diesem Kapitel werden die Prozessaktivitäten aus dem Kapitel 6.3.1 komplett aufgelistet. Für die Beschreibung der verwendeten BPEL-Syntax siehe Anhang B.

Listing C.1: Prozess-Aktivität 2

```
<bpel:wait name="WaitTheDelay">
   <bpel:for expressionLanguage="urn:oasis:names:tc:wsbpel:2.0:sublang:xpath1.0">
      concat( 'PT', string( $dbGetLoopDelayResponse.timeMillis ), 'S')
   </bpel:for>
</bpel:wait>
```

Listing C.2: Prozess-Aktivität 3

```
ï»¿<!-------------------------------------------------------------->
<!-------------------- AKTIVITAET 3 ---------------------------->
<!-------------------------------------------------------------->
<bpel:scope name="VOMS">
<bpel:variables>
<bpel:variable messageType="voms:getVONameRequest" name="GetVONameRequiest" />
<bpel:variable messageType="voms:getVONameResponse" name="GetVONameResponse" />
<bpel:variable messageType="db:insertGroupsRequest" name="insertGroups4DBRequest"
    />
<bpel:variable messageType="db:insertGroupsResponse" name="
   insertGroups4DBResponse" />
<bpel:variable messageType="voms:listRolesRequest" name="listRolesRequest" />
<bpel:variable messageType="voms:listRolesResponse" name="listRolesResponse" />
<bpel:variable messageType="db:insertRolesRequest" name="dbInsertRolesRequest" />
<bpel:variable messageType="db:insertRolesResponse" name="dbInsertRolesResponse"
    />
<bpel:variable messageType="voms:listMembersRequest" name="listUsersRequest" />
<bpel:variable messageType="voms:listMembersResponse" name="listUsersReponse" />
<bpel:variable messageType="voms:listCapabilitiesRequest" name="
   listCapabilitiesRequest" />
<bpel:variable messageType="voms:listCapabilitiesResponse" name="
   listCapabilitiesResponse" />
<bpel:variable messageType="db:insertUsersRequest" name="dbInsertUsersRequest" />
<bpel:variable messageType="db:insertUsersResponse" name="dbInsertUsersResponse"
    />
<bpel:variable messageType="db:insertCapabilitiesRequest" name="
   dbInsertCapabilitiesRequest" />
<bpel:variable messageType="db:insertCapabilitiesResponse"name="
   dbInsertCapabilitiesresponse" />
</bpel:variables>
<bpel:faultHandlers><!-- Fehlerbehandlungsroutinen --></bpel:faultHandlers>
<bpel:sequence>
<bpel:invoke inputVariable="GetVONameRequiest" name="GetVONameTesting" operation=
   "getVOName" outputVariable="GetVONameResponse" partnerLink="
   VOMSAdminPartnerLink" portType="voms:VOMSAdmin" />
<bpel:scope name="getAndSetAllVOGroups">
<bpel:variables>
<bpel:variable name="dynamicGroupListFromVoms" type="xsd:string" />
<bpel:variable name="currentGroupname" type="xsd:string" />
<bpel:variable messageType="voms:listSubGroupsRequest" name="
   listVOSubGroupsRequest" />
```

```
<bpel:variable messageType="voms:listSubGroupsResponse" name="
    listVOSubGroupsResponse" />
<bpel:variable messageType="db:insertGroupRequest" name="dbInsertGroupRequest" />
<bpel:variable messageType="db:insertGroupResponse" name="dbInsertGroupResponse"
     />
</bpel:variables>
<bpel:faultHandlers><!-- Fehlerbehandlungsroutine --></bpel:faultHandlers>
<bpel:sequence>
<bpel:assign name="initGroupLists">
<bpel:copy>
<bpel:from>concat($GetVONameResponse.getVONameReturn,'$')</bpel:from>
<bpel:to variable="dynamicGroupListFromVoms" />
</bpel:copy>
</bpel:assign>
<bpel:while>
<bpel:condition expressionLanguage="urn:oasis:names:tc:wsbpel:2.0:sublang:xpath1
    .0">boolean( string-length($dynamicGroupListFromVoms) &gt; 0)</bpel:condition
    >
<bpel:sequence>
<bpel:assign name="getCurrentGroupname">
<bpel:copy>
<bpel:from>substring-before($dynamicGroupListFromVoms,'$')</bpel:from>
<bpel:to variable="currentGroupname" /></bpel:copy>
</bpel:assign>
<bpel:assign name="actualizeDynamicGroupListFromVoms">
<bpel:copy>
<bpel:from>substring-after($dynamicGroupListFromVoms , concat($currentGroupname ,
    '$') )</bpel:from>
<bpel:to variable="dynamicGroupListFromVoms" />
</bpel:copy>
</bpel:assign>
<bpel:assign name="initGetSubGroupsRequest">
<bpel:copy>
<bpel:from variable="currentGroupname" />
<bpel:to part="groupname" variable="listVOSubGroupsRequest" />
</bpel:copy>
</bpel:assign>
<bpel:invoke inputVariable="listVOSubGroupsRequest" name="listVOSubGroups"
    operation="listSubGroups" outputVariable="listVOSubGroupsResponse"
    partnerLink="VOMSAdminPartnerLink" portType="voms:VOMSAdmin" />
<bpel:forEach counterName="counter" name="ForEach-ProcessSubGroups" parallel="no"
    >
<bpel:startCounterValue expressionLanguage="urn:oasis:names:tc:wsbpel:2.0:sublang
    :xpath1.0">1
</bpel:startCounterValue>
<bpel:finalCounterValue expressionLanguage="urn:oasis:names:tc:wsbpel:2.0:sublang
    :xpath1.0">count($listVOSubGroupsResponse.listSubGroupsReturn/item)</bpel:
    finalCounterValue>
<bpel:scope>
<bpel:variables>
<bpel:variable name="groupAtIndexCount" type="xsd:string" />
</bpel:variables>
<bpel:sequence>
<bpel:assign name="getCurrentName">
<bpel:copy>
<bpel:from>$listVOSubGroupsResponse.listSubGroupsReturn/item[position()=$counter
    ]</bpel:from>
<bpel:to>$groupAtIndexCount</bpel:to>
</bpel:copy>
</bpel:assign>
<bpel:assign name="actualiseDynamicGroupList">
```

```
<bpel:copy>
<bpel:from>concat($dynamicGroupListFromVoms, $groupAtIndexCount,'$')</bpel:from>
<bpel:to variable="dynamicGroupListFromVoms" />
</bpel:copy>
</bpel:assign>
</bpel:sequence>
</bpel:scope>
</bpel:forEach>
<bpel:assign name="initDbSetCurrentGroupRequest">
<bpel:copy>
<bpel:from variable="voID" />
<bpel:to part="voId" variable="dbInsertGroupRequest" />
</bpel:copy>
<bpel:copy>
<bpel:from variable="currentGroupname" />
<bpel:to part="groupname" variable="dbInsertGroupRequest" />
</bpel:copy>
</bpel:assign>
<bpel:invoke inputVariable="dbInsertGroupRequest" name="dbInsertCurrentGroup"
    operation="insertGroup" outputVariable="dbInsertGroupResponse" partnerLink="
    DatabasePartnerLink" portType="db:DBService" />
</bpel:sequence>
</bpel:while>
</bpel:sequence>
</bpel:scope>
<bpel:invoke inputVariable="listRolesRequest" name="listRoles" operation="
    listRoles" outputVariable="listRolesResponse" partnerLink="
    VOMSAdminPartnerLink" />
<bpel:assign name="initDBInsertRolesRequest">
<bpel:copy>
<bpel:from variable="voID" />
<bpel:to part="voId" variable="dbInsertRolesRequest" />
</bpel:copy>
<bpel:copy>
<bpel:from part="listRolesReturn" variable="listRolesResponse" />
<bpel:to part="rolenameList" variable="dbInsertRolesRequest" />
</bpel:copy>
</bpel:assign>
<bpel:invoke inputVariable="dbInsertRolesRequest" name="dbInsertRoles" operation=
    "insertRoles" outputVariable="dbInsertRolesResponse" partnerLink="
    DatabasePartnerLink" portType="db:DBService" />
<bpel:assign name="initGetUsersRequest">
<bpel:copy>
<bpel:from>$GetVONameResponse.getVONameReturn</bpel:from>
<bpel:to part="groupname" variable="listUsersRequest" />
</bpel:copy>
</bpel:assign>
<bpel:invoke inputVariable="listUsersRequest" name="getUsers" operation="
    listMembers" outputVariable="listUsersReponse" partnerLink="
    VOMSAdminPartnerLink" portType="voms:VOMSAdmin" />
<bpel:assign name="initDbInsertUsersRequest">
<bpel:copy>
<bpel:from variable="voID" />
<bpel:to part="voId" variable="dbInsertUsersRequest" />
</bpel:copy>
<bpel:copy>
<bpel:from part="listMembersReturn" variable="listUsersReponse" />
<bpel:to part="users" variable="dbInsertUsersRequest" />
</bpel:copy>
</bpel:assign>
```

```
<bpel:invoke inputVariable="dbInsertUsersRequest" name="dbInsertUsers" operation=
    "insertUsers" outputVariable="dbInsertUsersResponse" partnerLink="
    DatabasePartnerLink" portType="db:DBService" />
<bpel:invoke inputVariable="listCapabilitiesRequest" name="listCapabilities"
    operation="listCapabilities" outputVariable="listCapabilitiesResponse"
    partnerLink="VOMSAdminPartnerLink" portType="voms:VOMSAdmin" />
<bpel:assign name="initDbInsertCapabitilitesRequest">
<bpel:copy>
<bpel:from variable="voID" />
<bpel:to part="voId" variable="dbInsertCapabilitiesRequest" />
</bpel:copy>
<bpel:copy>
<bpel:from part="listCapabilitiesReturn" variable="listCapabilitiesResponse" />
<bpel:to part="capsName" variable="dbInsertCapabilitiesRequest" />
</bpel:copy>
</bpel:assign>
<bpel:invoke inputVariable="dbInsertCapabilitiesRequest" name="
    dbInsertCapabilities" operation="insertCapabilities" outputVariable="
    dbInsertCapabilitiesresponse" partnerLink="DatabasePartnerLink" portType="db:
    DBService" />
<bpel:forEach counterName="counter" name="ForEach-User-Set-GroupAndRoles"
    parallel="no">
<bpel:startCounterValue expressionLanguage="urn:oasis:names:tc:wsbpel:2.0:sublang
    :xpath1.0">1
</bpel:startCounterValue>
<bpel:finalCounterValue expressionLanguage="urn:oasis:names:tc:wsbpel:2.0:sublang
    :xpath1.0">count( $listUsersReponse.listMembersReturn/item)</bpel:
    finalCounterValue>
<bpel:scope>
<bpel:variables>
<bpel:variable name="currentUser" type="db:User" />
<bpel:variable messageType="voms:listGroupsRequest" name="listUserGroupsRequest"
     />
<bpel:variable messageType="voms:listGroupsResponse" name="listUserGroupsResponse
    " />
<bpel:variable messageType="db:setVOUserGroupsRequest" name="
    dbSetVOUserGroupsRequest" />
<bpel:variable messageType="db:setVOUserGroupsResponse" name="
    dbSetVOUserGroupsResponse" />
<bpel:variable messageType="voms:listRolesRequest1"name="listUserRolesRequest" />
<bpel:variable messageType="voms:listRolesResponse" name="listUserRolesResponse"
     />
<bpel:variable messageType="db:setVOUserRolesRequest" name="
    dbSetVOUserRolesRequest" />
<bpel:variable messageType="db:setVOUserRolesResponse" name="
    dbSetVOUserRolesResponse" />
<bpel:variable messageType="voms:listCapabilitiesRequest1" name="
    listUserCapabilitiesRequest" />
<bpel:variable messageType="voms:listCapabilitiesResponse" name="
    listUserCapabilitiesResponse" />
<bpel:variable messageType="db:setUserCapabilitiesRequest" name="
    dbSetUserCapabilitiesRequest" />
<bpel:variable messageType="db:setUserCapabilitiesResponse" name="
    dbSetUserCapabilitiesResponse" />
</bpel:variables>
<bpel:faultHandlers>.....</bpel:faultHandlers>
<bpel:sequence>
<bpel:assign name="getCurrentUser">
<bpel:copy>
<bpel:from>$listUsersReponse.listMembersReturn/item[position()=$counter]</bpel:
    from>
```

```
<bpel:to>$currentUser</bpel:to>
</bpel:copy>
</bpel:assign>
<bpel:assign name="initListUserGroupsRequest">
<bpel:copy>
<bpel:from variable="currentUser">
<bpel:query>DN</bpel:query>
</bpel:from>
<bpel:to part="username" variable="listUserGroupsRequest" />
</bpel:copy>
<bpel:copy>
<bpel:from variable="currentUser">
<bpel:query>CA</bpel:query>
</bpel:from>
<bpel:to part="userca" variable="listUserGroupsRequest" />
</bpel:copy>
</bpel:assign>
<bpel:invoke inputVariable="listUserGroupsRequest" name="listUserGroups"
    operation="listGroups" outputVariable="listUserGroupsResponse" partnerLink="
    VOMSAdminPartnerLink" />
<bpel:assign name="initDbSetVOUserGroupsRequest">
<bpel:copy>
<bpel:from variable="voID" />
<bpel:to part="voId" variable="dbSetVOUserGroupsRequest" />
</bpel:copy>
<bpel:copy>
<bpel:from variable="currentUser" />
<bpel:to part="user" variable="dbSetVOUserGroupsRequest" />
</bpel:copy>
<bpel:copy>
<bpel:from part="listGroupsReturn" variable="listUserGroupsResponse" />
<bpel:to part="grouplist" variable="dbSetVOUserGroupsRequest" />
</bpel:copy>
</bpel:assign>
<bpel:invoke inputVariable="dbSetVOUserGroupsRequest" name="dbSetVOUserGroups"
    operation="setVOUserGroups" outputVariable="dbSetVOUserGroupsResponse"
    partnerLink="DatabasePartnerLink" portType="db:DBService" />
<bpel:assign name="initListUserRolesRequest">
<bpel:copy>
<bpel:from>$currentUser/DN</bpel:from>
<bpel:to>$listUserRolesRequest.username</bpel:to>
</bpel:copy>
<bpel:copy>
<bpel:from>$currentUser/CA</bpel:from>
<bpel:to>$listUserRolesRequest.userca</bpel:to>
</bpel:copy>
</bpel:assign>
<bpel:invoke inputVariable="listUserRolesRequest" name="listUserRoles" operation=
    "listRoles" outputVariable="listUserRolesResponse" partnerLink="
    VOMSAdminPartnerLink2" portType="voms:VOMSAdmin2" />
<bpel:assign name="initDbSetVOUserRolesRequest">
<bpel:copy>
<bpel:from variable="voID" />
<bpel:to part="voId" variable="dbSetVOUserRolesRequest" />
</bpel:copy>
<bpel:copy>
<bpel:from variable="currentUser" />
<bpel:to part="user" variable="dbSetVOUserRolesRequest" />
</bpel:copy>
<bpel:copy>
<bpel:from part="listRolesReturn" variable="listUserRolesResponse" />
```

```
<bpel:to part="roleList" variable="dbSetVOUserRolesRequest" />
</bpel:copy>
</bpel:assign>
<bpel:invoke inputVariable="dbSetVOUserRolesRequest" name="dbSetVOUserRoles"
    operation="setVOUserRoles" outputVariable="dbSetVOUserRolesResponse"
    partnerLink="DatabasePartnerLink" portType="db:DBService" />
<bpel:assign name="initListUserCapabilitiesRequest">
<bpel:copy>
<bpel:from>$currentUser/DN</bpel:from>
<bpel:to part="username" variable="listUserCapabilitiesRequest" />
</bpel:copy>
<bpel:copy>
<bpel:from>$currentUser/CA</bpel:from>
<bpel:to part="userca" variable="listUserCapabilitiesRequest" />
</bpel:copy>
</bpel:assign>
<bpel:invoke inputVariable="listUserCapabilitiesRequest" name="
    listUserCapabilities" operation="listCapabilities" outputVariable="
    listUserCapabilitiesResponse" partnerLink="VOMSAdminPartnerLink2" portType="
    voms:VOMSAdmin2" />
<bpel:assign name="initDbSetUserCapabilities">
<bpel:copy>
<bpel:from variable="voID" />
<bpel:to part="voId" variable="dbSetUserCapabilitiesRequest" />
</bpel:copy>
<bpel:copy>
<bpel:from variable="currentUser" />
<bpel:to part="user" variable="dbSetUserCapabilitiesRequest" />
</bpel:copy>
<bpel:copy>
<bpel:from part="listCapabilitiesReturn" variable="listUserCapabilitiesResponse"
     />
<bpel:to part="caps" variable="dbSetUserCapabilitiesRequest" />
</bpel:copy>
</bpel:assign>
<bpel:invoke inputVariable="dbSetUserCapabilitiesRequest" name="
    dbSetUserCapabilities" operation="setUserCapabilities" outputVariable="
    dbSetUserCapabilitiesResponse" partnerLink="DatabasePartnerLink" portType="db
    :DBService" />
</bpel:sequence>
</bpel:scope>
</bpel:forEach>
</bpel:sequence>
</bpel:scope>
```

Listing C.3: Prozess-Aktivitäten 4 und 5

```
<!-------------------------------------------------------------->
<!-------------------- AKTIVITÄT 4 ---------------------------->
<!-------------------------------------------------------------->
<bpel:scope name="MDS-IndexService">
<bpel:variables>
<bpel:variable element="wsa:EndpointReference" name="IndexServiceEPR"/>
<bpel:variable messageType="db:getIndexServiceWithLowestPriorityRequest" name="
    dbGetIndexServiceWithLowestPriorityRequest"/>
<bpel:variable messageType="db:getIndexServiceWithLowestPriorityResponse" name="
    dbGetIndexServiceWithLowestPriorityResponse"/>
<bpel:variable messageType="ns3:QueryResourcePropertiesRequest" name="
    QueryRPRequest"/>
<bpel:variable messageType="ns3:QueryResourcePropertiesResponse" name="
    QueryRPResponse"/>
```

```
<bpel:variable name="EntryCount" type="xsd:unsignedInt"/>
</bpel:variables>
<bpel:faultHandlers>....</bpel:faultHandlers>
<bpel:sequence>
<bpel:assign name="initGetPriorityRequestVariable">
<bpel:copy><bpel:from variable="voID"/><bpel:to part="voId" variable="
    dbGetIndexServiceWithLowestPriorityRequest"/>
</bpel:copy>
</bpel:assign>
<bpel:invoke inputVariable="dbGetIndexServiceWithLowestPriorityRequest" name="
    dbGetIndexServiceWithLowestPriority" operation="
    getIndexServiceWithLowestPriority" outputVariable="
    dbGetIndexServiceWithLowestPriorityResponse" partnerLink="DatabasePartnerLink
    " portType="db:DBService"/>
<bpel:assign name="initIndexServiceEPRVariables">
<bpel:copy>
<bpel:from>
<bpel:literal>
<wsa:EndpointReference xmlns:s="http://docs.oasis-open.org/wsrf/2004/06/wsrf-WS-
    ResourceProperties-1.2-draft-01.wsdl/service" xmlns:wsa="http://schemas.
    xmlsoap.org/ws/2004/03/addressing">
<wsa:ReferenceProperties><wsa:ReplyTo><wsa:Address>http://schemas.xmlsoap.org/ws
    /2004/03/addressing/role/anonymous</wsa:Address></wsa:ReplyTo></wsa:
    ReferenceProperties>
<wsa:Address/><wsa:ServiceName PortName="QueryResourcePropertiesPort">s:WS-
    ResourcePropertiesService</wsa:ServiceName></wsa:EndpointReference>
</bpel:literal>
 </bpel:from>
<bpel:to variable="IndexServiceEPR"/>
</bpel:copy>
<bpel:copy>
<bpel:from>$dbGetIndexServiceWithLowestPriorityResponse.uri</bpel:from>
<bpel:to>$IndexServiceEPR/wsa:Address</bpel:to>
</bpel:copy>
</bpel:assign>
<bpel:assign name="setIndexServicePartnerLinksEPR">
<bpel:copy>
<bpel:from variable="IndexServiceEPR"/>
<bpel:to partnerLink="IndexServicePartnerLink"/>
</bpel:copy>
</bpel:assign>
<bpel:assign name="prepareQueryExpr">
<bpel:copy>
<bpel:from>
<bpel:literal>http://www.w3.org/TR/1999/REC-xpath-19991116</bpel:literal>
</bpel:from>
<bpel:to>$QueryRPRequest.QueryResourcePropertiesRequest/wsrp:QueryExpression/
    @Dialect</bpel:to>
</bpel:copy>
<bpel:copy>
<bpel:from>
<bpel:literal>/*</bpel:literal>
</bpel:from>
<bpel:to>$QueryRPRequest.QueryResourcePropertiesRequest/wsrp:QueryExpression</
    bpel:to>
</bpel:copy>
</bpel:assign>
<bpel:invoke inputVariable="QueryRPRequest" name="queryIndexService" operation="
    QueryResourceProperties" outputVariable="QueryRPResponse" partnerLink="
    IndexServicePartnerLink" portType="ns3:QueryResourceProperties"/>
<bpel:assign name="getEntriesCount">
```

```
<bpel:copy>
<bpel:from>count( $QueryRPResponse.QueryResourcePropertiesResponse/*/ns13:Entry )
    </bpel:from>
<bpel:to variable="EntryCount"/>
</bpel:copy>
</bpel:assign>
<bpel:forEach counterName="counter" name="ProcessEachEntry" parallel="no">
<bpel:startCounterValue>1</bpel:startCounterValue>
<bpel:finalCounterValue expressionLanguage="urn:oasis:names:tc:wsbpel:2.0:sublang
    :xpath1.0">$EntryCount</bpel:finalCounterValue>
<bpel:scope>
<bpel:variables>
<bpel:variable messageType="db:setResourceRequest" name="dbSetResourceRequest"/>
<bpel:variable messageType="db:setResourceResponse" name="dbSetResouceResponse"/>
<bpel:variable element="ns13:Entry" name="currentEntry"/>
</bpel:variables>
<bpel:sequence>
<bpel:assign name="initCurrentEntry">
<bpel:copy>
<bpel:from>$QueryRPResponse.QueryResourcePropertiesResponse/*/ns13:Entry[position
    ()= $counter]</bpel:from>
<bpel:to>$currentEntry</bpel:to>
</bpel:copy>
</bpel:assign>
<bpel:if name="IF-createWSResourceVar">
<bpel:condition expressionLanguage="urn:oasis:names:tc:wsbpel:2.0:sublang:xpath1
    .0">boolean(count($currentEntry/ns13:MemberServiceEPR/wsa:ReferenceProperties
    /*) &gt; 0)</bpel:condition>
<bpel:scope>
<bpel:faultHandlers>......</bpel:faultHandlers>
<bpel:sequence>
<bpel:assign>
<bpel:copy>
<bpel:from>concat('GRAM-',$currentEntry/ns13:MemberServiceEPR/wsa:
    ReferenceProperties/here1:ResourceID)</bpel:from>
<bpel:to>$dbSetResourceRequest.ws-res/resource</bpel:to>
</bpel:copy>
<bpel:copy>
<bpel:from>$currentEntry/ns13:MemberServiceEPR/wsa:Address</bpel:from>
<bpel:to>$dbSetResourceRequest.ws-res/host</bpel:to>
</bpel:copy>
</bpel:assign>
<bpel:if>
<bpel:condition expressionLanguage="urn:oasis:names:tc:wsbpel:2.0:sublang:xpath1
    .0">contains( $currentEntry/ns13:Content/here:AggregatorConfig/here:
    GetResourcePropertyPollType/here:ResourcePropertyName ,'GLUECE' )</bpel:
    condition>
<bpel:sequence>
<bpel:assign>
<bpel:copy>
<bpel:from>
<bpel:literal>GLUECE</bpel:literal>
</bpel:from>
<bpel:to>$dbSetResourceRequest.ws-res/resourceData</bpel:to>
</bpel:copy>
</bpel:assign>
</bpel:sequence>
<bpel:else>
<bpel:sequence>
<bpel:throw faultName="ResourceDataTypeUnknownException"/>
</bpel:sequence>
```

```
</bpel:else>
</bpel:if>
</bpel:sequence>
</bpel:scope>
<bpel:else>
<bpel:sequence>
<bpel:if>
<bpel:condition expressionLanguage="urn:oasis:names:tc:wsbpel:2.0:sublang:xpath1
    .0">boolean( contains( $currentEntry/ns13:MemberServiceEPR/wsa:Address , '
    ReliableFileTransferFactoryService' ))</bpel:condition>
<bpel:sequence>
<bpel:assign>
<bpel:copy>
<bpel:from>
<bpel:literal>RFT</bpel:literal>
</bpel:from>
<bpel:to>$dbSetResourceRequest.ws-res/resource</bpel:to>
</bpel:copy>
<bpel:copy>
<bpel:from>$currentEntry/ns13:MemberServiceEPR/wsa:Address</bpel:from>
<bpel:to>$dbSetResourceRequest.ws-res/host</bpel:to>
</bpel:copy>
<bpel:copy>
<bpel:from>
<bpel:literal>RFT-DATA</bpel:literal>
</bpel:from>
<bpel:to>$dbSetResourceRequest.ws-res/resourceData</bpel:to>
</bpel:copy>
</bpel:assign>
</bpel:sequence>
<bpel:else>
<bpel:sequence>
<bpel:assign>
<bpel:copy>
<bpel:from>
<bpel:literal>Service-URI</bpel:literal>
</bpel:from>
<bpel:to>$dbSetResourceRequest.ws-res/resource</bpel:to>
</bpel:copy>
<bpel:copy>
<bpel:from>$currentEntry/ns13:MemberServiceEPR/wsa:Address</bpel:from>
<bpel:to>$dbSetResourceRequest.ws-res/host</bpel:to>
</bpel:copy>
<bpel:copy>
<bpel:from>
<bpel:literal>UNKNOWN</bpel:literal>
</bpel:from>
<bpel:to>$dbSetResourceRequest.ws-res/resourceData</bpel:to>
</bpel:copy>
</bpel:assign>
</bpel:sequence>
</bpel:else>
</bpel:if>
</bpel:sequence>
</bpel:else>
</bpel:if>
<bpel:assign name="initDbSetResourceRequest">
<bpel:copy>
<bpel:from>$currentEntry/ns13:Content/here:AggregatorData/.</bpel:from>
<bpel:to>$dbSetResourceRequest.ws-res/data</bpel:to>
</bpel:copy>
```

```
<bpel:copy>
<bpel:from variable="voID"/>
<bpel:to part="voId" variable="dbSetResourceRequest"/>
</bpel:copy>
</bpel:assign>
<bpel:invoke inputVariable="dbSetResourceRequest" name="dbSetResource" operation
    ="setResource" outputVariable="dbSetResouceResponse" partnerLink="
    DatabasePartnerLink" portType="db:DBService"/>
</bpel:sequence>
</bpel:scope>
</bpel:forEach>

<!------------------------------------------------------------>
<!-------------------- AKTIVITÄT 5 ---------------------------->
<!------------------------------------------------------------>

<bpel:if name="IF-VOResMappingSupported">
<bpel:condition expressionLanguage="urn:oasis:names:tc:wsbpel:2.0:sublang:xpath1
    .0"> boolean(count($QueryRPResponse.QueryResourcePropertiesResponse/*//
    voresource/.)&gt;0)
</bpel:condition>
<bpel:scope>
<bpel:variables>
<bpel:variable messageType="db:setVOResourceMappingSupportRequest" name="
    dbSetVOResMappingSupportRequest" />
<bpel:variable element="vo-res:voresources" name="VOResMapping" />
</bpel:variables>
<bpel:sequence>
<bpel:assign name="initSetVoResMappingSupportVar">
<bpel:copy>
<bpel:from>$IndexServiceEPR/wsa:Address</bpel:from>
<bpel:to>$dbSetVOResMappingSupportRequest.uri</bpel:to>
</bpel:copy>
<bpel:copy>
<bpel:from>true()</bpel:from>
<bpel:to>$dbSetVOResMappingSupportRequest.bool </bpel:to>
</bpel:copy>
</bpel:assign>
<bpel:invoke inputVariable="dbSetVOResMappingSupportRequest" operation="
    setVOResourceMappingSupport" partnerLink="DatabasePartnerLink" portType="db:
    DBService" />
<bpel:assign>
<bpel:copy>
<bpel:from>$QueryRPResponse.QueryResourcePropertiesResponse/*//voresources/.</
    bpel:from>
<bpel:to variable="VOResMapping" />
</bpel:copy>
</bpel:assign>
<bpel:forEach counterName="counter" parallel="no">
<bpel:startCounterValue>1</bpel:startCounterValue>
<bpel:finalCounterValue expressionLanguage="urn:oasis:names:tc:wsbpel:2.0:sublang
    :xpath1.0">
count($QueryRPResponse.QueryResourcePropertiesResponse/*//voresource/.)
</bpel:finalCounterValue>
<bpel:scope>
<bpel:variables>
<bpel:variable element="vo-res:voresource" name="currentResourceMapping" />
<bpel:variable messageType="db:getResourceIdRequest" name="dbGetResourceIdRequest
    " />
<bpel:variable messageType="db:getResourceIdResponse" name="
    dbGetResourceIdResponse" />
```

```
</bpel:variables>
<bpel:faultHandlers>
<bpel:catchAll>
<bpel:empty name="Do-Nothing" />
</bpel:catchAll>
</bpel:faultHandlers>
<bpel:sequence>
<bpel:assign name="setCurrentVoResourceMappiing">
<bpel:copy>
<bpel:from>$QueryRPResponse.QueryResourcePropertiesResponse/*//voresource[
    position()= $counter ]</bpel:from>
<bpel:to>$currentResourceMapping</bpel:to>
</bpel:copy>
</bpel:assign>
<bpel:assign name="initDbGetResourceIdRequest">
<bpel:copy>
<bpel:from>$currentResourceMapping/resource/@host</bpel:from>
<bpel:to>$dbGetResourceIdRequest.host</bpel:to>
</bpel:copy>
<bpel:copy>
<bpel:from>$currentResourceMapping/resource/@serviceType</bpel:from>
<bpel:to>$dbGetResourceIdRequest.resType</bpel:to>
</bpel:copy>
<bpel:copy>
<bpel:from variable="voID" />
<bpel:to part="voId" variable="dbGetResourceIdRequest" />
</bpel:copy>
</bpel:assign>
<bpel:invoke inputVariable="dbGetResourceIdRequest" name="dbGetResourceId"
    operation="getResourceId" outputVariable="dbGetResourceIdResponse"
    partnerLink="DatabasePartnerLink" portType="db:DBService" />
<bpel:forEach counterName="counter" name="ForEach-GridmapUser" parallel="no">
<bpel:startCounterValue>1</bpel:startCounterValue>
<bpel:finalCounterValue
expressionLanguage="urn:oasis:names:tc:wsbpel:2.0:sublang:xpath1.0">count(
    $currentResourceMapping/gridmap-users/gridmap-user/.)
</bpel:finalCounterValue>
<bpel:scope>
<bpel:variables>
<bpel:variable name="currentGridmapUser" type="xsd:string" />
<bpel:variable messageType="db:getPath4UserRequest" name="dbGetPath4UserRequest
    " />
<bpel:variable messageType="db:getPath4UserResponse" name="dbGetPath4UserResponse
    " />
<bpel:variable messageType="db:setVOResourceRequest" name="dbSetVOResourceRequest
    " />
<bpel:variable messageType="db:setVOResourceResponse" name="
    dbSetVOResourceResponse" />
</bpel:variables>
<bpel:sequence>
<bpel:assign name="getCurrentGripmapUser">
<bpel:copy>
<bpel:from>$currentResourceMapping/gridmap-users/gridmap-user[position() =
    $counter ]</bpel:from>
<bpel:to>$currentGridmapUser</bpel:to>
</bpel:copy>
</bpel:assign>
<bpel:assign name="initDbGetPath4UserRequest">
<bpel:copy>
<bpel:from variable="voID" />
<bpel:to>$dbGetPath4UserRequest.voId</bpel:to>
```

```
</bpel:copy>
<bpel:copy>
<bpel:from variable="currentGridmapUser" />
<bpel:to>$dbGetPath4UserRequest.userDN</bpel:to>
</bpel:copy>
</bpel:assign>
<bpel:invoke inputVariable="dbGetPath4UserRequest" name="dbGetPath4User"
    operation="getPath4User" outputVariable="dbGetPath4UserResponse" partnerLink
    ="DatabasePartnerLink" portType="db:DBService" />
<bpel:forEach counterName="counter" name="ForEach-voms-path"parallel="no">
<bpel:startCounterValue>1</bpel:startCounterValue>
<bpel:finalCounterValue expressionLanguage="urn:oasis:names:tc:wsbpel:2.0:sublang
    :xpath1.0"> count( $dbGetPath4UserResponse.voms-paths/item )
</bpel:finalCounterValue>
<bpel:scope>
<bpel:variables>
<bpel:variable name="currentVoms-path" type="xsd:string" />
</bpel:variables>
<bpel:sequence>
<bpel:assign name="getCurrentVoms-apth">
<bpel:copy>
<bpel:from>$dbGetPath4UserResponse.voms-paths/item[position()= $counter ]</bpel:
    from>
<bpel:to variable="currentVoms-path" />
</bpel:copy>
</bpel:assign>
<bpel:if>
<bpel:condition expressionLanguage="urn:oasis:names:tc:wsbpel:2.0:sublang:xpath1
    .0">boolean( $currentVoms-path !='' )
</bpel:condition>
<bpel:sequence>
<bpel:assign name="initDbSetVOResourceRequest">
<bpel:copy>
<bpel:from variable="voID" />
<bpel:to>$dbSetVOResourceRequest.voId</bpel:to>
</bpel:copy>
<bpel:copy>
<bpel:from>$dbGetResourceIdResponse.resid</bpel:from>
<bpel:to>$dbSetVOResourceRequest.resid</bpel:to>
</bpel:copy>
<bpel:copy>
<bpel:from>$currentVoms-path</bpel:from>
<bpel:to>$dbSetVOResourceRequest.voms-path</bpel:to>
</bpel:copy>
</bpel:assign>
<bpel:invoke inputVariable="dbSetVOResourceRequest"name="dbSetVOResource"
    operation="setVOResource"outputVariable="dbSetVOResourceResponse" partnerLink
    ="DatabasePartnerLink" portType="db:DBService" />
</bpel:sequence>
</bpel:if>
</bpel:sequence>
</bpel:scope>
</bpel:forEach>
</bpel:sequence>
</bpel:scope>
</bpel:forEach>
<bpel:forEach counterName="counter" name="ForEach-VomsAttr"parallel="no">
<bpel:startCounterValue>1</bpel:startCounterValue>
<bpel:finalCounterValue expressionLanguage="urn:oasis:names:tc:wsbpel:2.0:sublang
    :xpath1.0">
count( $currentResourceMapping/voms-attributes/voms-attribute/.)
```

```
</bpel:finalCounterValue>
<bpel:scope>
<bpel:variables>
<bpel:variable name="currentVoms-path" type="xsd:string" />
<bpel:variable messageType="db:isPathExistsRequest" name="dbIsPathExistsRequest
    " />
<bpel:variable messageType="db:isPathExistsResponse" name="dbIsPathExistsResponse
    " />
<bpel:variable messageType="db:setVOResourceRequest" name="dbSetVOResourceRequest
    " />
<bpel:variable messageType="db:setVOResourceResponse" name="
    dbSetVOResourceResponse" />
</bpel:variables>
<bpel:sequence>
<bpel:assign name="initCurrentVomsPath">
<bpel:copy>
<bpel:from>$currentResourceMapping/voms-attributes/voms-attribute[position() =
    $counter ]
</bpel:from>
<bpel:to>$currentVoms-path</bpel:to></bpel:copy>
</bpel:assign>
<bpel:assign name="initIsPathExistsRequest">
<bpel:copy>
<bpel:from variable="voID" />
<bpel:to>$dbIsPathExistsRequest.voId</bpel:to>
</bpel:copy>
<bpel:copy>
<bpel:from variable="currentVoms-path" />
<bpel:to>$dbIsPathExistsRequest.voms-path</bpel:to>
</bpel:copy>
</bpel:assign>
<bpel:invoke inputVariable="dbIsPathExistsRequest"name="dbIspathExists" operation
    ="isPathExists" outputVariable="dbIsPathExistsResponse" partnerLink="
    DatabasePartnerLink" portType="db:DBService" />
<bpel:if>
<bpel:condition expressionLanguage="urn:oasis:names:tc:wsbpel:2.0:sublang:xpath1
    .0"> boolean( $dbIsPathExistsResponse.resp )
</bpel:condition>
<bpel:sequence>
<bpel:assign name="initDbSetVOResource">
<bpel:copy>
<bpel:from variable="voID" />
<bpel:to>$dbSetVOResourceRequest.voId</bpel:to>
</bpel:copy>
<bpel:copy>
<bpel:from>$dbGetResourceIdResponse.resid</bpel:from>
<bpel:to>$dbSetVOResourceRequest.resid
</bpel:to>
</bpel:copy>
<bpel:copy>
<bpel:from variable="currentVoms-path" />
<bpel:to>$dbSetVOResourceRequest.voms-path</bpel:to>
</bpel:copy>
</bpel:assign>
<bpel:invoke inputVariable="dbSetVOResourceRequest" name="dbSetVOResource"
    operation="setVOResource"
      outputVariable="dbSetVOResourceResponse" partnerLink="DatabasePartnerLink"
          portType="db:DBService" />
</bpel:sequence>
</bpel:if>
</bpel:sequence>
```

```
</bpel:scope>
</bpel:forEach>
</bpel:sequence>
</bpel:scope>
</bpel:forEach>
</bpel:sequence>
</bpel:scope>
</bpel:if>
</bpel:sequence>
</bpel:scope>
```

4

D GLUE Schema

Informationssysteme (Information Services - siehe Kapitel 2.4) stellen eine wichtige Komponente von Grid-Systemen dar. Sie bieten Mechanismen und Dienste für das Entdecken und das Monitoring von Grid-Ressourcen. Innerhalb einer VO kann beispielsweise ein Mitglied durch diese Dienste nach Ressourcen mit bestimmten Eigenschaften suchen, und sie anschließend verwenden. Damit das VO-Mitglied die Ressourceneigenschaften richtig interpretieren kann, muss ein Informationsmodel definiert werden, dass die sie auf eine standardisierte Weise beschreibt. Das GLUE Schema [GLUE] ist ein Informationsmodel, das diese Ziele verfolgt. Es ermöglicht den Austausch von Informationen zwischen Systemen, mit verschiedenen Architekturen und soll alle möglichen Eigenschaften einer beliebigen Grid-Ressource darstellen können.
Der GLUE Schema Projekt wurde im April 2002 im Rahmen der EU-DataTAG-[1] und US-iVDGL-Projekte[2] gestartet, mit dem Ziel, Informationsmodelle und ihre Abbildung zu konkreten Schemas zur Darstellung von Grid-Ressourcen zu definieren. Die aktuelle Version ist GLUE Version 2.0, wir werden aber hier die Version 1.1 beschreiben. Dies ist darauf zurückzuführen, dass MDS (Globus Toolkit Informationssystem) die Version 1.1 verwenden. Im Folgenden wird das GLUE Schema kurz vorgestellt. Hierbei wird der Fokus auf das *Computing Element (CE)* des GLUE Schema gelegt. Dieses *CE* wird vom MDS verwendet, um die Eigenschaften von Grid-Ressourcen zu publizieren. Die komplette Beschreibung anderer Elemente wie das *Storage Element (SE)* und *Network Element (NE)* kann auf die offizielle Webseite (`http://glueschema.forge.cnaf.infn.it/`) gefunden werden.

D.1 Das Computing Element (CE) Schema

Das Computing Element Schema ermöglicht die Modellierung von Ressourceneigenschaften. Dabei können statische (z.B. Betriebsystemversion) sowie dynamische (z.B. Anzahl laufender Jobs) Eigenschaften modelliert werden. Gewöhnlich werden physikalischen Grid-Ressourcen durch Grid Services (WS-Ressourcen) repräsentiert. Diese Grid-Services bieten Standard-Schnittstellen für den Zugriff auf die Ressourcen an. Für den Zugriff und das Management von physikalischen Ressourcen werden Job Scheduler wie Condor[3], Portable Batch System (PBS)[4] eingesetzt. Diese Job Scheduler ermöglichen die direkte Ausführung von Jobs auf die physikalischen Ressourcen und liefern ebenso Informationen über die Ressourcen und die darauf laufenden Jobs zurück. Sie verwenden daher das *GLUE CE*, um alle diese Informationen darzustellen. Die so dargestellten Daten werden zum Informationssystem gesendet und dann da als Ressourceneigenschaften zur Verfügung stehen.

Die Abbildung D.1 stellt die Struktur des Computing Element Schemas dar. Das Schema besteht aus den folgenden Elementen: *Computing Element, Cluster, Sub-Cluster* und *Host*.

- **Computing Element**
  Ein `Computig Element` repräsentiert einen Service oder ein System (z.B. GRAM), das das Management und die Ausführung von Jobs auf Grid- Ressourcen ermöglicht. Es besteht aus folgenden Sub-Elementen: `Info, State, Policy, Job, AccessControlBase`. Das `Info`-Element liefert die statischen Eigenschaften einer Ressource. Das `State`-Element liefert den Status (dynamische Eigenschaften) einer Ressource. Das `Policy`-Element liefert die Anwendungsrichtlinien auf einer Ressource. Das `Job`-Element liefert Informationen über die auf die Ressource laufenden Jobs. Das `AccessControlBase`-Element liefert eine Liste von autorisierten Benutzern oder Gruppen von Be-

[1] `http://www.datatag.org/`
[2] `http://www.ivdgl.org/`
[3] `http://www.cs.wisc.edu/condor/`
[4] `http://www.openpbs.org`

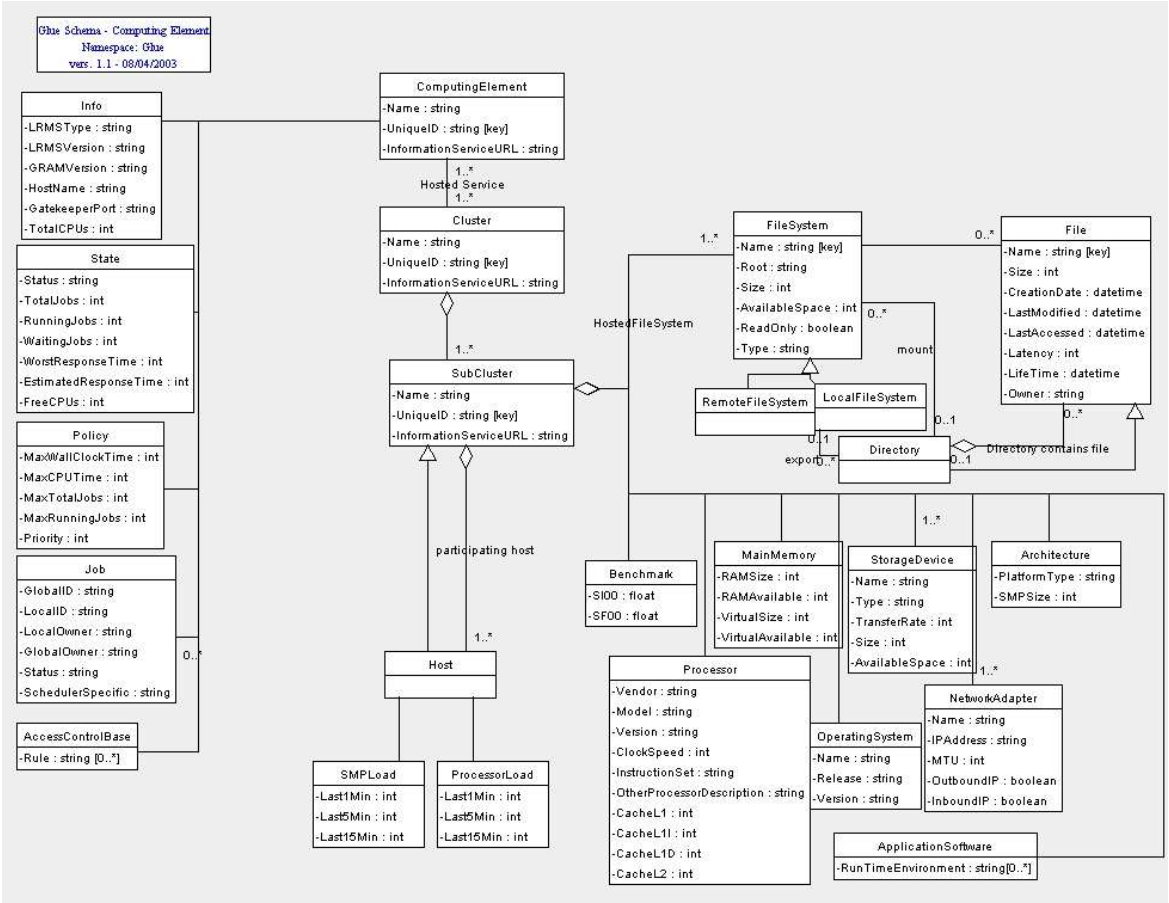


Abbildung D.1: Computing Element UML-Klassendiagramm [GLUE]

nutzern, die auf die Ressource zugreifen dürfen. Die vollständige Beschreibung dieser Elemente ist in der Abbildung D.2 zu finden.

- **Cluster und SubCluster**
  Ein `Cluster` ist eine Menge von verschiedenartigen Ressourcen (z.B. Rechner, die zu einem `Cluster` gehören, können verschiedene RAM, CPU oder Betriebsysteme haben). `SubCluster` sind hingegen homogen. Ein `Cluster` kann aus einer Menge von `SubClusters` oder von `Hosts` bestehen. Die Beschreibung der `Cluster`- und `SubCluster`-Elemente ist in der Abbildung D.3 zu finden.

- **Host**
  Ein `Host` stellt eine physikalische Ressource dar. Es kann z.B. ein Rechner, eine Datenbank oder eine Datei sein. Das UML-Diagramm in Abbildung D.1 stellt einen Überblick über die verschiedenen `Host`-Arten dar. In Abbildung D.4 ist die Beschreibung einiger `Host`-Elemente zu finden.

Im Rahmen des Globus Toolkit Projekts wurde das Computing Element Schema in XSD (XML Schema Definition) umgewandelt. Die Auflistung D.1 zeigt das CLUE CE Schema in XML Schema Definition (XSD) [XSD]. Dieses XSD wird in dieser Arbeit verwendet (Siehe Listing 5.5 im Kapitel 5.1.2)

Listing D.1: CLUE CE in XML Schema Definition

```
<?xml version="1.0" encoding="UTF-8"?>
<schema targetNamespace="http://mds.globus.org/glue/ce/1.1"
          xmlns:ce="http://mds.globus.org/glue/ce/1.1"
          attributeFormDefault="qualified"
          elementFormDefault="qualified"
          xmlns="http://www.w3.org/2001/XMLSchema"
          xml:lang="en" >
```

Entry	Description
Computing Element	Service that manages jobs and offers them execution environments provided by computing resources. The considered computing resources are those assigned to a single batch queue.
Property	**Description**
UniqueID	A unique identifier for the computing element. For example, EDG uses CE-hn:CE-port/jobmanager-CE-lrms-CE-queue
InformationServiceURL	URL of the local information service providing for info about this entity
Name	A name for this service
Info.LRMSType	Name of local resource management system
Info.LRMSVersion	Version of local resource manager
Info.GRAMVersion	The GRAM version
Info.HostName	Fully qualified host name for host on which the gatekeeper that corresponds to the computing element runs.
Info.GatekeeperPort	Port number for the gatekeeper
Info.TotalCPUs	Number of CPUs available to the queue. NB: this number should not be used to total available resources as more then one queue may be pointed to the same physical resources
Policy.MaxWallClockTime	The maximum wall clock time allowed for jobs submitted to the CE in mins (0=not specified)
Policy.MaxCPUTime	The maximum CPU time allowed for jobs submitted to the CE in mins (0=not specified)
Policy.MaxTotalJobs	The maximum allowed number of jobs in the CE (0=not specified)
Policy.MaxRunningJobs	The maximum number of jobs allowed to be running (0=not specified)
Policy.Priority	Info about the Queue Priority
State.RunningJobs	Number of currently running jobs
State.TotalJobs	Number of jobs in the CE (=RunningJobs+WaitingJobs)
State.Status	1. Queueing: the queue can accept job submission, but can't be served by the scheduler 2. Production: the queue can accept job submissions and is served by a scheduler 3. Closed: The queue can't accept job submission and can't be served by a scheduler 4. Draining: the queue can't accept job submission, but can be served by a scheduler
Status.WaitingJobs	Number of jobs that are in a state different than running
Status.WorstResponseTime	Worst time between job submission till when job starts its execution in sec
Status.EstimatedResponseTime	Estimated time between job submission till when job starts its execution in sec
Status.FreeCPUs	Number of free CPUs available to a scheduler (generally used with Condor)
AccessControlPolicyBase.Rule	A rule that grant/deny access to the Computing Element service, specific semantic needs to be defined (e.g. list of X509 user certificate subjects, VO names)
Job.LocalOwner	Owner local username
Job. GlobalOwner	Owner GSI subject name
Job.LocalID	Job local id
Job.GlobalID	Job global id
Job.Status	Job status {SUBMITTED, WAITING, READY, SCHEDULED, RUNNING, ABORTED, DONE, CLEARED, CHECKPOINTED}
Job.SchedulerSpecific	Scheduler specific info

Abbildung D.2: Beschreibung des Elements `Computing Element` [GLUE]

Entry	Description
Cluster	Set of machines providing computing power managed by a local management system
Property	**Description**
Name	Name of the cluster, taken from I.g.4, MDS-host-node-group-name
UniqueID	Unique ID for the cluster
InformationServiceURL	URL of the local information service providing for info about this entity

Entry	Description
SubCluster	Information about an homogeneous set of hosts as regards a selected number of host attributes
Property	**Description**
Name	Name of the Subcluster
UniqueID	Unique ID for the Subcluster
InformationServiceURL	URL of the local information service providing for info about this entity

Abbildung D.3: Beschreibung der Elemente `Cluster` und `SubCluster` [GLUE]

```
<element name="GLUECE" type="ce:GLUECERPType" />

<element name="GLUECESummary" type="ce:GLUECERPType" />

<element name="Cluster" type="ce:ClusterType" />
```

Entry	Description
Host	The Host element is used to represent details of a specific computing element. Many of the objects that may be contained at the host level can be included in the subcluster, these are marked.

Property	Description	Included in SubCluster?
Architecture.PlatformType	informally describes the platform type of the computing element	Yes
Architecture.SMPSize	number of CPUs in an SMP node	Yes
Operating System.Name	informally names the OS using a vendor-specific convention	Yes
Operating System.Release	informally names the OS release using a vendor-specific convention	Yes
Operating System.Version	informally names the OS or kernel version using a vendor-specific convention	Yes
Processor.Vendor	Informally names CPU vendor	Yes
Processor.Model	Informally names CPU model	Yes
Processor.Version	Informally names CPU version	Yes
Processor.ClockSpeed	The MHz associated with the CPUS in the subcluster	Yes
Processor.ComputerISA	informally names the Instruction Set Architecture (ISA) of the computing element	Yes
Processor.Features	informally names optional CPU features	Yes
Processor.CacheL1	first-level unified cache size (in kb) of a cpu	Yes
Processor.CacheL1I	first-level instruction cache size (in kb) of a cpu	Yes
Processor.CacheL1D	first-level data cache size (in kb) of a cpu	Yes
Processor.Cache2	second-level unified cache size (in kb) of a cpu	Yes
MainMemory.RAMSize	configured physical memory on any one CPU in the subcluster in MB	Yes
MainMemory.RAMAvailable	unallocated RAM size in MB	Yes
MainMemory.VirtualAvailable	available virtual memory	Yes
MainMemory.Viurtualsize	configured disk-based virtual memory (VM) in MB in a computing node	Yes
NetworkAdapter.Name	names a network interface	No
NetworkAdapter.IPAddress	ip address of a network interface	No
NetworkAdapter.OutboundIP	Defines if outbound connectivity is allowed from "worker nodes"- can a worked node initiate outbound connectivity	Yes
NetworkAdapter.InboundIP	Defines if inbound connectivity is allowed	Yes
NetworkAdapter.MTU	maximum transmission unit size (in bytes) for a network interface	no

Abbildung D.4: Beschreibung des Elements `Host` [GLUE]

```
<attributeGroup name="IdentifiableEntity">
  <annotation><documentation>
    Named identifiable entities should have this attribute group.
   </documentation></annotation>
  <attribute name="Name" type="string"/>
  <attribute name="UniqueID" type="string"/>
  <attribute name="InformationServiceURL" type="anyURI"/>
</attributeGroup>

<!-- ========== WS-ResourceProperties types -->

<complexType name="GLUECERPType">
  <annotation><documentation>
     A type suitable for use for a WS-Resource Property, containing
     some ComputingElements and some Clusters, bound to each other.
  </documentation></annotation>
  <sequence>

    <!-- <element name="Cluster" type="ce:ClusterType" minOccurs="0" maxOccurs="
        unbounded" /> -->
    <element ref="ce:Cluster" minOccurs="0" maxOccurs="unbounded" />
    <element name="ComputingElement" type="ce:ComputingElementType" minOccurs="0"
        maxOccurs="unbounded" />
    <any namespace="##other" minOccurs="0" maxOccurs="unbounded"/>
  </sequence>
</complexType>


<!-- ========== ComputingElement bits ========== -->
```

```
<complexType name="ComputingElementType">
  <annotation><documentation>
    Contains information about a ComputeElement.
  </documentation></annotation>
  <sequence>
    <element name="Info" type="ce:InfoType" minOccurs="0" maxOccurs="1" />
    <element name="State" type="ce:StateType" minOccurs="0" maxOccurs="1" />
    <element name="Policy" type="ce:PolicyType" minOccurs="0" maxOccurs="1" />
    <element name="Job" type="ce:JobType" minOccurs="0" maxOccurs="unbounded" />
    <element name="AccessControlBase" type="ce:AccessControlBaseType" minOccurs="
        0" maxOccurs="1" />
    <any namespace="##other" minOccurs="0" maxOccurs="unbounded"/>
  </sequence>

  <attributeGroup ref="ce:IdentifiableEntity" />
  <anyAttribute namespace="##other"/>
</complexType>


<complexType name="InfoType">
  <sequence>
    <any namespace="##other" minOccurs="0" maxOccurs="unbounded"/>
  </sequence>
  <attribute name="LRMSType" type="string"/>
  <attribute name="LRMSVersion" type="string"/>
  <attribute name="GRAMVersion" type="string"/>
  <attribute name="HostName" type="string"/>
  <attribute name="GatekeeperPort" type="string"/>
  <attribute name="TotalCPUs" type="string"/>
  <anyAttribute namespace="##other"/>
</complexType>


<complexType name="StateType">
  <sequence>
    <any namespace="##other" minOccurs="0" maxOccurs="unbounded"/>
  </sequence>

  <attribute name="Status" type="string">
  </attribute>
  <attribute name="TotalJobs" type="int"/>
  <attribute name="RunningJobs" type="int"/>
  <attribute name="WaitingJobs" type="int"/>
  <attribute name="WorstResponseTime" type="int">
    <annotation><documentation>in seconds</documentation></annotation>
  </attribute>
  <attribute name="EstimatedResponseTime" type="int">
    <annotation><documentation>in seconds</documentation></annotation>
  </attribute>
  <attribute name="FreeCPUs" type="int"/>
  <anyAttribute namespace="##other"/>
</complexType>


<complexType name="PolicyType">
  <sequence>
    <any namespace="##other" minOccurs="0" maxOccurs="unbounded"/>
  </sequence>
  <attribute name="MaxWallClockTime" type="int">
    <annotation><documentation>in minutes</documentation></annotation>
  </attribute>
```

```
  <attribute name="MaxCPUTime" type="int">
    <annotation><documentation>in minutes</documentation></annotation>
  </attribute>
  <attribute name="MaxTotalJobs" type="int"/>
  <attribute name="MaxRunningJobs" type="int"/>
  <attribute name="Priority" type="int"/>
  <anyAttribute namespace="##other"/>
</complexType>


<complexType name="JobType">
  <sequence>
    <any namespace="##other" minOccurs="0" maxOccurs="unbounded"/>
  </sequence>
  <attribute name="GlobalID" type="string"/>
  <attribute name="LocalID" type="string"/>
  <attribute name="LocalOwner" type="string"/>
  <attribute name="GlobalOwner" type="string"/>
  <attribute name="Status" type="string">
    <annotation><documentation>
      This can be one of SUBMITTED, WAITING, READY, SCHEDULED, RUNNING,
      ABORTED, DONE, CLEARED, CHECKPOINTED
    </documentation></annotation>
  </attribute>
  <attribute name="SchedulerSpecific" type="string"/>
  <anyAttribute namespace="##other"/>
</complexType>


<complexType name="AccessControlBaseType" >
  <sequence>
    <element name="Rule" type="string" minOccurs="0" maxOccurs="unbounded" />
    <any namespace="##other" minOccurs="0" maxOccurs="unbounded"/>
  </sequence>
  <anyAttribute namespace="##other"/>
</complexType>


<!-- ========== Cluster definitions ========== -->

<complexType name="ClusterType">
  <annotation><documentation>
    Contains information about a Cluster. This consists of a name,
    unique ID, and one or more subclusters.
  </documentation></annotation>
  <sequence>
    <element name="SubCluster" type="ce:SubClusterType" minOccurs="0" maxOccurs="
        unbounded"/>
    <any namespace="##other" minOccurs="0" maxOccurs="unbounded"/>
  </sequence>
  <attributeGroup ref="ce:IdentifiableEntity" />
  <anyAttribute namespace="##other"/>
</complexType>


<complexType name="SubClusterOrHostType">
  <annotation><documentation>
    A subcluster contains at least one host, as well as having properties
    associated with it that represent the state of every host within the
    subcluster.
  </documentation></annotation>
```

```
<sequence>
   <element name="Benchmark" type="ce:BenchmarkType" minOccurs="0" />
   <element name="Processor" type="ce:ProcessorType" minOccurs="0" />
   <element name="MainMemory" type="ce:MainMemoryType" minOccurs="0" />
   <element name="OperatingSystem" type="ce:OperatingSystemType" minOccurs="0" />
   <element name="StorageDevice" type="ce:StorageDeviceType" maxOccurs="unbounded
       " minOccurs="0" />
   <element name="Architecture" type="ce:ArchitectureType" minOccurs="0" />
   <element name="ApplicationSoftware" type="ce:ApplicationSoftwareType"
       minOccurs="0" />
   <element name="FileSystem" type="ce:FileSystemType" maxOccurs="unbounded"
       minOccurs="0"/>
   <element name="NetworkAdapter" type="ce:NetworkAdapterType" maxOccurs="
       unbounded" minOccurs="0"/>
   <any namespace="##other" minOccurs="0" maxOccurs="unbounded"/>
</sequence>
  <attributeGroup ref="ce:IdentifiableEntity" />
  <anyAttribute namespace="##other"/>
</complexType>

<complexType name="SubClusterType">
  <annotation><documentation>
  </documentation></annotation>
  <complexContent>
    <extension base="ce:SubClusterOrHostType">
     <sequence>
        <element name="Host" type="ce:HostType" minOccurs="0" maxOccurs="
            unbounded"/>
     </sequence>
    </extension>
  </complexContent>
</complexType>

<complexType name="HostType">
  <annotation><documentation>
    A host may have any of the properties of a subcluster.
  </documentation></annotation>
  <complexContent>
    <extension base="ce:SubClusterOrHostType">
     <sequence>
       <element name="ProcessorLoad" type="ce:LoadType" minOccurs="0" />
       <element name="SMPLoad" type="ce:LoadType" minOccurs="0" />
     </sequence>
    </extension>
  </complexContent>
</complexType>


<complexType name="LoadType">
  <annotation><documentation>
    Represents the unix-style CPU load-average multiplied by 100.
  </documentation></annotation>
    <sequence>
      <any namespace="##other" minOccurs="0" maxOccurs="unbounded"/>
    </sequence>
    <attribute name="Last1Min" type="int"/>
    <attribute name="Last5Min" type="int"/>
    <attribute name="Last15Min" type="int"/>
  <anyAttribute namespace="##other"/>
</complexType>
```

```
<complexType name="BenchmarkType">
   <sequence>
     <any namespace="##other" minOccurs="0" maxOccurs="unbounded"/>
   </sequence>
   <attribute name="SI00" type="float"/>
   <attribute name="SF00" type="float"/>
  <anyAttribute namespace="##other"/>
</complexType>


<complexType name="ProcessorType">
   <sequence>
     <any namespace="##other" minOccurs="0" maxOccurs="unbounded"/>
   </sequence>
   <attribute name="Vendor" type="string"/>
   <attribute name="Model" type="string"/>
   <attribute name="Version" type="string"/>
   <attribute name="ClockSpeed" type="int"/>
   <attribute name="InstructionSet" type="string"/>
   <attribute name="OtherProcessorDescription" type="string"/>
   <attribute name="CacheL1" type="int">
     <annotation><documentation>in kb</documentation></annotation>
   </attribute>
   <attribute name="CacheL1I" type="int">
     <annotation><documentation>in kb</documentation></annotation>
   </attribute>
   <attribute name="CacheL1D" type="int">
     <annotation><documentation>in kb</documentation></annotation>
   </attribute>
   <attribute name="CacheL2" type="int">
     <annotation><documentation>in kb</documentation></annotation>
   </attribute>
   <anyAttribute namespace="##other"/>
</complexType>


<complexType name="MainMemoryType">
   <sequence>
     <any namespace="##other" minOccurs="0" maxOccurs="unbounded"/>
   </sequence>
   <attribute name="RAMSize" type="long"> <!-- is long big enough here? -->
     <annotation><documentation>in MB</documentation></annotation>
   </attribute>
   <attribute name="RAMAvailable" type="long">
     <annotation><documentation>in MB</documentation></annotation>
   </attribute>
   <attribute name="VirtualSize" type="long">
     <annotation><documentation>in MB</documentation></annotation>
   </attribute>
   <attribute name="VirtualAvailable" type="long">
     <annotation><documentation>in MB</documentation></annotation>
   </attribute>

   <anyAttribute namespace="##other"/>
</complexType>


<complexType name="OperatingSystemType">
   <sequence>
     <any namespace="##other" minOccurs="0" maxOccurs="unbounded"/>
```

```
   </sequence>
   <attribute name="Name" type="string"/>
   <attribute name="Release" type="string"/>
   <attribute name="Version" type="string"/>
   <anyAttribute namespace="##other"/>
</complexType>


<complexType name="StorageDeviceType">
   <sequence>
     <any namespace="##other" minOccurs="0" maxOccurs="unbounded"/>
   </sequence>
   <attribute name="Name" type="string"/>
   <attribute name="Type" type="string"/>
   <attribute name="TransferRate" type="int"/>
   <attribute name="Size" type="int"/>
   <attribute name="AvailableSpace" type="int"/>
  <anyAttribute namespace="##other"/>
<!-- are these ints big enough? -->
</complexType>


<complexType name="NetworkAdapterType">
   <sequence>
     <any namespace="##other" minOccurs="0" maxOccurs="unbounded"/>
   </sequence>
   <attribute name="Name" type="string"/>
   <attribute name="IPAddress" type="string"/> <!-- restrict syntax here? can it
       be IPv6? -->
   <attribute name="MTU" type="int">
     <annotation><documentation>in bytes</documentation></annotation>
   </attribute>
   <attribute name="OutboundIP" type="boolean"/>
   <attribute name="InboundIP" type="boolean"/>
  <anyAttribute namespace="##other"/>
</complexType>


<complexType name="ArchitectureType">
   <sequence>
     <any namespace="##other" minOccurs="0" maxOccurs="unbounded"/>
   </sequence>
   <attribute name="PlatformType" type="string"/>
   <attribute name="SMPSize" type="int"/>
  <anyAttribute namespace="##other"/>
</complexType>


<!-- this deviates a little from the usual method of rendering the UML
diagram into XSD. Each RuntimeEnvironment is made an element to provide
somewhere to put any elements.
-->
<complexType name="ApplicationSoftwareType">
<sequence>
   <element name="RunTimeEnvironment" type="ce:RunTimeEnvironmentType" minOccurs=
       "0" maxOccurs="unbounded"/>
   <any namespace="##other" minOccurs="0" maxOccurs="unbounded"/>
</sequence>
  <anyAttribute namespace="##other"/>
</complexType>
```

```
<complexType name="RunTimeEnvironmentType">
  <sequence>
     <any namespace="##other" minOccurs="0" maxOccurs="unbounded"/>
  </sequence>
  <attribute name="Name" type="string" />
</complexType>

<complexType name="FileSystemType">
   <sequence>
     <any namespace="##other" minOccurs="0" maxOccurs="unbounded"/>
   </sequence>
 <attribute name="Name" type="string" />
 <attribute name="Root" type="string" />
 <attribute name="Size" type="long">
   <annotation><documentation>in MB</documentation></annotation>
 </attribute>
 <attribute name="AvailableSpace" type="long">
   <annotation><documentation>in MB</documentation></annotation>
 </attribute>
 <attribute name="ReadOnly" type="boolean" />
 <attribute name="Type" type="string" />
 <anyAttribute namespace="##other"/>
</complexType>

<complexType name="RemoteFileSystemType">
 <complexContent>
    <extension base="ce:FileSystemType" />
 </complexContent>
</complexType>

<complexType name="LocalFileSystemType">
 <complexContent>
    <extension base="ce:FileSystemType">
       <sequence>
          <element name="export" type="ce:DirectoryType" minOccurs="0" maxOccurs=
              "unbounded"/>
       </sequence>
    </extension>
 </complexContent>
</complexType>


<complexType name="FileType">
   <sequence>
     <any namespace="##other" minOccurs="0" maxOccurs="unbounded"/>
   </sequence>
 <attribute name="Name" type="string" />
 <attribute name="Size" type="int" />
 <attribute name="CreationDate" type="dateTime" />
 <attribute name="LastModified" type="dateTime" />
 <attribute name="LastAccessed" type="dateTime" />
 <attribute name="Latency" type="duration">
   <annotation><documentation>
      In UML version, this is an int number of seconds
   </documentation></annotation>
 </attribute>
 <attribute name="LifeTime" type="dateTime" />
 <attribute name="Owner" type="string" />
 <anyAttribute namespace="##other"/>
</complexType>
```

```
<complexType name="DirectoryType">
 <complexContent>
    <extension base="ce:FileType">
       <sequence>
          <element name="File" type="ce:FileType" minOccurs="0" maxOccurs="
              unbounded">
             <annotation><documentation>
                From the UML diagram: Directory contains file
             </documentation></annotation>
          </element>
          <element name="mount" minOccurs="0" maxOccurs="unbounded">
             <annotation><documentation>
                The FileSystemName attribute should refer to the name
                of a FileSystem object. From the UML diagram: mount.
             </documentation></annotation>
             <complexType>
                <attribute name="FileSystemName" type="string" />
             </complexType>
          </element>
       </sequence>
    </extension>
 </complexContent>
</complexType>

</schema>
```

Literaturverzeichnis

[ANAT] IAN FOSTER, CARL KESSELMAN, STEVEN TUECKE: *The Anatomy of the Grid.Enabling Scalable Virtual Organisations*, http://www.globus.org/alliance/publications/papers/anatomy.pdf .

[BPELv2.0] OASIS: *Web Services Business Process Execution Language Version 2.0*, April 2007, http://docs.oasis-open.org/wsbpel/2.0/OS/wsbpel-v2.0-OS.pdf .

[CAS] *CAS*, http://www.globus.org/toolkit/docs/4.0/security/cas/ .

[Fran 03] FRANK, RAINER: *Architekturen mit Web Services*. JavaSpektrum, 2003.

[GARA] *A GUIDE TO GARA*, Mai 2000, http://www-fp.mcs.anl.gov/qos/papers/gara_guide.pdf .

[GLO05] ABT, TOBIAS: *Adaptive Manageability Services im Grid*, http://wwwmnmteam.informatik.uni-muenchen.de/pub/Fopras/abt06/PDF-Version/abt06.pdf .

[GLUE] *GLUE SCHEMA WEBSITE*, http://glueschema.forge.cnaf.infn.it/ .

[Grid 1] FOSTER, IAN und CARL KESSELMAN: *The Grid: Blueprint for a New Computing Infrastructure: Blueprint for the New Computing Infrastructure*. Morgan Kaufmann, ISBN 1-55860-475-8, First Edition Auflage, 1998.

[Grid 2] FOSTER, IAN und CARL KESSELMAN: *The GRID 2:Blueprint for a new Computing Infrastructure*. Morgan Kaufmann, ISBN 1-55860-933-4, Second Edition Auflage, 2004.

[GridFTP] W. ALLCOCK1, I. FOSTER, U.A.: *The Globus Striped GridFTP Framework and Server*, http://www.globus.org/alliance/publications/papers/gridftp_final.pdf .

[GridShib1] J. BASNEY, T. BARTON, U.A.: *Identity Federation and Attribute-based Authorization through the Globus Toolkit, Shibboleth, GridShib, and MyProxy*.

[GridShib2] WELCH, T. BARTON, U.A. W.: *Attributes, Anonymity, and Access: Shibboleth and Globus.Integration to Facilitate Grid Collaboration*.

[GT4] ALLIANCE, GLOBUS: *Globus Toolkit Official Documentation*, http://www.globus.org/toolkit/docs/ .

[GT4-Paper] FOSTER, IAN: *Globus Toolkit Version 4: Software for Service-Oriented Systems*, http://www.globus.org/alliance/events/sc05/GT4.pdf .

[HAN 99] HEGERING, H.-G., S. ABECK und B. NEUMAIR: *Integrated Management of Networked Systems – Concepts, Architectures and their Operational Application*. Morgan Kaufmann Publishers, ISBN 1-55860-571-1, 1999. 651 p.

[LIB] *Project Liberty Homepage*, http://www.projectliberty.org/ .

[Lipp 06] LIPP, MANFRED: *VPN - Virtuelle Private Netzwerke. Aufbau und Sicherheit*. Addison-Wesley, München, Januar 2006.

[OGSAv1.5] W3C: *Open Grid Services Architecture WG (OGSA-WG)*, http://forge.gridforum.org/sf/docman/do/listDocuments/projects.ogsa-wg/docman.root.published_documents.ogsa_1_5 .

[OGSI] I. FOSTER, C. KESSELMAN, U.A.: *Open Grid Services Infrastructure (OGSI) Version 1.0*, http://www.globus.org/toolkit/draft-ggf-ogsi-gridservice-33_2003-06-27.pdf .

[PBS] *OpenPBS WEBSEITE*, http://www.openpbs.org/ .

[PERMIS] *PERMIS WEBSEITE*, http://www.permis.org/ .

[pki 02] *Understanding PKI. Concepts, Standards, and Deployment Considerations.: Concepts, Standards, and Deployment Considerations*. Addison-Wesley Longman, Amsterdam, 2002.

[ProxyCert] IAN FOSTER, CARL KESSELMAN, STEVEN TUECKE U.A.: *X.509 Proxy Certificates for Dynamic Delegations*, http://www.globus.org/alliance/publications/papers/pki04-welch-proxycert-final.pdf .

[SAML] OASIS: *Technical Overview of the OASIS Security Assertion Markup Language (SAML) V1.1*, Mai 2004, http://www.oasis-open.org/committees/download.php/6837/sstc-saml-tech-overview-1.1-cd.pdf . Committee Draft.

[Schi 07] SCHIFFERS, MICHAEL: *Management dynamischer Virtueller Organisationen in Grids*. Dissertation, Ludwig–Maximilians–Universität München, 2007.

[SLA] JOHN J. LEE, RON BEN-NATAN: *Integrating Service Level Agreement: Optimizing Your OSS for SLA Delivery*. Wiley & Sons, ISBN 0-471-21012-9, 2002.

[SOA1] MUNINDAR P. SINGH, MICHAEL N. HUHNS: *Service-Oriented Computing*. WILEY, ISBN 0-470-09148-7, 2005.

[SOAPv1.2] W3C: *SOAP*, Mai 2000, http://www.w3.org/TR/2000/NOTE-SOAP-20000508/ .

[SSO] GARMAN, JASON: *Kerberos. The Definitive Guide. Cross-Platform Authentication & Single-Sign-On.: The Definitive Guide*. O'Reilly Media, ISBN 0-596-00403-6, 2003.

[tls 00] *SSl and TLS. Building and Designing Secure Systems.: Building and Designing Secure Systems*. Addison-Wesley Longman, ISBN 0201615983, 2000.

[TTL 02] THAIN, DOUGLAS, TODD TANNENBAUM und MIRON LIVNY: *Condor and the Grid*. John Wiley & Sons Inc., December 2002.

[TUT1] JIM HEUMANN, IBM: *Online Tutorial:Introduction to business modeling using the Unified Modeling Language (UML)*, 2003, http://www.ibm.com/developerworks/rational/library/360.html .

[UDDI] OASIS: *UDDI Version 3.0.2*, Oktober 2004, http://uddi.org/pubs/uddi_v3.htm .

[UMLv1.4.2] OMG: *UML 1.4.2 Specification*, 2001, http://www.omg.org/docs/formal/05-04-01.pdf .

[VOMS] *Virtual Organization Membership Service*, http://edg-wp2.web.cern.ch/edg-wp2/security/voms/voms.html .

[VOMS05] CZYZEWSKI, MARCIN: *Management von Mitgliedschaften zu Virtuellen Organisationen im Grid*, http://wwwmnmteam.informatik.uni-muenchen.de/pub/Fopras/czyz06/PDF-Version/czyz06.pdf .

[VOX] *VOMRS User Documentation*, http://wwwserver2.fnal.gov/www/docs/vox/voxconv/Output/voxTOC.html .

[WS-Addr] W3C: *Web Services Addressing (WS-Addressing)*, August 2004, http://www.w3.org/Submission/2004/SUBM-ws-addressing-20040810/ .

[WS-Bf] OASIS: *Web Services Base Faults 1.2 (WS-BaseFaults)*, Juni 2004, http://docs.oasis-open.org/wsrf/2004/06/wsrf-WS-BaseFaults-1.2-draft-02.pdf . Working Draft.

[WS-CDL] W3C: *Web Services Choreography Description Language Version 1.0*, Dezember 2004, http://www.w3.org/TR/2004/WD-ws-cdl-10-20041217/ .

[WS-Res] IAN FOSTER, JEFFREY FREY, STEVE GRAHAM U.A.: *Modeling Stateful Resources in Web Services.*, März 2004, http://www-106.ibm.com/developerworks/library/ws-resource/ws-modelingresources.pdf .

[WS-ResLt] IAN FOSTER, JEFFREY FREY, STEVE GRAHAM U.A.: *Web Services Resource LifetimeWS-ResourceLifetime)*, März 2004, http://www.ibm.com/developerworks/library/ws-resource/ws-resourcelifetime.pdf .

[WS-RP] IAN FOSTER, JEFFREY FREY, STEVE GRAHAM U.A.: *Web Services Resource Properties (WS-ResourceProperties)*, März 2003, http://www.ibm.com/developerworks/library/ws-resource/ws-resourceproperties.pdf .

[WS-S] OASIS: *Web Services Security: SOAP Message Security 1.1 (WS-Security 2004)*, Februar 2006, http://www.oasis-open.org/committees/download.php/16790/wss-v1.1-spec-os-SOAPMessageSecurity.pdf .

[WS-SC] *Web Services Secure Conversation Language (WS-SecureConversation)*, Februar 2005, http://specs.xmlsoap.org/ws/2005/02/sc/WS-SecureConversation.pdf .

[WS-SGr] OASIS: *Web Services Service Group 1.2*, Juni 2004, http://docs.oasis-open.org/wsrf/2004/06/wsrf-WS-ServiceGroup-1.2-draft-02.pdf .

[WSDLv1.1] W3C: *Web Services Description Language (WSDL) 1.1*, März 2001, http://www.w3.org/TR/2001/NOTE-wsdl-20010315 .

[WSN] OASIS: *Web Services Base Notification 1.3 (WS-BaseNotification)*, Oktober 2006, http://docs.oasis-open.org/wsn/wsn-ws_base_notification-1.3-spec-os.pdf .

[WSRF] IAN FOSTER, JEFFREY FREY, STEVE GRAHAM U.A.: *The WS-Resource Framework*, März 2004, http://www.globus.org/wsrf/specs/ws-wsrf.pdf .

[xml 02] *Web Services Essentials. Distributed Applications with XML-RPC, SOAP, UDDI & WSDL*. O'Reilly Media, 2002.

[XPathv1.0] W3C: *XML Path Language (XPATH) Version 1.0*, November 1999, http://www.w3.org/TR/1999/REC-xpath-19991116 .

[XPathv2.0] W3C: *XML Path Language (XPath) 2.0*, Januar 2007, http://www.w3.org/TR/xpath20/ .

[XSD] W3C: *XML Schema Definition*, März 2001, http://www.w3.org/TR/2004/REC-xmlschema-1-20041028/structures.html .