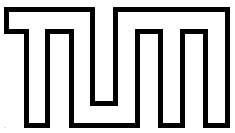


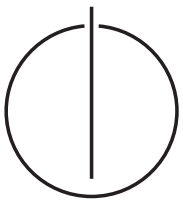
Technische Universität München

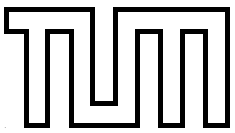
Fakultät für Informatik

**Diplomarbeit in Informatik**

# Entwicklung einer einheitlichen Autorisierungs- und Authentifizierungs-Schnittstelle für heterogene Grids am Beispiel D-Grid

Wolfgang Kirchler





Technische Universität München

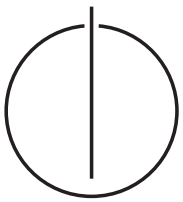
Fakultät für Informatik

**Diplomarbeit in Informatik**

# Entwicklung einer einheitlichen Autorisierungs- und Authentifizierungs-Schnittstelle für heterogene Grids am Beispiel D-Grid

# Development of a uniform authentication and authorization interface for heterogeneous Grids with special emphasis on the D-Grid

| | |
|---|---|
| Bearbeiter: | Wolfgang Kirchler |
| Aufgabensteller: | Prof. Dr. Heinz-Gerd Hegering |
| Betreuer: | Dr. Michael Schiffers |
| Abgabedatum: | 15. September 2008 |



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

München, den 15. September 2008

. . . . . .
*(Unterschrift des Kandidaten)*

**Abstract**

Neben dem heute etablierten Internet und dem damit entstandenen verteilten Rechnen, wurde in den vergangenen Jahren eine in Zukunft immer wichtiger werdende neue IT-Infrastruktur entwickelt - das so genannte Grid. Das Ziel eines Grids ist die gemeinsame, institutionsübergreifende Nutzung von verteilten Ressourcen, welche über standardisierte Protokolle und Schnittstellen erreicht werden. Ein wichtiger Aspekt in einer Grid-Umgebung ist die Authentifizierung und Autorisierung der Benutzer, damit diese auf die gewünschten Ressourcen zugreifen können. Leider implementieren die vorhandenen Grid Middleware-Technologien die Authentifizierung und Autorisierung auf sehr unterschiedliche Art und Weise. Damit die Verwendung eines Grids und dessen Ressourcen vereinfacht wird, müssen die Authentifizierungs- und Autorisierungsmechanismen der verschiednen Middleware-Technologien vereinheitlicht werden.

Im Rahmen dieser Diplomarbeit wird deshalb eine Abstraktionsschicht eingeführt, welche die Authentifizierung und Autorisierung von der Grid-Middleware entkoppelt und in eine darüber liegende Schicht, den so genannten VO-Layer, integriert. Diese Schicht hat ihren Namen vom Konzept der virtuellen Organisation (VO), welches im Grid-Umfeld sehr verbreitet ist und zur Autorisierung der Benutzer verwendet wird. Der VO-Layer hat die Aufgabe die Authentifizierungs- und Autorisierungsanfragen der Benutzer entgegenzunehmen, zu prüfen und anschließend an die entsprechende Middleware-Technologie weiterzuleiten. Der VO-Layer fungiert somit als Proxy zwischen dem Benutzer und der Grid-Middleware, welche die Ressource bereitstellt.

Damit die Autorisierung vereinheitlicht werden kann, wird in dieser Diplomarbeit eine generische VO-Struktur entwickelt, in der die verschiedenen VO-Konzepte der einzelnen D-Grid Communities abgebildet werden können. Dadurch kann jede Community ihr eigenes VO-Konzept behalten, die Autorisierung aber wird dadurch vereinheitlicht.

# Inhaltsverzeichnis

# 1 Einleitung

Grid-Computing hat sich in den vergangenen Jahren, vor allem im Bereich der Wissenschaft, rasant weiterentwickelt und wird in immer mehr wissenschaftlichen Projekten, aber auch in der Industrie eingesetzt. Das Ziel eines Grids ist die gemeinsame, koordinierte Nutzung von Ressourcen und die gemeinsame Lösung von Problemen innerhalb dynamischer, institutionsübergreifender, virtueller Organisationen. Das bedeutet, nach der Festlegung von Abrechnungsdaten und Rechten, soll ein direkter Zugang zu Rechnern, Daten oder anderen Instrumenten gemeinschaftlich ermöglicht werden. Eine virtuelle Organisation ist, - im Kontext des Grid-Computings - ein dynamischer Zusammenschluss von Individuen und/oder Institutionen, die gemeinsame Ziele bei der Nutzung des Grids verfolgen. Zwar liegt der Fokus vieler Arbeiten im Bereich des verteilten Rechnens, dennoch ist das oberste Ziel die Entwicklung eines einheitlichen, globalen Grids.

Damit das Grid und dessen Ressourcen nutzbar werden, wird eine Software benötigt, mittels welcher der Benutzer auf das Grid zugreifen kann. Diese Software wird Grid-Middleware genannt. Sie hat die Aufgabe, die Benutzer zu authentifizieren, die Ressourcen zu verwalten, den Zugriff auf die Ressourcen zu steuern, den Transport der Daten zu organisieren und die Jobs auf die Ressourcen zu verteilen. Es existieren mehrere Grid Middleware-Technologien, welche aus verschiedenen Bereichen der Wissenschaft stammen und zum Teil mit unterschiedlichen Anforderungen entwickelt wurden. Ein wichtiger Schritt auf dem Weg zum globalen Grid ist die Standardisierung der Grid-Architektur und der Grid-Protokolle. Denn nur wenn diese standardisiert sind, ist die Interoperabilität zwischen den verschiedenen Grid Middleware-Technologien gewährleistet.

## 1.1 Motivation

Eine wichtige Aufgabe der Grid-Middleware ist es, Benutzer zu authentifizieren und den Zugriff auf Ressourcen zu autorisieren. Jede Grid Middleware-Technologie verwendet dazu ihre eigenen Mechanismen. Die Authentifizierung erfolgt bei allen Middleware-Technologien mittels X.509 Zertifikaten, welche von einer vertrauenswürdigen Zertifizierungsstelle (engl. certificate authority, CA) ausgestellt und signiert werden. Die Autorisierung auf den Ressourcen erfolgt über verschiedene Zugriffskontrolllisten (engl. access control list, ACL), welche in den aktuellen Middleware-Technologien unterschiedlich implementiert sind. Dabei muss für jede Ressource genau festgelegt werden, welche Benutzer in welchem Umfang Zugriff auf diese Ressource haben.

Ein wichtiger Mechanismus zur Benutzer- und Zugriffsverwaltung im Grid ist die virtuelle Organisation (VO, siehe Kapitel 2.2). In einer VO werden Benutzer und Ressourcen aus rea-

len Organisationen zusammengefasst, um ein gemeinsames Ziel zu erreichen. Dabei erhalten Benutzer nur Zugriff auf jene Ressourcen, die sich in der selben VO wie sie selbst befinden. Innerhalb von VOs können auch noch verschiedene Gruppen und Rollen existieren, wodurch eine feingranulare Autorisierung möglich wird.

Die effiziente Verwaltung von VOs wird dadurch erschwert, dass es nicht nur eine einzige Grid Middleware-Technologie gibt, sondern dass mehrere verschiedene Lösungen existieren. Im konkreten Fall des D-Grids (siehe Kapitel 2.5) werden die drei Middleware-Technologien Globus Toolkit, UNICORE und gLite (siehe Kapitel 2.6, 2.7 und 2.8) verwendet. Aufgrund dieser Heterogenität im D-Grid, gibt es zur Zeit kein einheitliches VO-Management und auch keine einheitliche Authentifizierungs- und Autorisierungs-Infrastruktur. Des weiteren ist es nicht möglich, VOs unabhängig von der eingesetzten Grid Middleware zu erstellen.

Durch das Konzept der virtuellen Organisationen wird die Benutzer- und Zugriffsverwaltung wesentlich vereinfacht. Leider implementieren aktuelle Grid Middleware-Technologien das VO-Konzept nur unzureichend beziehungsweise gar nicht. So ist es etwa in der am weitesten verbreiteten Grid Middleware, dem Globus Toolkit, ohne Erweiterungen nicht möglich virtuelle Organisationen zu erstellen. Das Implementieren einer effizienten Authentifizierungs- und Autorisierungs-Infrastruktur (AAI) wird deshalb erschwert.

Aktuelle VO-Management Systeme wie VOMS/VOMRS (siehe Kapitel 4.4) unterstützen zwar VOs, allerdings können diese VOs nur aus Benutzern bestehen. Da es nicht möglich ist, Ressourcen in eine VO aufzunehmen, muss die Autorisierung auf den Ressourcen über einen anderen Mechanismus erfolgen (beispielsweise Zugriffskontrolllisten oder attributbasierte Autorisierung). Durch die Integration von Ressourcen in VOs könnte die Autorisierung wesentlich vereinfacht werden, da die Grid-Middleware nur überprüfen muss, ob sich der Benutzer und die angeforderte Ressource in der selben VO befinden.

## 1.2 Aufgabenstellung

Ziel dieser Arbeit ist die Entwicklung einer einheitlichen Authentifizierungs- und Autorisierungsschnittstelle für heterogene Grids. In diesen Grids können die Grid Middleware-Technologien Globus Toolkit, UNICORE und gLite zum Einsatz kommen. Dazu sind die folgenden Fragen zu klären:

1. Welches VO-Verständnis liegt den unterschiedlichen Middleware-Technologien zugrunde?
2. Wie können VOs mit bestehenden Techniken etabliert und administriert werden und wo liegen die Grenzen?
3. Wie kann eine Authentifizierungs- und Autorisierungs-Schnittstelle für heterogene Grids aussehen, welche die Benutzer authentifiziert und den Zugriff auf die Ressourcen steuert?
4. Wie kann eine VO-Struktur aussehen, welche nicht nur Benutzern, sondern auch Ressourcen die Mitgliedschaft in VOs erlaubt?

Um das Ziel einer einheitlichen Authentifizierungs- und Autorisierungs-Schnittstelle zu erreichen, wird - aufbauend auf den bestehenden Grid Middleware-Technologien - eine Abstraktionsschicht eingeführt, welche die Authentifizierung und Autorisierung übernimmt. Abbildung 1.1 zeigt die schematische Darstellung dieser neuen Schicht, welche im Folgenden *VO-Layer* genannt wird. Bei der Entwicklung des VO-Layers werden insbesondere folgende Komponenten in dieser Diplomarbeit behandelt, die in Abbildung 1.1 zusammen mit den darunter liegenden Grid Middleware-Technologien abgebildet sind:

- VO-Layer mit Authentifizierungs- und Autorisierungs-Funktion.
- Datenstruktur zum Speichern der VOs in der so genannten VO-Layer Datenbank (VOL-DB).
- Schnittstelle zu den einzelnen Grid Middleware-Technologien.
- API zum Entwickeln von Client-Software.
- Deployment des VO-Layers in einer Testumgebung.

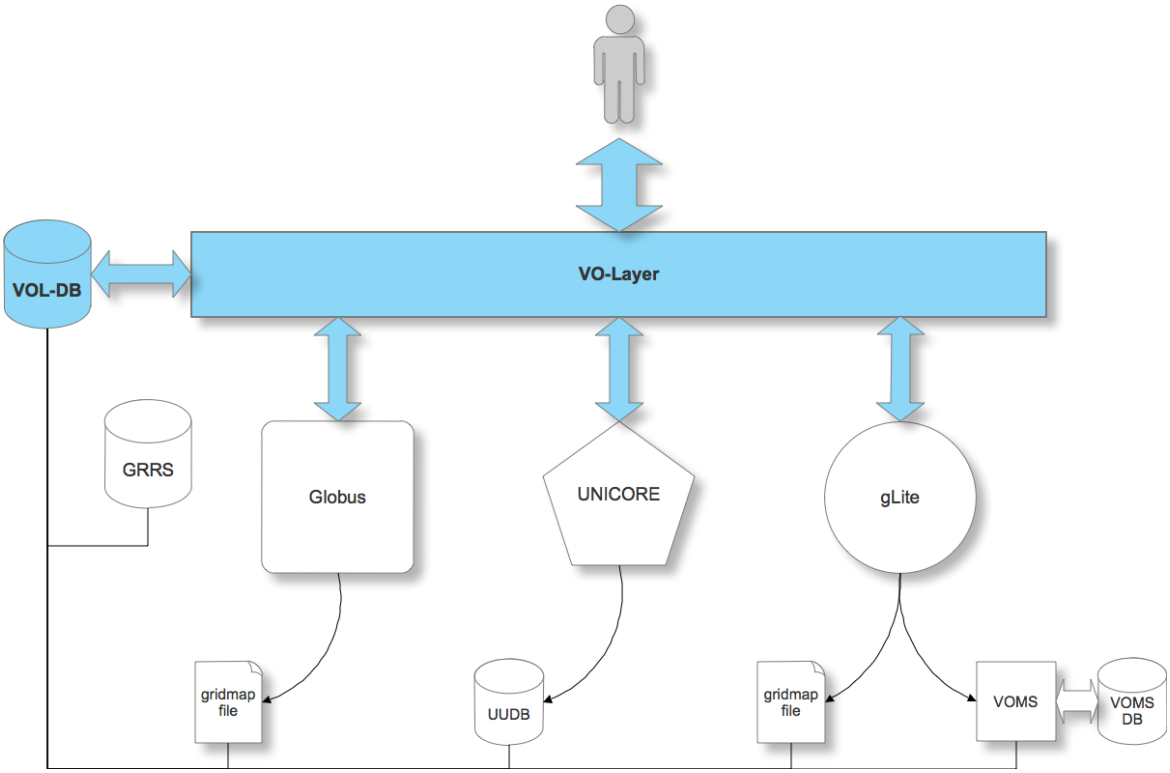



Abbildung 1.1: VO-Layer

Der neu eingeführte VO-Layer soll unabhängig von der verwendeten Grid-Middleware sein und nur auf bestehende Grid-Befehle zugreifen, die Grid-Middleware also nicht verändern. Die Anbindung an die Grid Middleware-Technologien erfolgt dabei ausschließlich über die von der Middleware bereitgestellten Kommandozeilenbefehle, eine Unterstützung für Grid-Portalsoftware, welche über einen Webbrowser oder über eine eigene Anwendung mit einer GUI gesteuert wird, ist in dieser Diplomarbeit nicht vorgesehen. Bei der Entwicklung der

Datenstruktur des VO-Layers, muss sichergestellt werden, dass diese VO-Struktur generisch ist, damit die bestehenden VO-Konzepte mit der neuen VO-Struktur abgebildet werden können. Des weiteren muss sichergestellt werden, dass die VO-Informationen in der Datenbank des VO-Layers mit denen in den Grid Middleware-Technologien synchronisiert werden. Dazu zählen insbesondere Informationen aus gridmap files, VOMS Datenbanken und den UNICORE User Database's (UUDB).

## 1.3 Vorgehensweise

Diese Diplomarbeit gliedert sich im Wesentlichen in drei Teile: der Anforderungsanalyse, dem Systementwurf und der prototypischen Implementierung des VO-Layers. In Abbildung 1.2 ist die methodische Vorgehensweise nochmals bildhaft dargestellt.

**Analyse** In der ersten Phase der Analyse werden zunächst die Grundlagen des Grid-Computings eingeführt. Dazu werden die wichtigsten Begriffe wie Grid-Computing, virtuelle Organisation, Authentifizierung und Autorisierung definiert und erklärt. Des weiteren wird ein genauerer Blick auf das D-Grid geworfen und die darin verwendeten Middleware-Technologien (Globus Toolkit, UNICORE und gLite) auf ihre Authentifizierungs- und Autorisierungs-Mechanismen hin untersucht.

Anschließend werden die Szenarien *authentifizieren* und *autorisieren* im D-Grid beschrieben. Mit Hilfe dieser Szenarien und der Aufgabenstellung, werden Use Cases erstellt, welche die genauen funktionalen Anforderungen an den VO-Layer definieren.

**Entwurf** Sobald die Anforderungsanalyse abgeschlossen ist, wird das eigentliche System entworfen. Das System - in diesem Fall der VO-Layer mit seinen Schnittstellen - wird als Objektmodell definiert und die Schnittstellen werden spezifiziert. Des weitern gehört zum Systementwurf die Modellierung einer Datenstruktur zur Speicherung der Informationen in der VO-Layer Datenbank.

**Realisierung** Für die Realisierung wird ein Testbett erstellt, welches alle Komponenten enthält, die für die Realisierung dieser Arbeit benötigt werden. In diesem Testbett wird nun das zuvor entwickelte Konzept prototypisch implementiert. Anschließend wird das System, anhand des im ersten Schritt erstellten Anforderungskatalogs, bewertet.

Die Diplomarbeit gliedert sich in acht Kapitel, wobei der Abbildung 1.2 entnommen werden kann, welches Kapitel in welchem Teil der Arbeit behandelt wird.

Das **zweite Kapitel** befasst sich mit den Grundlagen, die für das weitere Verständnis dieser Arbeit nötig sind. Dazu gehört die Erklärung der Begriffe Grid-Computing, virtuelle Organisationen, Authentifizierung und Autorisierung. Ein weiterer Abschnitt beschäftigt sich mit dem D-Grid, dessen Aufgaben und Ziele. Anschließend werden die drei im D-Grid verwendeten Grid Middleware-Technologien Globus Toolkit, UNICORE und gLite vorgestellt.

Das **dritte Kapitel** beschreibt das Ergebnis der Anforderungsanalyse. Zunächst werden die Szenarien beschrieben, die für den VO-Layer erstellt wurden. Anschließend werden die aus den Szenarien abgeleiteten Use Cases beschrieben.

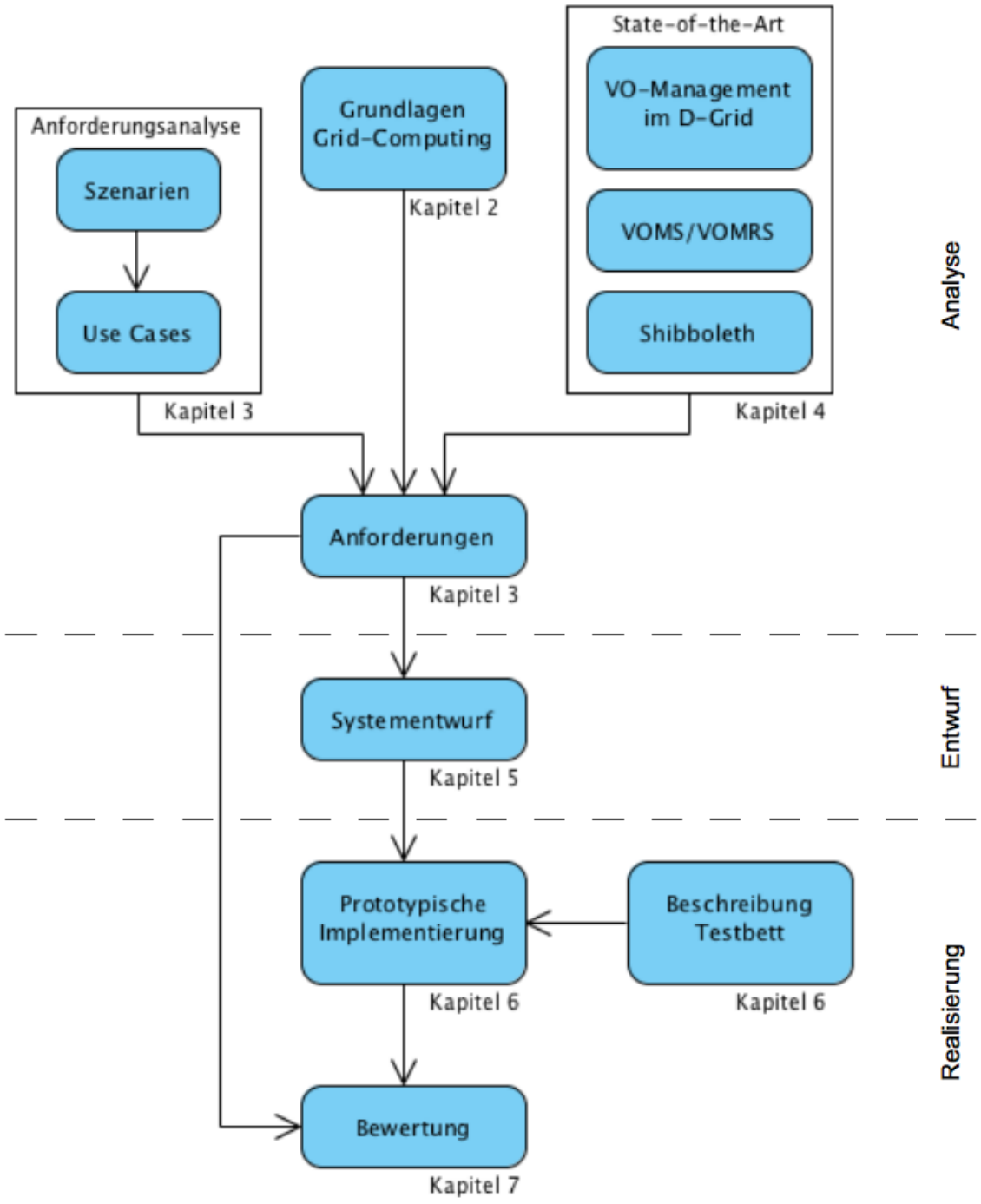



Abbildung 1.2: Vorgehensweise

Das **vierte Kapitel** stellt die aktuellen Authentifizierungs- und Autorisierungstechnologien vor und es wird analysiert, ob sich diese für die zuvor definierten Anforderungen eignen. Des weiteren wird untersucht, ob es bereits Projekte im D-Grid mit ähnlicher Zielsetzung gibt und inwieweit diese die im vorigen Kapitel gestellten Anforderungen erfüllen.

Das **fünfte Kapitel** beschreibt den Entwurf des Systems und der Schnittstellen. Dazu gehört zunächst der konzeptuelle Entwurf des Systems und einer geeigneten VO-Struktur. In einem weiteren Schritt wird der Objektentwurf des VO-Layers beschrieben.

Das **sechste Kapitel** beschreibt zunächst das Testbett, in dem das aus dem vorigen Kapitel erstellte System als Prototyp implementiert wird. Anschließend werden die wichtigsten Teile des Quellcodes vorgestellt und erklärt. Am Ende wird noch ein Leitfaden zum Kompilieren und Installieren des VO-Layers auf einem Zielsystem gegeben.

Das **siebte Kapitel** bewertet das entworfene VO-Layer System und die Proof-of-Concept (PoC) Implementierung desselbigen. Die Bewertung erfolgt dabei anhand der in der Anforderungsanalyse definierten Anforderungen.

Das abschließende **achte Kapitel** fasst die Ergebnisse dieser Arbeit nochmals zusammen, gibt einen Ausblick und Anregungen für weiterführende Arbeiten.

# 2 Grundlagen

In diesem Kapitel werden alle wesentlichen Begriffe und Konzepte erklärt, die für das Verständnis dieser Arbeit von Bedeutung sind. Allen voran wird der Begriff *Grid* beziehungsweise *Grid-Computing* eingeführt und erklärt. Ein wichtiges Konzept in Grids ist die *virtuelle Organisation* (VO), welche in einem weiteren Abschnitt erklärt wird. Anschließend werden die Begriffe *Authentifizierung* und *Autorisierung* eingeführt und anhand von grundlegenden Verfahren erklärt. Aufgrund des engen Bezugs dieser Arbeit zum D-Grid, wird der Aufbau und die Ziele des D-Grids in einem eigenen Abschnitt erklärt und die Heterogenität innerhalb des D-Grids anhand von Beispielen aufgezeigt. Die restlichen Abschnitte befassen sich mit der Einführung in die drei Grid Middleware-Technologien *Globus Toolkit*, *UNICORE* und *gLite*.

## 2.1 Grid-Computing

Der Begriff Grid wurde ursprünglich vor allem für das Stromnetz (engl. power grid) verwendet. Dass der Begriff Grid auch im IT-Bereich zur Anwendung kam, liegt daran, dass ein Computer-Grid einem Benutzer ebenso einfach Ressourcen (wie beispielsweise Rechenleistung oder Speicherplatz) über ein Netzwerk zur Verfügung stellen soll, wie es möglich ist, Strom aus einer Steckdose zu beziehen. Der Benutzer übergibt seinen Auftrag über genormte Schnittstellen an das Grid, woraufhin die Ressourcenallokation oder die Berechnung automatisch erfolgt.

Der Begriff Grid-Computing wurde erstmals 1998 in dem Buch "The Grid: Blueprint for a New Computing Infrastructure" [FoKe 98] vorgestellt. In diesem Buch definierten Foster und Kesselman ein Grid wie folgt:

> "A computational grid is a hardware and software infrastructure, that provides dependable, consistent, pervasive, and inexpensive access to high-end computational capabilities."

Diese frühe Definition erwies sich aber als nicht hinreichend, um alle Eigenschaften eines Computer-Grids wiederzugeben. Aus diesem Grund erweiterte Foster im Jahr 2002 die Definition von Grids in "What is the Grid? A Three Point Checklist" [Fost 02], wonach ein Grid ein System ist, welches:

1. Ressourcen kontrolliert, die nicht von einer zentralen Instanz verwaltet werden,
2. standardisierte, offene und multifunktionsfähige Protokolle und Schnittstellen verwendet und

3. nichttriviale Dienstgüte anbietet.

Ein Computer-Grid verfolgt dabei andere Ziele als etwa ein Rechen-Cluster und ermöglicht eine effiziente Ressourcenausnutzung, indem Ressourcen, die geografisch und administrativ verteilt sein können, zu einem Grid zusammengeschaltet werden. Foster, Kesselman und Tuecke beschreiben in "The Anatomy of the Grid" [FKT 01] das wichtigste Ziel eines Grids wie folgt:

> "The real and specific problem that underlies the Grid concept is coordinated resource sharing and problem solving in dynamic, multi-institutional virtual organizations."

Dabei geht es in einem Grid nicht nur um einfachen Datenaustausch, sondern auch um direkten Zugriff auf Rechner, Software und andere Ressourcen im Grid. Dazu definiert der Ressourcen-Anbieter (engl. resource provider, RP), welche Ressourcen im Grid bereitgestellt werden, wer Zugriff darauf besitzt und unter welchen Bedingungen der Zugriff gestattet ist.

Die heute gängige Softwarearchitektur für Grids ist die so genannte *Open Grid Service Architecture* (OGSA) [FKS+ 06], welche vom Open Grid Forum[1] spezifiziert wurde. Deren Grundidee ist die Darstellung von beteiligten Komponenten (Rechner, Speicherplatz, Mikroskope, Teleskope u.s.w.) als Grid-Dienste in einer offenen Komponentenarchitektur. Mit der Einführung von Web-Services (WS) in Grids wurde das Wort *Grid-Services* zum Synonym für Web-Services, welche Grid-Funktionalitäten ermöglichen. OGSA schlägt in diesem Zusammenhang den Einsatz des *Web Services Resource Framework* (WSRF) als grundlegenden Baustein für Service-Grids vor. Das WSRF ist eine OASIS[2] Spezifikation für Web-Services (mehr dazu unter [CFF+ 04]). So bekommen Web-Services zusätzlich noch einen Zustand - sie werden *stateful*. Erst so ist es möglich, Funktionalitäten auszuführen, die sich über mehrere Transaktionen erstrecken. Eine Beschreibung, wie eine dienstorientierte Architektur in einer Grid-Umgebung implementiert werden kann gibt [FKNT 02].

Grids waren nicht von Anfang an dienstorientiert, allerdings soll im Rahmen dieser Einführung nicht näher auf die geschichtliche Entwicklung eingegangen werden. Einen Überblick über die Evolution von Grids und die Entwicklung der ersten Grid Middleware-Technologien gibt [RBJS 01]. Die Arbeit [ScNi 02] beschäftigt sich mit den zehn wichtigsten Fragen in Bezug auf Grids, welche geklärt werden müssen, damit *das Grid* Realität wird und nicht nur eine Nische in der Wissenschaft besetzt.

Die Grid-Architektur welche in [FKT 01] beschrieben wird, ist in Abbildung 2.1 dargestellt. Die Architektur besteht aus fünf Schichten (engl. layer), die jeweils einen Satz von Protokollen definieren. Diese fünf Schichten werden im Folgenden kurz erklärt:

**Fabric Layer** In dieser Schicht befinden sich alle Ressourcen, welche über das Grid bereitgestellt werden. Hier werden die lokalen, ressourcenspezifischen Operationen implementiert, welche von den höheren Schichten verwendet werden können.

**Connectivity Layer** Hier werden Kommunikations- und Authentifizierungs Protokolle definiert, welche für die Grid-spezifischen Netzwerktransaktionen benötigt werden. Insbe-

[1] http://www.ogf.org/
[2] http://www.oasis-open.org/

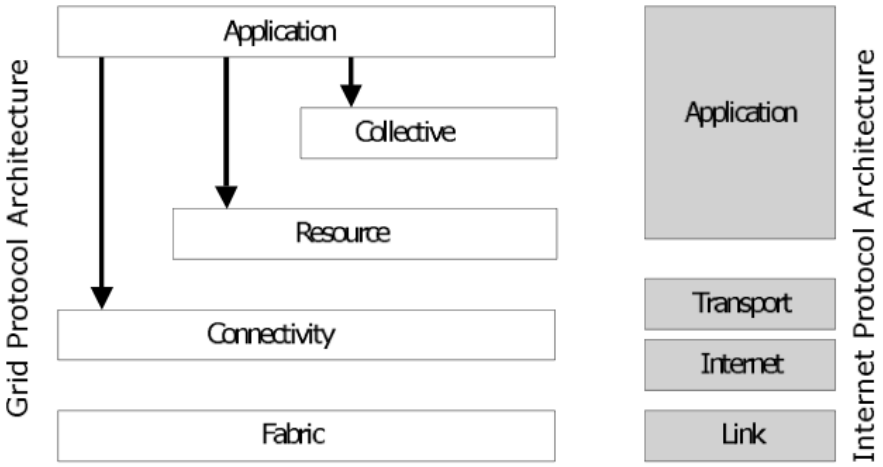



Abbildung 2.1: Die Grid Architektur und ihre Beziehung zur Internet Protokoll Architektur [FKT 01].

sondere sollte diese Schicht auch Single-Sign-On (SSO), Delegation, Autorisierung und lokale Sicherheitskonfigurationen unterstützen. Für eine genaue Erklärung von SSO und Delegation siehe Abschnitt 2.3.2 in diesem Kapitel.

**Resource Layer** Diese Schicht definiert im Wesentlichen zwei Klassen von Protokollen. Zum einen Informationsprotokolle und zum anderen Managementprotokolle. Die Informationsprotokolle werden dazu benötigt, um den Zustand einer Ressource beobachten zu können. Die Managementprotokolle ermöglichen den Zugriff auf die Ressourcen und das Starten von Programmen (im Grid-Umfeld *Jobs* genannt) auf den Ressourcen.

**Collective Layer** Diese Schicht enthält Dienste und Protokolle für eine Menge von Ressourcen. Es gibt Dienste zum Monitoring und Finden von Grid-Ressourcen, oder aber Dienste wie Directory Services oder Data Replication Service.

**Application Layer** In dieser Schicht befinden sich alle Benutzeranwendungen, die innerhalb einer VO operieren. Die Pfeile in Abbildung 2.1 deuten darauf hin, dass Programme des Application Layers unmittelbar den Connectivity-, Resource- oder Collective-Layer ansprechen können.

Um Grid-Computing langfristig zum Erfolg zu verhelfen, ist es wichtig, dass für den Connectivity- und Resource-Layer (in zweiter Linie auch für den Collective-Layer), einheitliche Protokolle entwickelt werden. Erst diese so genannten *Intergrid-Protokolle* ermöglichen den verschiedenen Institutionen, dem Grid Ressourcen zur Verfügung zu stellen und andere Ressourcen zu nutzen.

## 2.2 Virtuelle Organisation

Der Begriff virtuelle Organisation (VO) für Grids wird oft noch sehr unterschiedlich definiert und eine allgemein gültige Definition hat sich in der Literatur noch nicht durchgesetzt.

Zwar sind die VO-Definitionen in der Literatur durchaus zutreffend, allerdings meist nur für einen bestimmten Anwendungsbereich und sind deshalb in der Regel nicht allgemein gültig. Die folgende Definition stammt von Schiffers, der sich in seiner Dissertation "Management dynamischer virtueller Organisationen in Grids" [Schi 07] eingehend mit dem VO-Begriff befasst hat:

> "Eine virtuelle Organisation ist eine zeitlich begrenzte koordinierte Kooperation von Elementen in Form von Individuen, Gruppen von Individuen, Organisationseinheiten oder ganzer Organisationen, die Teile ihrer physischen oder logischen Ressourcen oder Dienste auf diesen, ihre Kenntnisse und Fähigkeiten sowie Teile ihrer Informationsbasis in Form virtueller Ressourcen und Dienste über eine Grid-Infrastruktur derart zur Verfügung stellen, dass die gemeinsam vereinbarten Ziele unter Berücksichtigung lokaler und globaler Policies erreicht werden können."

Ein Vorteil der VOs ist die vereinfachte Autorisierung der Benutzer auf die Ressourcen. Der Besitzer einer Ressource kann diese dem Grid zu Verfügung stellen und dabei genau festlegen wer, wann und was mit der Ressource machen darf. Die Mitglieder einer VO erhalten Zugriff auf die Ressourcen innerhalb dieser VO und können auf diese - nach den entsprechenden Regeln - zugreifen. Wegen der Dynamik der VOs können Zugriffsregeln auf Ressourcen nicht für jeden Benutzer einzeln definiert werden. Man bedient sich hierbei dem Konstrukt der Gruppen und Rollen. Somit kann der Zugriff auf eine Ressource gestattet werden, wenn der Benutzer Teil einer bestimmten Gruppe ist (z.B. Forscher), oder wenn er eine bestimmte Rolle in der VO inne hat (z.B. VO-Admin). Abbildung 2.2 zeigt eine einfache VO die aus Mitarbeitern und Ressourcen von zwei realen Organisationen zusammenstellt ist.

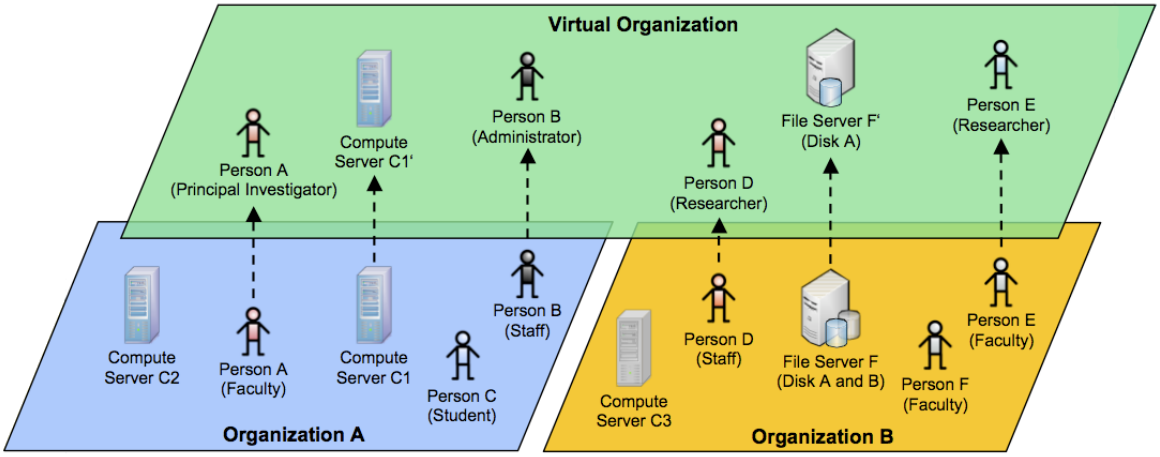



Abbildung 2.2: Beispiel einer einfachen VO [Schi 08].

VOs können auch komplexe Strukturen annehmen. So ist des durchaus möglich, dass ein Benutzer Teil von mehreren VOs ist, in diesen aber zu unterschiedlichen Gruppen gehört. Des Weiteren ist es möglich, dass eine VO mehrere Sub-VOs beinhalten kann, oder dass sich VOs überlappen (sie teilen sich die selben Benutzer/Ressourcen). Auch die Ressourcen können beliebig auf die VOs verteilt werden. Beispielsweise ist es möglich, dass von einem

Supercomputer mit 16 Rechenknoten für eine VO nur zwei Knoten nutzbar sind, für eine andere VO hingegen alle.

## 2.3 Authentifizierung

Authentifizierung bezeichnet die Überprüfung beziehungsweise das Beweisen einer Identität. Dabei gibt es im Wesentlichen drei verschiedene Klassen von Authentifizierungsverfahren:

- Authentifizierung durch Kenntnis bestimmter Informationen (z.B. Passwort).
- Authentifizierung durch den Besitz bestimmter Hardware (z.B. Schlüssel).
- Authentifizierung durch eine bestimmte Eigenschaft (z.B. biometrisches Merkmal).

### 2.3.1 Grundlegende Verfahren

In diesem Abschnitt werden kurz die grundlegenden Authentifizierungsverfahren beschrieben. Dabei wird ausschließlich auf Verfahren eingegangen, welche die Kenntnis bestimmter Informationen voraussetzen. Authentifizierung durch Besitz von Hardware oder durch biometrische Merkmale wird im Folgenden nicht weiter untersucht, da diese beiden Klassen in der Informationstechnologie keine große Bedeutung besitzen (Biometrische Authentifizierungs-Verfahren könnten in Zukunft allerdings an Bedeutung gewinnen). Jedoch wird keines der hier vorgestellten Verfahren in Grid-Umgebungen eingesetzt. Die in Grids verwendeten Verfahren werden in Kapitel 4.1 eingeführt und untersucht. Für eine ausführliche Beschreibung der im Folgenden vorgestellten Konzepte und Verfahren sei auf das Buch "IT-Sicherheit" von Eckert verwiesen [Ecke 05].

#### Username/Password

Das *Username/Password* Verfahren [Ecke 05] ist vermutlich das älteste Authentifizierungsverfahren in der Informationstechnologie. Dabei wird auf dem System, wo sich der Benutzer authentifizieren soll, eine Tabelle mit allen bekannten Benutzern und Passwörtern gespeichert. Bei jedem Authentifizierungsversuch wird die eingegebene Kombination aus Benutzername und Passwort mit dem Einträgen in der Tabelle verglichen und so wird die Identität des Benutzers bestätigt oder der Anmeldevorgang zurückgewiesen. Die Passwörter liegen auf dem Zielsystem in der Regel verschlüsselt oder als Hashwert vor, womit ein unbefugtes Auslesen der Passwörter verhindert wird.

Für Grid-Anwendungen eignet sich das *Username/Password* Verfahren nicht, da die Sicherheit des Systems allein von der Geheimhaltung und der Komplexität des Passworts abhängt und damit für Grid-Anwendungen zu unsicher ist.

### Challenge/Respond

Das *Challenge/Respond* Verfahren [Ecke 05] ist dem *Username/Password* Verfahren sehr ähnlich. Auch hier wird auf dem Zielsystem eine Tabelle mit allen Benutzern und geheimen Authentifizierungsdaten gespeichert. Der wesentliche Unterschied besteht darin, dass die geheime Information nicht statisch, sondern dynamisch ist. Eine weit verbreitete Implementierung ist eine nummerierte TAN[3]-Liste. Hierbei wird der Benutzer gebeten, eine bisher nicht benutzte TAN aus der Liste einzugeben. Dieses Verfahren wird häufig in Kombination mit dem *Username/Password* Verfahren beim Online Banking verwendet und wird dort als PIN/TAN Verfahren bezeichnet.

Im Vergleich zum *Username/Password* Verfahren bietet das *Challenge/Respond* Verfahren bereits eine erhöhte Sicherheit, allerdings steigt dadurch auch der Verwaltungsaufwand für das Authentifizierungssystem. Aus diesem Grund wird dieses Verfahren in Grids zugunsten anderer Verfahren, welche in Kapitel 4.1 eingeführt werden, nicht verwendet.

### Public-Key-Verfahren

Das *Public-Key-Verfahren* [Ecke 05] basiert auf so genannten asymmetrischen Chiffren. Bei diesen Chiffren gibt es stets einen öffentlichen und einen privaten Schlüssel. Nur wer in Besitz dieser beiden Schlüssel ist, kann eine Nachricht sowohl verschlüsseln als auch entschlüsseln. Dieses Verfahren beruht auf der Tatsache, dass aus dem einen Schlüssel alleine der andere Schlüssel - nach heutigem Kenntnisstand - nicht effizient berechnet werden kann. Zum Authentifizieren verschlüsselt der Benutzer eine Nachricht mit seinem privaten Schlüssel. Diese Nachricht kann nun von jedem mit dem öffentlichen Schlüssel entschlüsselt werden, somit kann jeder die Identität des Benutzers überprüfen. Die Nachricht, die zur Authentifizierung verwendet wird, nennt man in diesem Zusammenhang *Zertifikat*. Dieses enthält unter anderem Informationen über den Besitzer, den Aussteller, die Verfahren welche zur Erstellung dienten und die Gültigkeitsdauer.

Für die Authentifizierung wird in der Praxis nicht das gesamte Zertifikat verschlüsselt, sondern lediglich der Hashwert davon, der an das Ende des Zertifikats angehängt wird. Man spricht nun von der *Signatur* einer Nachricht. Damit das *Public-Key-Verfahren* auch für sicherheitskritische Szenarien verwendet werden kann, wird eine so genannte *Trusted-Third-Party* (TTP) benötigt. Deren Aufgabe ist es, die öffentlichen Schlüssel zu verwalten und sicherzustellen, dass sich hinter den öffentlichen Schlüsseln auch die entsprechenden Personen verbergen.

In Grid-Umgebungen wird das *Publik-Key-Verfahren* eingesetzt, allerdings nur in Verbindung mit einer TTP welche im Grid *Certificate Authority* (CA) genannt wird. Das System wird als *Public-Key-Infrastruktur* (PKI) bezeichnet und verwendet zur Authentifizierung X.509 Zertifikate. X.509 ist ein ITU[4] Standard für eine PKI und ist derzeit der wichtigste Standard für digitale Zertifikate [X.509] .

[3] Transaktionsnummer
[4] International Telecommunication Union http://www.itu.int/

### 2.3.2 Proxy-Zertifikat, Single-Sign-On und Delegation

Damit ein Benutzer sich nicht bei jeder Aktion neu authentifizieren muss, gibt es das Konzept des **Single-Sign-On** (SSO) [Ecke 05]. Dabei muss sich der Benutzer für jede Sitzung nur einmal authentifizieren. Bei Zertifikat-basierten Authentifizierungsmechanismen erhält der Benutzer ein so genanntes **Proxy-Zertifikat** (nach RFC 3820 [TWE+ 04]) vom System, welches nur eine begrenzte Zeit gültig ist und in dieser Zeit anstelle des Benutzerzertifikats verwendet wird. Ein wichtiger Vorteil dieses Verfahrens ist, dass das Benutzerzertifikat nur noch zur Erstellung des Proxy-Zertifikats verwendet wird. Dadurch wird es für Angreifer aufgrund der spärlichen Daten noch schwieriger den Private-Key des Benutzers zu rekonstruieren. Bei der **Delegation** [Ecke 05] kann ein Benutzer Aufgaben - oder Teile davon - an andere Benutzer weiterleiten, welche dann die Aufgaben abarbeiten oder diese ihrerseits weiter delegieren. Diese Verfahren werden in verschiedenen Grid Middleware-Technologien als Ergänzung zu einer PKI verwendet.

## 2.4 Autorisierung

In der Informationstechnologie bezeichnet die Autorisierung die Zuweisung und Überprüfung von Zugriffsrechten auf Ressourcen. In einem System wird dadurch sichergestellt, dass Benutzer nicht gegen geltende Sicherheitsrichtlinien verstoßen. Die Autorisierung setzt üblicherweise die Authentifizierung des Benutzers voraus. Alle hier beschriebenen Autorisierungs-Verfahren sind ausführlich im Buch "IT-Sicherheit" von Eckert [Ecke 05] beschrieben.

### 2.4.1 Access Controll List (ACL)

Ein weit verbreitetes Konzept zur Autorisierung ist die so genannte *Access Controll List* beziehungsweise Zugriffskontrollliste [Ecke 05]. Dabei wird für jeden Benutzer genau festgelegt, auf welche Ressourcen er zugreifen darf. Diese Informationen werden entweder lokal auf jedem System gespeichert oder aber zentral, bei einem speziellen Autorisierungsdienst. In der Betriebssystemwelt werden häufig ACLs verwendet, um den Zugriff der Benutzer auf Dateien und Prozesse zu steuern. Dabei besitzt jedes Objekt eine eigene ACL, die für jedes Subjekt die entsprechenden Zugriffsrechte enthält.

Mit ACLs kann einfach festgestellt werden, welche Subjekte auf ein bestimmtes Objekt zugreifen dürfen. Umgekehrt aber, muss man alle ACLs durchsuchen, wenn man für ein Subjekt alle seine Zugriffsrechte ermitteln will.

In Grid-Umgebungen werden ACLs häufig zur Autorisierung verwendet. Ein Beispiel dafür ist das so genannte *gridmap file* im Globus Toolkit welches in Abschnitt 4.2.1 genauer vorgestellt wird.

### 2.4.2 Capability List

Die *Capability List* (Zugriffsausweise) [Ecke 05] basiert auf dem selben Konzept wie die ACL. Allerdings wird hierbei jedem Subjekt eine Liste von Objekten zugeordnet, auf die zugegriffen werden darf. Somit ist es einfach, alle Objekte zu bestimmen, auf die ein Subjekt zugreifen darf. Umgekehrt müssen aber alle Capability Lists durchsucht werden, um alle berechtigten Subjekte für ein bestimmtes Objekt zu ermitteln.

Capability Lists haben sich nie durchsetzen können. Die meisten heutigen Betriebssysteme und auch Grid Middleware-Technologien benutzen ACLs.

### 2.4.3 Weitere Autorisierungs-Verfahren

Es gibt noch weitere Autorisierungs-Konzepte wie die **Discretionary Access Control** (DAC), wo die Zugriffsrecht für die Objekte pro Benutzer festgelegt werden. Ein anderes Konzept ist die **Mandatory Access Control** (MAC), welche den Zugriff aufgrund der Identität des Benutzers und zusätzlicher Informationen, wie etwa einer Sicherheitshierarchie (vertraulich, geheim, streng geheim), steuert. Da für einige Anwendungen DAC zu schwach und MAC zu restriktiv ist, wurde Anfang der 90er die **Role-Based Access Control** (RBAC) entwickelt. Dabei werden einem Benutzer verschiedene Rollen zugeteilt, aufgrund derer er auf Ressourcen zugreifen darf. RBAC ist sehr flexibel und insbesondere bei einer großen Benutzer- und Ressourcenanzahl von Vorteil. Ein weiteres Konzept ist die **Attribute-Based Access Control** (ABAC), wo die Zugriffsrechte zwischen Subjekten und Objekten nicht statisch, sondern dynamisch sind. Die Attribute, auch *Credential* genannt, können dabei allgemeine Eigenschaften sein (z.B. Position oder Fähigkeiten) aber es kann beispielsweise auch der Standort des Benutzers - für eine mobile Anwendung - als Attribut abgebildet werden.

Für weitere und ausführlichere Informationen zu DAC, MAC, RBAC und ABAC sei auf [DEF+ 06a] und [Ecke 05] verwiesen.

## 2.5 D-Grid

Die vom Bundesministerium für Bildung und Forschung (BMBF) im Jahr 2004 initiierte D-Grid Initiative[5] hat sich zu Ziel gesetzt, eine nachhaltige Grid-Infrastruktur in Deutschland aufzubauen. Dazu wird eine Grid-Infrastruktur für Projekte aus verschiedenen wissenschaftlichen Gebieten, mit den benötigten Diensten und einer Benutzerverwaltung aufgebaut. Die ersten D-Grid Projekte starteten im Jahr 2005 und umfassten die Entwicklung und Implementierung von IT-Diensten für die Wissenschaft. So genannte Grid-Communities testeten die globale Dienste-Infrastruktur mit ihren Anwendungen aus den Gebieten der Hochenergiephysik, Astrophysik, alternative Energien, Medizin, Klimaforschung, Ingenieuranwendungen und Geisteswissenschaften.

[5] http://www.d-grid.de/

Die zweite Phase der D-Grid Initiative läuft seit 2007 und soll voraussichtlich 2010 abgeschlossen sein. Geplant sind dabei weitere Schritte zur Erweiterung der D-Grid Infrastruktur, durch die Einführung professioneller Betriebskonzepte, Service-Level-Agreements (SLA), einer Wissensschicht, dem Aufbau von virtuellen Kompetenzzentren, die Anbindung Service-orientierter Architekturen (SOA) der Industrie und die Bereitstellung von Grid-Ressourcen zum Nutzen der ganzen Gesellschaft.

Am D-Grid sind verschiedenen Communities beteiligt, welche das Ziel haben, eine Grid Infrastruktur für den jeweiligen Wissenschaftsbereich zu etablieren. Zusätzlich zu den Communities wird auch noch das D-Grid Integrationsprojekt (DGI[6]) gefördert, das die Aufgabe hat, die Entwicklungen der Community-Projekte aufzunehmen, zu verallgemeinern und in die allgemeine D-Grid Plattform zu integrieren. Abbildung 2.3 stellt graphisch dar, wie die einzelnen Community Projekte auf dem Integrationsprojekt aufbauen.

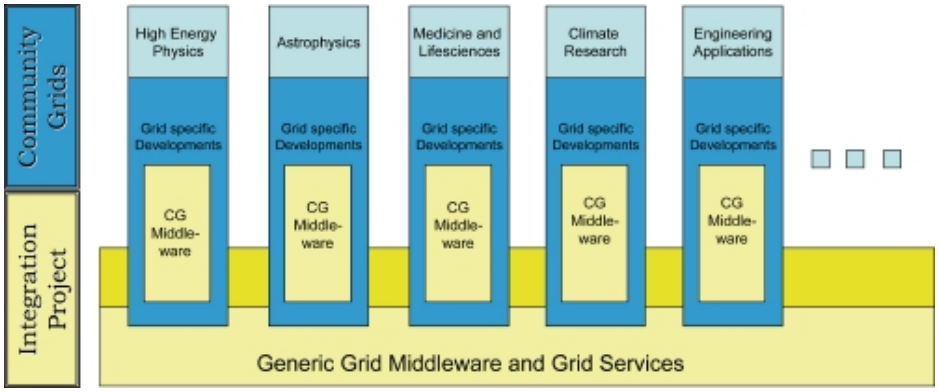



Abbildung 2.3: Überblick über die D-Grid Projekte (Quelle: www.d-grid.de).

Die entstehende Plattform und Grid-Infrastruktur ist dabei offen für neue Projekte aus den verschiedensten Bereichen der Wissenschaft. Diese werden sich im Laufe der Zeit neben den fünf "Pionier" Communities etablieren, indem die gemeinsame Plattform - die D-Grid Dienste und die Infrastruktur des Integrationsprojektes - verwendet wird. Die Communities sind weitestgehend autonom und verwenden teilweise unterschiedliche Grid Middleware-Technologien und VO-Konzepte. Im Folgenden werden drei D-Grid Communities vorgestellt, welche alle unterschiedliche Grid Middleware-Technologien und VO-Konzepte verwenden. Nähere Informationen zum D-Grid und allen seinen Communities gibt [NKG 07].

**AstroGrid-D** Das Hauptziel des AstroGrid-D[7] ist die Einbindung der astronomischen Forschungsinstitute in Deutschland in eine einheitliche Grid-basierte Infrastruktur, um verteiltes, kollaboratives Arbeiten zu fördern. Das resultierende AstroGrid-D wird bereits bestehende, geografisch verteilte Hochleistungsrechner mit großen astronomischen Datenarchiven, ferngesteuerten Radio- und Roboter-Teleskopen sowie bodengebundenen Gravitationswellen-Detektoren in einem kohärenten Rechen- und Daten-Grid integrieren. Das AstroGrid-D verwendet Globus Toolkit als Grid-Middleware und benutzt zur Autorisierung VOs, sub-VOs und Gruppen.

[6] http://dgi.d-grid.de/
[7] http://www.gac-grid.org/

**HEPCG** Das Hochenergiephysik-Computing-Grid[8] (HEPCG) ergänzt die Software des Welt-LHC-Grids (WLHCG) in den Bereichen Datenverwaltung, Job-Überwachung und verteilte Datenanalyse. Das WLHCG ist eine weltweit verteilte Grid Infrastruktur, basierend auf der gLite Grid Middleware-Technologie, die dazu dient, die enormen Datenmengen der LHC-Experimente am CERN zu analysieren. Im HEPCG kommt die Grid-Middleware gLite zum Einsatz, in der VOs unterstützt werden und rollenbasierte Autorisierung zum Einsatz kommt.

**BauVOGrid** Das BauVOGrid[9] ist eine Grid-basierte Plattform für die virtuelle Organisation im Bauwesen. Da speziell im Bauwesen jedes Projekt einmalig in Bezug auf Zusammenarbeit und Kooperation der beteiligten Unternehmen und der vertraglichen Bindungen untereinander ist, gibt es spezielle Anforderungen an die Effizient und die Verwaltung virtueller Organisationen. Damit diese hohen Anforderungen erfüllt werden können, bedarf es einer Grid-Plattform und Grid-basierter Services. Im BauVOGrid werden die Grid-Middlewares Globus Toolkit und UNICORE, sowie eine feingranulare Autorisierung mit Hilfe von VOs und Rollen verwendet.

## 2.6 Globus Toolkit

Das Globus Toolkit (kurz Globus) ging aus dem I-WAY Projekt von 1995 hervor und ist eine von der *Globus Alliance*[10] entwickelte Sammlung von Software Komponenten, welche es ermöglichen, Grid-basierte Anwendungen zu erstellen. Seit der Version 4.0 unterstützt das Globus Toolkit die *Open Grid Service Architecture* (OGSA) [FKS+ 06] und verbessert dadurch die Interoperabilität zu jenen Grid Middleware-Technologien die ebenfalls OGSA unterstützen. Im Wesentlichen besteht das Globus Toolkit aus fünf unterschiedlichen Teilbereichen: *Security*, *Data Management*, *Execution Management*, *Information Services* und *Common Runtime*. In Abbildung 2.4 sind alle Komponenten dieser fünf Teilbereiche dargestellt. Für eine Ausführliche Einführung in das Globus Toolkit der Version 4.0 sei auf [Fost 05] verwiesen. Für Entwickler empfiehlt sich das Buch "Globus Toolkit 4: Programming Java Services" [SoCh 05]. Im Rahmen dieser Diplomarbeit wird lediglich auf die Version 4.0 des Globus Toolkits (auch GT4 genannt) eingegangen.

**Security** Das Globus Toolkit stellt mit der *Grid Security Infrastructure* (GSI) [FKTT 98] ein Tool zum Aufbau einer sicheren Grid Infrastruktur zur Verfügung. Mit Hilfe der GSI ist es möglich, Proxy-Zertifikate zu erstellen. Globus bietet somit die Möglichkeit von Single-Sign-On und Delegation.

**Data Management** Dieser Teil ist für das Auffinden, den Transfer und den Zugriff auf große Daten zuständig. Die Datenübertragung erfolgt über ein modifiziertes FTP wie etwa *GridFTP*.

**Execution Management** Hiermit wird die Ausführung, das Scheduling und das Monitoring von Programmen, welche im Grid als *Jobs* bezeichnet werden, ermöglicht. Ein Job

[8] http://www.hepcg.org/
[9] http://www.bauvogrid.de
[10] http://www.globus.org

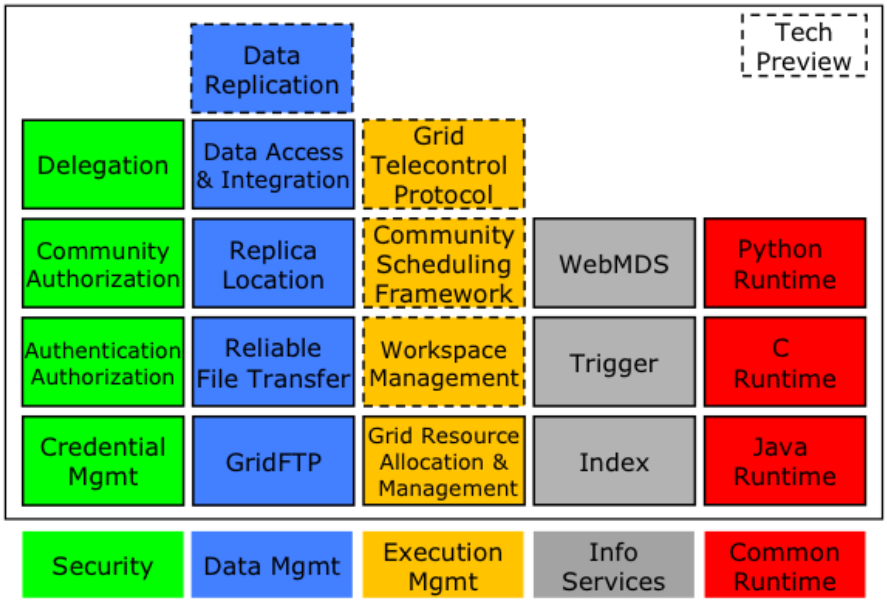



Abbildung 2.4: Globus Toolkit 4 Komponenten [Fost 05]

kann auch aus mehreren Sub-Jobs bestehen, zwischen denen Abhängigkeiten existieren können.

**Information Services** Dieser Teil wird im Globus Toolkit auch als *Monitoring and Discovery System* (MDS) bezeichnet und ermöglicht das Monitoring und das Auffinden von Ressourcen in virtuellen Organisationen.

**Common Runtime** Darin enthalten sind die fundamentalen Bibliotheken und Tools, welche für das Bereitstellen von existierenden Diensten und zum Entwickeln von neuen Diensten benötigt werden.

Die **Authentifizierung** basiert in GT4 auf X.509 Zertifikaten. Sobald ein Benutzer erfolgreich authentifiziert wurde, erhält er von Globus ein zeitlich begrenzt gültiges Proxy-Zertifikat. Die **Autorisierung** überprüft ob ein Benutzer berechtigt ist, auf einen bestimmten Dienst im Grid zuzugreifen. Dazu gibt es verschiedene Mechanismen, wobei im Rahmen dieser Arbeit nur das so genannte **gridmap file** benötigt wird. Eine kurze Beschreibung diese Konzeptes gibt das Kapitel 4.2.1. Für eine ausführliche Beschreibung aller Autorisierungs-Mechanismen in Globus Toolkit 4 sei auf [SoCh 05] verwiesen.

Globus integriert aber nicht alle Funktionen, die man sich für eine Grid-Middleware wünscht. So bietet die Software kein VO-Konzept, keine Fehlertoleranz und keine Accounting Funktionalität. Allerdings existieren Erweiterungen hierfür, die sich jedoch teilweise noch im Entwicklungsstadium befinden.

Die Installation des Globus Toolkits erweist sich als relativ komplex, da die verschiedenen Komponenten alle einzeln konfiguriert werden müssen (siehe dazu Kapitel A.1). Globus

wird ausschließlich über die Kommandozeile bedient und besitzt keine grafische Benutzeroberfläche. Allerdings gibt es Erweiterungen, wie die webbasierte Plattform *GridSphere*[11], welche eine GUI zur Verfügung stellen.

## 2.7 UNICORE

UNICORE ist ein Akronym und steht für *UNiform Interface to COmputing REsources*. Diese Grid-Middleware wird maßgeblich vom Forschungszentrum Jülich mit Unterstützung durch das Bundesministerium für Bildung und Forschung entwickelt. Zur Zeit wird im D-Grid vor allem noch UNICORE 5 verwendet, obwohl im August 2007 die finale Version von UNICORE 6 veröffentlicht wurde. Diese neueste Version ist eine große Weiterentwicklung der UNICORE Plattform und unterstützt Web-Services welche den OGSA Standard verwenden. Einen genauen Einblick in die UNICORE Funktionalität und Architektur bietet der Abschlussbericht des UNICORE-Plus Projektes [Erwi 03], welches sich mit der Entwicklung von UNICORE bis Version 4.1 beschäftigt hat. Einen kurze Einführung in UNICORE Version 6 gibt [Schu 07].

Im Gegensatz zum Globus Toolkit ist UNICORE ein Komplettpaket, welches alle notwendigen Komponenten integriert. Dadurch wird die Installation enorm vereinfacht und Benutzer können über ein einziges Client Programm auf das Grid zugreifen. Dadurch, dass der UNICORE Client und die Server Komponenten in Java geschrieben sind, unterstützt UNICORE viele verschiedene Betriebssysteme, darunter Unix, Windows und MacOS. Die einzelnen Komponenten sind einfach zu installieren (siehe dazu Kapitel A.2) und wegen der GUI-Unterstützung auch einfach zu konfigurieren. In dieser Diplomarbeit wird nur UNICORE Version 6 analysiert und verwendet und im Folgenden der Einfachheit halber lediglich mit *UNICORE* bezeichnet.

Das **Client** Programm in UNICORE besitzt eine grafische Oberfläche und der Benutzer kann mit ein paar Mausklicks Jobs erstellen und abschicken, aber auch Statusinformationen und Ergebnisse abfragen. In UNICORE gibt es drei verschiedene Clients:

**Application Client** Das ist eine einzelne, leicht zu bedienende GUI Anwendung. Der Client ermöglicht es, Jobs zu erstellen, abzuschicken und die Ergebnisse aufzurufen. Dank der Java Technologie kann dieser Client auch über einen Web Browser gesteuert werden.

**Expert Client** Er besitzt die selben Funktionen wie der Application Client, zusätzlich unterstützt er Workflows, mit denen die Ausgabe eines Jobs direkt als Eingabe für den nächsten Job verwendet werden kann. Dadurch lassen sich Jobketten erstellen, ohne dass der Benutzer zwischendurch eingreifen muss. Der Expert Client ersetzt den UNICORE 5 Client und kann ebenfalls über einen Web Browser gesteuert werden.

**Command Line Client** Dieser wird über die Kommandozeile bedient und kann einzelne Jobs starten, aber auch im Batch Modus arbeiten. Er kann Dateien von Local zu Remote oder von Server zu Server übertragen.

So genannte **GridBeans** erweitern die Clients um zusätzliche Funktionen. GridBeans erstel-

[11] http://www.gridsphere.org/

len Jobs für die vorgegebenen Aufgaben und die Konfiguration dieser Jobs erfolgt über eine GUI. Gegenüber den Vorgängern der GridBeans - den UNICORE Plugins unter UNICORE 5 - sind diese als Web-Services aufgebaut.

Neben dem Client Programm gibt es den UNICORE **Server**, welcher alle Komponenten der UNICORE Systemarchitektur integriert (siehe dazu Abbildung 2.5). UNICORE ermöglicht es den Anbietern von Ressourcen, die volle Kontrolle über ihre Ressourcen zu behalten. Dazu kann jeder Ressourcenanbieter seine Ressourcen in einer so genannten **UNICORE Site** (USite) zur Verfügung stellen. Die einzelnen Ressourcen können wiederum in so genannte **Virtual Sites** (VSite) eingeteilt werden. Wenn ein Client einen Job an einen UNICORE Server zur Ausführung schickt, liegt dieser als so genanntes **Abstract Job Object** (AJO) vor. Dieses AJO ist eine Plattform- und Rechnerunabhängige Beschreibung eines Arbeitsablaufs für eine UNICORE Ressource. Das AJO passiert dann die zentrale Authentifizierungsstelle der USite, den **Gateway**. Dieser prüft die Authentizität des Benutzers welcher das AJO versendet und leitet dieses an einen lokalen **Network Job Supervisor** (NJS) einer VSite weiter (jede VSite besitzt ihren eigenen NJS). Der NJS ist der zentrale Scheduler für eine VSite, er empfängt AJOs vom Gateway, übersetzt diese - anhand der Informationen in der **Incarnation Database** - in einen Batch Job für das Zielsystem und sendet den Batch Job an das **Target System Interface** (TSI). Ein NJS kann auch ein sub-AJO an einen Gateway einer anderen USite schicken, um Berechnungen (oder Teile davon) dort durchzuführen. Eine weitere wichtige Aufgabe des NJS ist es, die Berechtigungen der Benutzer zu überprüfen. Dazu hat der NJS Zugriff auf die **UNICORE User Database** (UUDB), welche alle Informationen über die Autorisierung der Benutzer auf die Ressourcen der aktuellen USite beinhaltet. Sobald das übersetzte AJO an das TSI übergeben wurde, kann es am Zielsystem ausgeführt werden. Das TSI läuft dabei direkt auf der ausführenden Hardware und muss dementsprechend auch angepasst sein. In UNICORE 6 unterstützen der NJS und der UNICORE Client den OGSA Standard, wohingegen das TSI auch in UNICORE 6 nach wie vor nicht Web-Service orientiert ist.

Die **Authentifizierung** und **Autorisierung** basiert in UNICORE auf X.509 Zertifikaten, wobei es einige Einschränkungen gibt. So wird beispielsweise SSO nicht unterstützt und es gibt keine echte, sondern lediglich eine statische Delegation. Ein weiterer großer Nachteil ist das Fehlen von VOs und rollenbasierter Autorisierung. Es existiert noch kein Service Discovery und keine Abrechnungsfunktionalität, des weiteren ist UNICORE beim Ausfall einzelner Rechnerknoten nicht fehlertolerant .

Die Interoperabilität von UNICORE mit anderen Grid Middleware-Technologien ist sehr eingeschränkt. So besteht keine Interoperabilität zwischen UNICORE 5 und UNICORE 6. Auch zwischen UNICORE 6 und dem Globus Toolkit 4 besteht keine Kompatibilität, da GT4 eine ältere Version des WSRF implementiert und UNICORE 6 die neueste Version 1.2 implementiert. Zwischen UNICORE 5 und dem Globus Toolkit besteht dank des GRIP-Projektes[12] eine Interoperabilität, so lassen sich Globus Jobs in einem UNICORE 5 Grid starten - allerdings nicht umgekehrt.

Ausführliche Informationen, Dokumentationen und Tutorials finden sich auf der UNICORE Webseite unter http://www.unicore.eu/documentation/.

[12] http://www.grid-interoperability.eu/

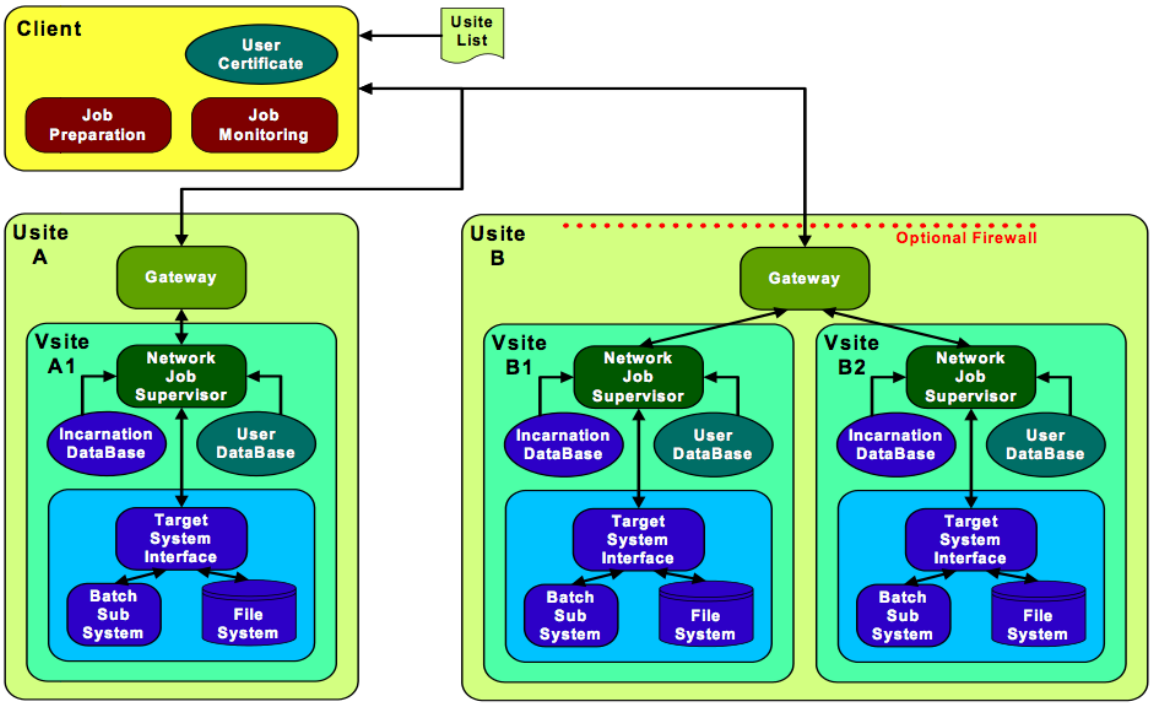



Abbildung 2.5: UNICORE Architektur [Lato 06]

## 2.8 gLite

Die Grid-Middleware gLite wurde im Rahmen des Projektes *Enabling Grids for E-sciencE* (EGEE[13]) entwickelt. EGEE ist ein Projekt der Europäischen Kommission, dessen Ziel es ist, aktuelle Forschungsergebnisse im Bereich Grid-Technologien erfolgreich für die Entwicklung einer Dienstleistungs-Grid-Infrastruktur zu verwenden. Diese soll Wissenschaftlern in Industrie und Forschung zur Verfügung gestellt werden. Eine Hauptaufgabe dieses Projektes ist die Entwicklung der Grid-Middleware gLite für den Einsatz in verschiedenen wissenschaftlichen Gebieten, allen voran aber in der Hochenergiephysik. Dabei baut gLite auf dem *LHC Computing Grid* (LCG[14]) auf, dessen Aufgabe es ist, Ressourcen für die enormen Datenmengen des *Large Hadron Collider* (LHC) am *CERN* zur Verfügung zu stellen.

gLite ist als modulares System konzipiert, welches es dem Benutzer erlaubt, die Services entsprechend den Bedürfnissen anzupassen, ohne gezwungen zu sein, das ganze System zu nutzen. Die einzelnen Services von gLite Version 3.1, welche in dieser Diplomarbeit untersucht wird, sind in Abbildung 2.6 abgebildet und werden in [JDT 05] ausführlich beschrieben. Im Folgenden sind die wichtigsten Eigenschaften dieser Services kurz zusammengefasst:

**Security Services** Beinhaltet Komponenten zur Authentifizierung, Autorisierung und zum Auditing. Die Authentifizierung basiert auf X.509 Zertifikaten und der *Grid Security*

[13] http://www.eu-egee.org/
[14] http://lcg.web.cern.ch/LCG/

*Infrastructure* (GSI) [FKTT 98]. Durch die Verwendung von Proxy-Zertifikaten wird SSO und Delegation ermöglicht. Die Autorisierung erfolgt über virtuelle Organisationen und VOMS (siehe dazu 4.4). Eine genauere Einführung in die Authentifizierungs- und Autorisierungsverfahren in gLite liefert [Henn 06].

**Information and Monitoring Services** Stellen q Mechanismen zum Erstellen und Verarbeiten von Daten für Monitoring-Zwecke bereit. Aufbauend auf den grundlegenden Funktionen dieses Dienstes, existieren stark spezialisierte Dienste wie *Job Monitoring* oder *Network Monitoring*. Zusätzlich verwenden diese Dienste ein feingranulares Autorisierungsschema, damit die Benutzer nur in ihrer jeweiligen Domäne arbeiten können.

**Job Management Services** Die wichtigsten Dienste, um einen Job ausführen und verwalten zu können sind *Computing Element*, *Workload Management*, *Accounting*, *Job Provenance* und *Package Management*. Diese Dienste kommunizieren miteinander, sodass die Informationen über den gerade ausgeführten Job stets aktuell sind.

**Data Services** Die drei wichtigsten Dienste in dieser Gruppe sind *Storage Element*, *Catalog Services* und *Data Movement*. Der Benutzer der Data Services kann darauf wie auf ein normales Dateisystem - gemäß seiner Berechtigungen - zugreifen.

**Helper Services** Grid-Infrastrukturen integrieren neben den Basisdiensten meist auch noch einige unterstützende Dienste, um damit eine höhere Abstraktionsschicht zu schaffen oder um eine bessere Dienstgüte anzubieten.

Auch in gLite schreitet die Integration mit WSRF und OGSA immer weiter voran und so wird gLite in Zukunft OGSA und WSRF unterstützen, womit die Interoperabilität mit den anderen Grid Middleware-Technologien verbessert wird.

Der Funktionsumfang von gLite ist umfangreich. So besitzt gLite einen eigenen Scheduler und die Möglichkeit, Abrechnungsfunktionen aufzuzeichnen. Weiters können Statusinformationen über einzelne Jobs, den Scheduler oder einzelne Cluster abgefragt werden. Mit dem Service Discovery hat man die Möglichkeit, die Eigenschaften und Fähigkeiten eines Clusters herauszufinden. Des Weiteren integriert gLite als einzige der in dieser Diplomarbeit betrachteten Grid Middleware-Technologien standardmäßig ein VO-Konzept.

Die **Authentifizierung** basiert in gLite auf X.509 Zertifikaten. Der Benutzer muss sich beim Grid mit Hilfe seines Benutzerzertifikates anmelden, anschließend erhält er von gLite ein begrenzt gültiges Proxy-Zertifikat. Die **Autorisierung** erfolgt über einen VOMS-Server (siehe Kapitel 4.4), somit unterstützt gLite die Autorisierung mit Hilfe von VOs. Die einzelnen Knoten in einem gLite Grid verwenden zur Autorisierung - ähnlich wie das Globus Toolkit - ein gridmap file, wobei dessen Einträge periodisch mit dem zentralen VOMS-Server abgeglichen werden.

Als Betriebssystem wird von gLite nur eine Linux Distribution (Scientific Linux) unterstützt. Die Installation gestaltet sich als vergleichsweise schwierig, da die benötigten Dienste einzeln auswählen werden müssen. Zudem müssen einige der Dienste auf getrennten Rechnern laufen. Auch die Konfiguration ist, verglichen mit UNICORE oder Globus, vergleichsweise aufwendig. Andererseits erhält man gerade durch diese vielen Freiheiten ein auf die entsprechenden Bedürfnisse angepasstes System. Zur Installation siehe auch Kapitel A.3.

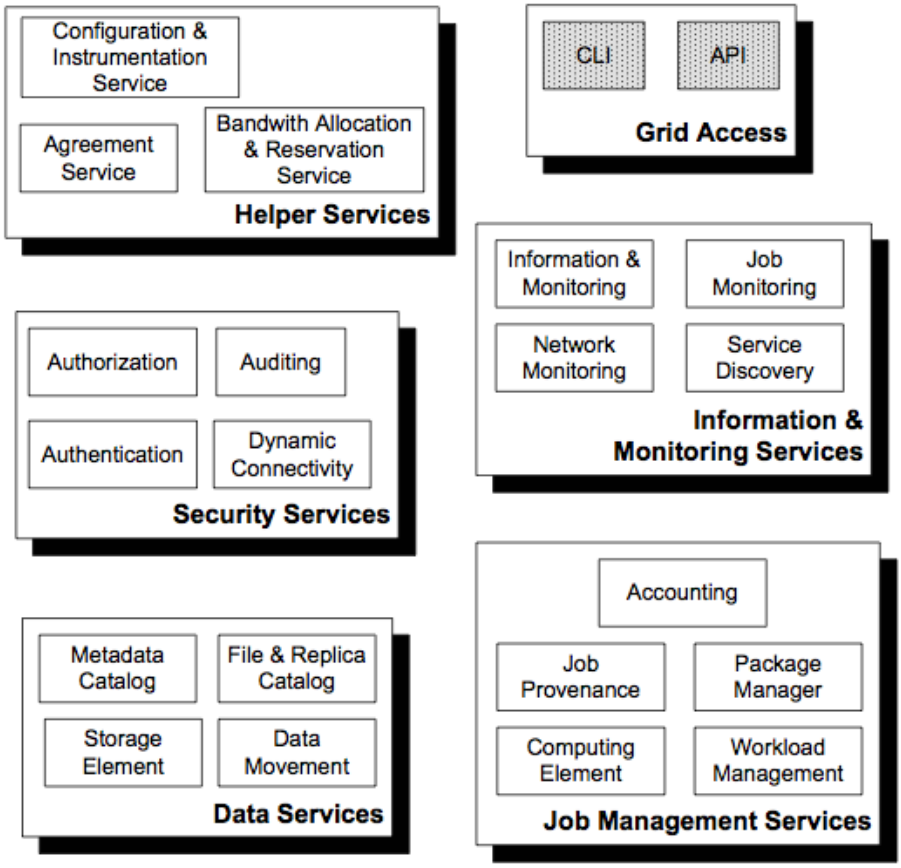



Abbildung 2.6: gLite Services [JDT 05]

## 2.9 Zusammenfassung

In diesem Kapitel wurden die grundlegenden Begriffe für das Verständnis dieser Arbeit eingeführt und erklärt. Des Weiteren wurde die Heterogenität des D-Grids anhand von Beispielen aufgezeigt und festgestellt, dass je nach Community eine andere Grid-Middleware zum Einsatz kommt. Auch die Autorisierungskonzepte unterscheiden sich von Community zu Community. So kommt es vor, dass manche Communities gar keine VOs verwenden, andere wiederum verwenden VOs mit rollenbasierter Autorisierung, wieder andere verwenden VOs, sub-VOs und gruppenbasierte Autorisierung. Die Autorisierungskonzepte unterscheiden sich zum einen wegen den unterschiedlichen Anforderungen in den Communities und zum anderen aufgrund der verwendeten Grid Middleware-Technologie. Im nächsten Kapitel werden die Anforderungen an eine einheitliche Authentifizierungs- und Autorisierungs-Schnittstelle identifiziert.

# 3 Anforderungsanalyse

In diesem Kapitel werden zunächst die Szenarien beschrieben, welche anhand der Aufgabenstellung identifiziert wurden. Anschließend werden aus diesen Szenarien jene Use Cases herausgearbeitet, welche durch den VO-Layer abgedeckt werden sollen. Zusätzlich zu den Use Cases für die Authentifizierung und Autorisierung der Benutzer, werden auch noch einige weitere Use Cases beschrieben, welche für das Management des VO-Layers wichtig sind. Neben den funktionalen Anforderungen, werden in einem weiteren Abschnitt die nichtfunktionalen Anforderungen des VO-Layers beschrieben.

## 3.1 Szenarien

Aus der Aufgabenstellung lassen sich im wesentliche zwei Szenarien herausarbeiten. Das erste Szenario beschäftigt sich mit der Authentifizierung der Benutzer. Das zweite Szenario beschäftigt sich mit der Autorisierung der Benutzer auf die Ressourcen, wobei dieses Szenario auf dem ersten aufbaut, da es ohne erfolgreiche Authentifizierung erst gar nicht zur Autorisierung kommen kann. Da die Szenarien eine vereinheitlichte Authentifizierungs- und Autorisierungs-Infrastruktur (AAI) beschreiben, müssen für die Szenarien einige Anfangsbedingungen gelten:

- Der Benutzer verfügt über einen gültigen Account auf einem Client-System, von welchem aus er auf das Grid zugreifen kann.
- Der Benutzer ist im Besitz eines gültigen Zertifikates für das Grid. Er kann auch mehrere Zertifikate für verschiedene Grid Middleware-Technologien und für verschiedene VOs besitzen.
- Der Benutzer kann Zugriff auf mehrere Client-Systeme haben, die ihrerseits Zugang zu verschiedenen VOs, mit verschiedenen Grid Middleware-Technologien besitzen. Für den Zugang zu den Ressourcen der VOs benötigt der Benutzer auf jedem Client Zugriff auf die entsprechenden Zertifikate, für die von dort aus erreichbaren VOs.
- Der Benutzer ist im gesamten Grid eindeutig durch seinen Benutzernamen und den Hostnamen des verwendeten Client-Rechners identifiziert. Alternativ dazu kann eine eindeutige Identifizierung durch die E-Mail Adresse des Benutzers erfolgen.

Im Folgenden wird, anhand zweier visionärer Szenarien beschrieben, wie das zukünftige System mit dem VO-Layer in Bezug auf Authentifizierung und Autorisierung der Benutzer funktioniert.

### 3.1.1 Szenario 1: Authentifizieren

Das folgende visionäre Szenario beschreibt beispielhaft die Authentifizierung eines fiktiven Benutzers *Max*, der im D-Grid Mitglied in der VO *AstroGrid-D* ist. Der Benutzer möchte sich dort anmelden, um anschließend einen Job auf einem der Hochleistungsrechner zu starten. Als Grid-Middleware kommt in diesem Szenario das Globus Toolkit zum Einsatz.

| Szenario | Authentifizieren |
|---|---|
| Akteure | Max, VO-Layer, Globus Toolkit |
| Ereignisfluss | 1. Max ist Mitglied der VO *AstroGrid-D* und möchte sich bei dieser VO anmelden, um anschließend einen Job auf einem der verfügbaren Hochleistungsrechner zu starten.<br>2. Max authentifiziert sich bei der VO, indem er mit Hilfe des VO-Layers den Authentifizierungsvorgang startet. Dazu übergibt er dem VO-Layer sein Zertifikat und teilt ihm die VO mit, bei der er sich authentifizieren möchte (AstroGrid-D).<br>3. Der VO-Layer nimmt das Zertifikat entgegen und erkennt anhand des Namens, dass es sich um ein Globus Zertifikat handelt. Deshalb wird das Zertifikat an das Globus Toolkit weitergeleitet, damit dieses ein Proxy-Zertifikat erstellen kann.<br>4. Das Globus Toolkit nimmt das Zertifikat entgegen und überprüft es. Wenn die Überprüfung erfolgreich war, wird ein Proxy-Zertifikat generiert, welches in einem Verzeichnis abgelegt wird, sodass der Benutzer darauf zugreifen kann.<br>5. Max erhält eine Bestätigung, dass nun ein Proxy-Zertifikat für ihn verfügbar ist und kann jetzt die Ressourcen der AstroGrid-D VO benutzen. |

Tabelle 3.1: Szenario: *Authentifizieren*

### 3.1.2 Szenario 2: Autorisieren

Dieses visionäre Szenario baut direkt auf dem obigen *authentifizieren* Szenario auf. Der fiktive Benutzer *Max* möchte einen Job auf einem Cluster der Max Planck Gesellschaft in Garching ausführen. Um dies zu erreichen wird - wie beim Globus Toolkit üblich - ein Web-Service aufgerufen, welcher den Job auf dem Zielsystem ausführt.

| | |
|---|---|
| Szenario | Autorisieren |
| Akteure | Max, VO-Layer, Globus Toolkit |
| Ereignisfluss | 1. Max hat sich - wie im ersten Szenario beschrieben - erfolgreich authentifiziert. Nun möchte er auf den Rechencluster der Max Planck Gesellschaft in Garching zugreifen um dort einen Job auszuführen.<br>2. Max startet den Job indem er mit Hilfe des VO-Layers einen Web-Service aufruft, welcher den gewünschten Job ausführen kann.<br>3. Der VO-Layer überprüft anhand der VO-Layer Datenbank, ob sich Max und die Ressource (in diesem Fall der Web-Service) in der gleichen VO befinden. Falls die zutrifft wird zusätzlich überprüft, ob Max den erforderlichen Berechtigungsnachweis für diesen Web-Service besitzt. Nach erfolgreicher Überprüfung durch den VO-Layer wird der Web-Service aufgerufen, ansonsten beendet der VO-Layer an dieser Stelle die Autorisierungsanfrage.<br>4. Das Globus Toolkit nimmt den Aufruf des Web-Service entgegen und überprüft seinerseits erneut, ob Max auf diesen Web-Service zugreifen darf. Nach erfolgreicher Überprüfung wird der Web-Service gestartet und der gewünschte Job auf dem Zielsystem ausgeführt.<br>5. Max erhält eine Bestätigung über den erfolgreichen Start des Jobs und kann - nach dessen Abarbeitung - die Ergebnisse aufrufen. |

Tabelle 3.2: Szenario: *Autorisieren*

## 3.2 Anforderungen (Use Cases)

Da ein Szenario eine Instanz eines Anwendungsfalles (engl. use case) ist, werden in diesem Abschnitt die Use Cases aus den zuvor erstellten Szenarien identifiziert. Die beiden Use Cases *authentifizieren* und *autorisieren*, welche in Abbildung 3.1 als UML Use Case Diagramm abgebildet sind, werden im folgenden Abschnitt genau definiert. Neben diesen zwei zentralen Use Cases, werden noch einige weitere Use Cases beschrieben, welche für das Management des VO-Layers wichtig sind. Diese werden in Abschnitt 3.2.3 eingeführt. In Abbildung 3.2 sind vier ausgewählte Management Use Cases als UML Use Case Diagramm dargestellt.

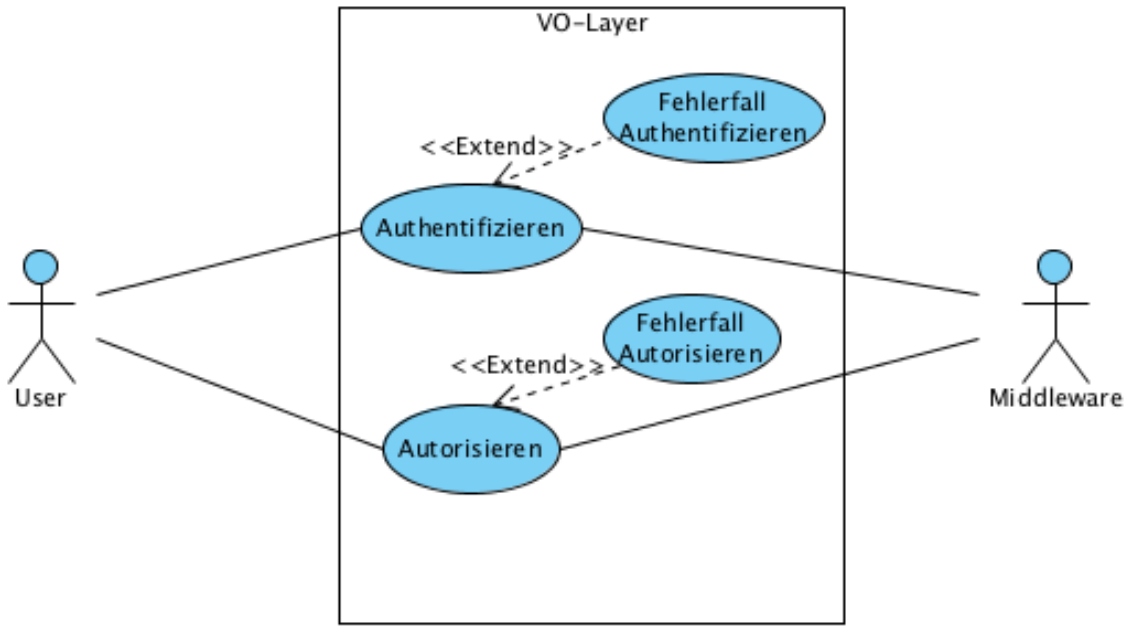



Abbildung 3.1: Dieses UML Use Case Diagramm stellt die Anwendungsfälle *Authentifizieren* und *Autorisieren* dar. Dabei werden die Akteure "User" und "Middleware" über den VO-Layer miteinander verbunden. Die Fehlerfälle sind nicht alle einzeln aufgelistet, sondern über eine *extend* Beziehung ist ein allgemeiner Fehlerfall Use Case eingebunden.

### 3.2.1 Authentifizierung

| Use Case | Authentifizieren |
|---|---|
| Akteure | User, Middleware |
| Ereignisfluss | 1. User sendet Authentifizierungsanfrage mit Benutzername und Benutzerzertifikat an den VO-Layer.<br>2. VO-Layer überprüft die Parameter und initiiert die Erstellung eines Proxy-Zertifikates bei der entsprechenden Middleware.<br>3. Middleware überprüft das Benutzerzertifikat und erstellt ein Proxy-Zertifikat, welches im Dateisystem des Users abgelegt wird.<br>4. VO-Layer teilt dem User die erfolgreiche Erstellung eines Proxy-Zertifikats mit. |
| Anfangsbedingungen | • User ist an einem Client-Rechner angemeldet und kann von dort aus auf die Grid-Dienste zugreifen.<br>• User ist beim VO-Layer und der Middleware registriert. |
| Abschlussbedingungen | • User erhält ein Proxy-Zertifikat ODER<br>• User erhält eine Nachricht, mit der Erklärung dafür, warum die Authentifizierung fehlgeschlagen ist. |

Tabelle 3.3: Use Case: *Authentifizieren*

Bei diesem Use Case gibt es einige Fehlerfälle, die unterschieden werden müssen. Jeder dieser Fehlerfälle zieht eine eigene Meldung nach sich, damit der Benutzer über den Grund des gescheiterten Authentifizierungsvorgangs informiert wird. Diese Fehlerfälle sind:

- Der Benutzername, die VO oder das Zertifikat existiert nicht.
- Der Benutzername passt nicht zur angegebenen VO.
- Der Benutzername befindet sich auf einer schwarzen Liste.
- Das Zertifikat des Benutzers ist nicht gültig.
- Das Zertifikat des Benutzers ist abgelaufen.
- Der Grid-Dienst ist nicht erreichbar.

### 3.2.2 Autorisierung

| Use Case | Autorisieren |
|---|---|
| Akteure | User, Middleware |
| Ereignisfluss | 1. User möchte auf einen Job auf einer Ressource im Grid ausführen und sendet dazu einen Job mit Autorisierungsanfrage und der gewünschten Ressource an den VO-Layer.<br>2. VO-Layer überprüft, ob der User laut VO-Layer Datenbank Zugriff auf die gewünschte Ressource hat und leitet den Job daraufhin an die entsprechende Middleware weiter.<br>3. Die Middleware überprüft anhand des gridmap files oder der UUDB die Zugriffsberechtigung des Users erneut und führt den Job aus.<br>4. User erhält eine Bestätigung über die erfolgreiche Ausführung des Jobs und kann, nach dessen Abarbeitung die Ergebnisse abrufen. |
| Anfangsbedingungen | • Der User hat sich bereits erfolgreich bei der Middleware authentifiziert, das heißt er ist in Besitz eines gültigen Proxy-Zertifikats. |
| Abschlussbedingungen | • Der User erhält den Zugriff auf die Ressource und kann seinen Job starten ODER<br>• Der User erhält eine Nachricht, mit der Erklärung dafür, warum die Autorisierung fehlgeschlagen ist. |

Tabelle 3.4: Use Case: *Autorisieren*

Es müssen auch in diesem Use Case einige Fehlerfälle unterschieden werden, über die der Benutzer im Fehlerfall informiert wird. Diese Fehlerfälle sind:

- Der Benutzer besitzt kein gültiges Proxy-Zertifikat.
- Das Proxy Zertifikat des Benutzers ist abgelaufen.
- Die angeforderte Ressource existiert nicht.
- Der Benutzer und die angeforderte Ressource befinden sich nicht in der selben VO.
- Der Benutzer besitzt nicht die nötigen Rechte, um die Ressource benutzen zu können.
- Die angeforderte Ressource ist nicht verfügbar.

### 3.2.3 Management Anforderungen

In diesem Abschnitt werden jene Use Cases beschrieben, welche für das Management des VO-Layers benötigt werden. Die Datenstruktur des VO-Layers soll so dynamisch wie möglich sein und sich, mit Hilfe der Management Use Cases, beliebig verändern lassen. Die Use Cases in diesem Abschnitt beschreiben insbesondere das Erstellen und Löschen von Benutzern, Ressourcen, VOs, Gruppen, Rollen und Fähigkeiten. Es gibt Use Cases, um Benutzer oder Ressourcen in eine VO oder Gruppe hinzuzufügen oder sie von dort zu entfernen. Des Weiteren können den Benutzern oder Ressourcen beliebige Rollen und Fähigkeiten zugeteilt oder wieder entzogen werden. Die Management Use Cases können in vier Gruppen eingeteilt werden:

- Erstellen und löschen von Benutzern und Ressourcen.
- Erstellen und löschen von VOs, Gruppen, Rollen und Fähigkeiten.
- Hinzufügen von Benutzern oder Ressourcen zu einer VO, Gruppe, Rolle, Fähigkeit.
- Entfernen von Benutzern oder Ressourcen aus einer VO, Gruppe, Rolle, Fähigkeit.

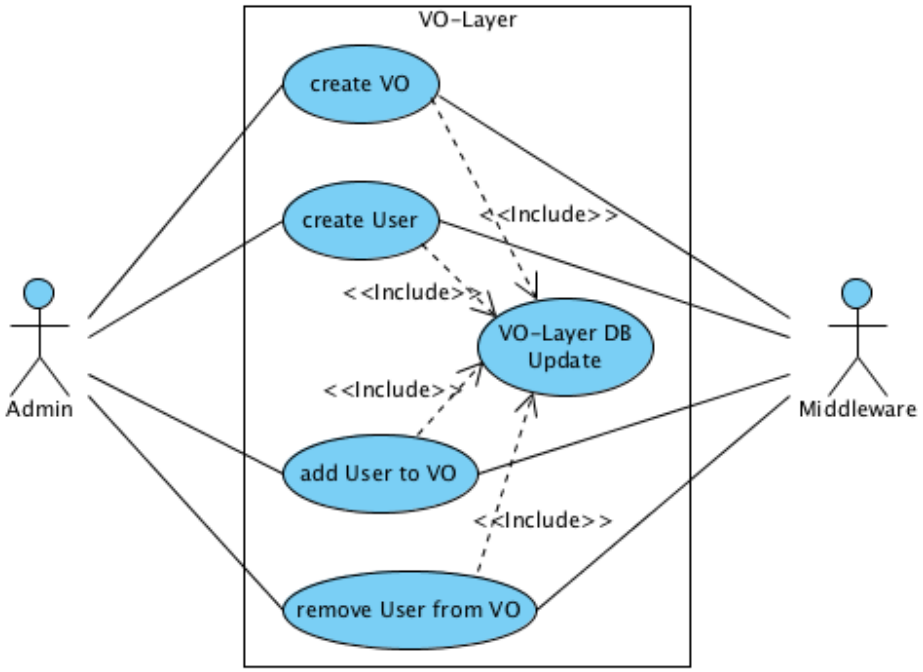



Abbildung 3.2: Ausgewählte Use Cases für das VO-Layer Management.

In Abbildung 3.2 sind vier ausgewählte Management Use Cases dargestellt. Die restlichen Use Cases aus der obigen Auflistung sind den vier in der Abbildung dargestellten sehr ähnlich und sind deshalb im UML Use Case Diagramm ausgespart worden. Im Folgenden werden vier ausgewählte Use Cases genauer beschrieben. Zusätzlich wird immer darauf hingewiesen, welche anderen Use Cases analog dazu sind. Jeder Use Case ruft den *VO-Layer DB Update* Use Case auf, um die Änderungen des Systems in der Datenbank festzuschreiben.

#### Erstellen eines Benutzers

| Use Case | create User |
|---|---|
| Akteure | Admin, Middleware |
| Ereignisfluss | 1. Admin möchte einen neuen Benutzer für das Grid erstellen und sendet den Befehl dafür an den VO-Layer.<br>2. VO-Layer überprüft die Parameter, erstellt den neuen Benutzer und meldet es den entsprechenden Middlewares.<br>3. Die Middlewares tragen ihrerseits den neuen Benutzer ein. |
| Anfangsbedingungen | Der Benutzer ist als Administrator angemeldet. |
| Abschlussbedingungen | Der Admin erhält eine Bestätigung oder eine Fehlermeldung. |

Tabelle 3.5: Use Case: *create User*

Es gibt analoge Use Cases für das Erstellen von Ressourcen und das Löschen von Benutzern oder Ressourcen.

#### Erstellen einer VO

| Use Case | create VO |
|---|---|
| Akteure | Admin, Middleware |
| Ereignisfluss | 1. Admin möchte eine neue VO erstellen und sendet dazu einen Befehl an den VO-Layer.<br>2. VO-Layer überprüft die Parameter, erstellt die neue VO und meldet es den entsprechenden Middlewares.<br>3. Die Middlewares tragen ihrerseits die neuen Autorisierungsinformationen bei sich ein. |
| Anfangsbedingungen | Der Benutzer ist als Administrator angemeldet. |
| Abschlussbedingungen | Der Admin erhält eine Bestätigung oder eine Fehlermeldung. |

Tabelle 3.6: Use Case: *create VO*

Dieser Use Case gilt analog für Erstellen von Gruppen, Rollen und Fähigkeiten sowie für das Löschen der eben genannten Objekte.

#### Füge Benutzer zu VO hinzu

| | |
|---|---|
| Use Case | add User to VO |
| Akteure | Admin, Middleware |
| Ereignisfluss | 1. Admin möchte einen Benutzer zu einer weiteren VO hinzufügen.<br>2. VO-Layer überprüft die Parameter, fügt den Benutzer in der anderen VO ein, speichert dies in der VO-Layer Datenbank und meldet es den entsprechenden Middlewares.<br>3. Die Middlewares fügen ihrerseits die neuen Autorisierungsinformationen bei sich ein. |
| Anfangsbedingungen | Der Benutzer ist als Administrator angemeldet. |
| Abschlussbedingungen | Der Admin erhält eine Bestätigung oder eine Fehlermeldung. |

Tabelle 3.7: Use Case: *add User to VO*

Es gibt analoge Use Cases für das Hinzufügen von Benutzern zu Gruppen, Rollen und Fähigkeiten. Gleiches gilt für das Hinzufügen von Ressourcen.

#### Entferne Benutzer von VO

| | |
|---|---|
| Use Case | remove User from VO |
| Akteure | Admin, Middleware |
| Ereignisfluss | 1. Admin möchte einen Benutzer aus einer VO entfernen.<br>2. VO-Layer überprüft die Parameter, entfernt den Benutzer aus der VO in der VO-Layer DB und meldet es an alle betroffenen Grid Middlewares.<br>3. Die Middleware löscht ihrerseits den Benutzer aus der entsprechenden VO. |
| Anfangsbedingungen | Der Benutzer ist als Administrator angemeldet. |
| Abschlussbedingungen | Der Admin erhält eine Bestätigung oder eine Fehlermeldung. |

Tabelle 3.8: Use Case: *remove user from VO*

Es existieren analoge Use Cases für das Entfernen von Benutzern oder Ressourcen aus, Gruppen, Rollen oder Fähigkeiten.

Bei allen gerade beschriebenen Use Cases werden die Änderungen auch an die Grid Middleware-Technologien weitergegeben und dort gespeichert. Dabei werden die Änderungen, je nach verwendeter Grid-Middleware, in einem gridmap file (Globus Toolkit und gLite) oder einer UUDB (UNICORE) gespeichert.

### 3.2.4 Nicht-Funktionale Anforderungen

Dieser Abschnitt geht auf die nicht-funktionalen Anforderungen nach dem ISO 9126 Standard [ISO9126] ein. Diese hier identifizierten nicht-funktionalen Anforderungen, sollten bei einem produktiven Einsatz des VO-Layers ebenfalls erfüllt sein.

- **Benutzerfreundlichkeit**: Um die Bedienbarkeit zu erleichtern, soll eine einfache Benutzerschnittstelle erstellt werden. Die für Entwickler bereitgestellten Schnittstellen sollen einheitlich und gut dokumentiert sein.
- **Zuverlässigkeit**: Der VO-Layer soll den Benutzer stets über Erfolg oder Misserfolg einer Aktion unterrichten. Die Datenstruktur des VO-Layers soll stets konsistent sein und mit den Daten in den Middleware-Technologien übereinstimmen.
- **Sicherheit**: Der VO-Layer darf nur von autorisierten Benutzern verwaltet werden.
- **Leistung**: Die Antwortzeit des VO-Layers soll so kurz wie möglich sein. Der Durchsatz an Anfragen soll ausreichen, damit auch viele Benutzer gleichzeitig arbeiten können.
- **Skalierbarkeit**: Das System muss gut skalieren, da es auch mit vielen Benutzern und großen Datenmengen umgehen können muss.
- **Portabilität**: Der VO-Layer soll so implementiert sein, dass er auf unterschiedlicher Hardware und mit verschiedenen Betriebssystemen funktioniert.

## 3.3 Zusammenfassung

Dieses Kapitel hat zunächst die zwei zentralen Szenarien *authentifizieren* und *autorisieren* eingeführt. Aus diesen beiden Szenarien wurden anschließend die entsprechenden Use Cases abgeleitet. Diese Use Cases beschreiben das Zusammenspiel von Benutzer, VO-Layer und Grid-Middleware. Beim Erstellen dieser Use Cases wurde größter Wert auf eine allgemein gültige Struktur gelegt, damit ein einheitliches AAI geschaffen werden kann, welches auf allen im D-Grid verwendeten Grid Middleware-Technologien aufsetzt. Zusätzlich zu den Use Cases *authentifizieren* und *autorisieren*, wurden auch noch eine Reihe von Management Use Cases eingeführt, welche die Verwaltung des VO-Layers beschreiben. Diese Use Cases haben die Eigenschaft, dass die Änderungen, die sie hervorrufen, einerseits in der VO-Layer Datenbank gespeichert werden müssen, andererseits auch in den betroffenen Grid Middlewares festgeschrieben werden müssen.

Im folgenden Kapitel wird überprüft, ob diese Anforderungen auch mit bereits bestehenden Techniken erfüllt werden, oder ob eine Eigenentwicklung nötig ist, um die Anforderungen vollständig zu erfüllen.

# 4 State-of-the-Art

In diesem Kapitel werden die Authentifizierungs- und Autorisierungsverfahren, welche im D-Grid zum Einsatz kommen, genauer untersucht. Für eine grundlegende Einführung in Authentifizierung und Autorisierung, sei auf die Kapitel 2.3 und 2.4 verwiesen. Mit der Untersuchung und der Analyse von Authentifizierungs- und Autorisierungs-Infrastrukturen in Grid-Middleware, hat sich bereits das D-Grid Integrationsprojekt (DGI) eingehend beschäftigt. Aus diesem Grund werden einige der Ergebnisse aus der Arbeit [DEF+ 06a] hier aufgegriffen und anschließend, hinsichtlich der in Kapitel 3 identifizierten Anforderungen, untersucht.

## 4.1 Authentifizierung im D-Grid

Die Authentifizierung im D-Grid erfolgt über eine Publik-Key-Infrastruktur (PKI), wobei die verwendeten Zertifikate auf dem X.509 Standard beruhen. Es gibt innerhalb des D-Grids einige Zertifizierungsstellen, welche als Trusted-Third-Party (TTP) fungieren. In den Zertifikaten steht unter anderem der so genannte *Distinguished Name* (DN), welcher einen Grid Benutzer weltweit eindeutig identifiziert. Der DN setzt sich dabei aus folgenden Teilen zusammen:

1. Country (C)
2. Organisation (O)
3. Organisational Unit (OU)
4. Common Name (CN)

Ein Beispiel für einen DN wäre:

C=DE/O=GridGermany/OU=TU München/CN=Wolfgang Kirchler

Die Zertifikate dienen lediglich der Authentifizierung der Benutzer und nicht der Autorisierung. Die Autorisierung wird über ACLs oder VOs gesteuert. Durch die Trennung der beiden Bereiche Autorisierung und Authentifizierung erreicht man eine Reduktion der Komplexität und somit ein höheres Maß an Sicherheit. Eine ausführliche Beschreibung des Authentifizierungsvorgangs im D-Grid gibt [EpPa 05].

## 4.2 Autorisierung im D-Grid

Die Art des Autorisierungsmechanismus im D-Grid hängt wesentlich von der verwendeten Grid Middleware-Technologie ab. Die drei Grid-Middlewares Globus Toolkit, UNICORE und gLite implementieren die Autorisierung auf unterschiedliche Weise. Im Folgenden wird auf die Unterschiede in den Autorisierungsmechanismen der einzelnen Grid Middleware-Technologien eingegangen. Da für eine effiziente und flexible Autorisierungs-Infrastruktur ein VO-Konzept wünschenswert ist, werden die Grid-Middlewares auch hinsichtlich ihrer VO-Unterstützung untersucht. In der Arbeit [DEF+ 06b], welche im Rahmen des DGI Projektes erstellt wurde, werden die unterschiedlichen Autorisierungsmethoden, der im D-Grid eingesetzten Middleware-Technologien, für ausgewählte Use Cases untersucht. Ein Zusammenfassung der Autorisierungsmethoden im D-Grid und ein Ausblick auf zukünftige Entwicklungen ist in [Zieg 08] zu finden.

### 4.2.1 Autorisierung in Globus

Die standardmäßige Autorisierung erfolgt in Globus über ein so genanntes *gridmap file*. Das gridmap file ist im Grunde eine ACL, bei welcher in einer Datei für jede Zugriffsberechtigung eine Zeile eingetragen wird. Jede Zeile besteht aus aus den Spalten *Distinguished Name* (DN) und *User Account*. Wenn nun ein Benutzer Zugriff auf eine Ressource haben möchte, wird sein DN (aus dem Zertifikat) mit den Einträgen im gridmap file verglichen. Wenn dort der gleiche DN auftaucht, wird dem Benutzer der User Account aus dem gridmap file zugeteilt. Mit diesem Account hat der Benutzer Zugriff auf die gewünschte Ressource.

GT4 besitzt in der Standardversion kein VO-Konzept, allerdings gibt es Erweiterungen mit denen VOs etabliert werden können und somit rollenbasierte Autorisierung ermöglicht wird. Die wichtigste Erweiterung zu Globus, um VOs zu unterstützen ist der *Virtual Organisation Membership Service* (VOMS). Für eine Beschreibung und Analyse von VOMS, im Rahmen der hier gestellten Anforderungen, sei auf das Kapitel 4.4 verwiesen.

Das Globus Toolkit kann auch um eine attributbasierte Autorisierung erweitert werden. Diese Erweiterung basiert auf dem Shibboleth-Konzept, welches in Kapitel 4.5 eingeführt und analysiert wird. Um GT4 um Shibboleth-Mechanismen zu erweitern, wurde das Projekt *GridShib*[1] initiiert [WBKS 05].

Die Abbildung 4.1 zeigt die AAI von Globus mit den verschiednen Autorisierungskonzepten. Zum Teil sind die Konzepte auf Web-Services aufgebaut (WS), zum Teil noch nicht (pre-WS). Für die genaue Erklärung der abgebildeten aber hier nicht vorgestellten Konzepte, sei auf [DEF+ 06a] verwiesen.

[1] http://gridshib.globus.org/

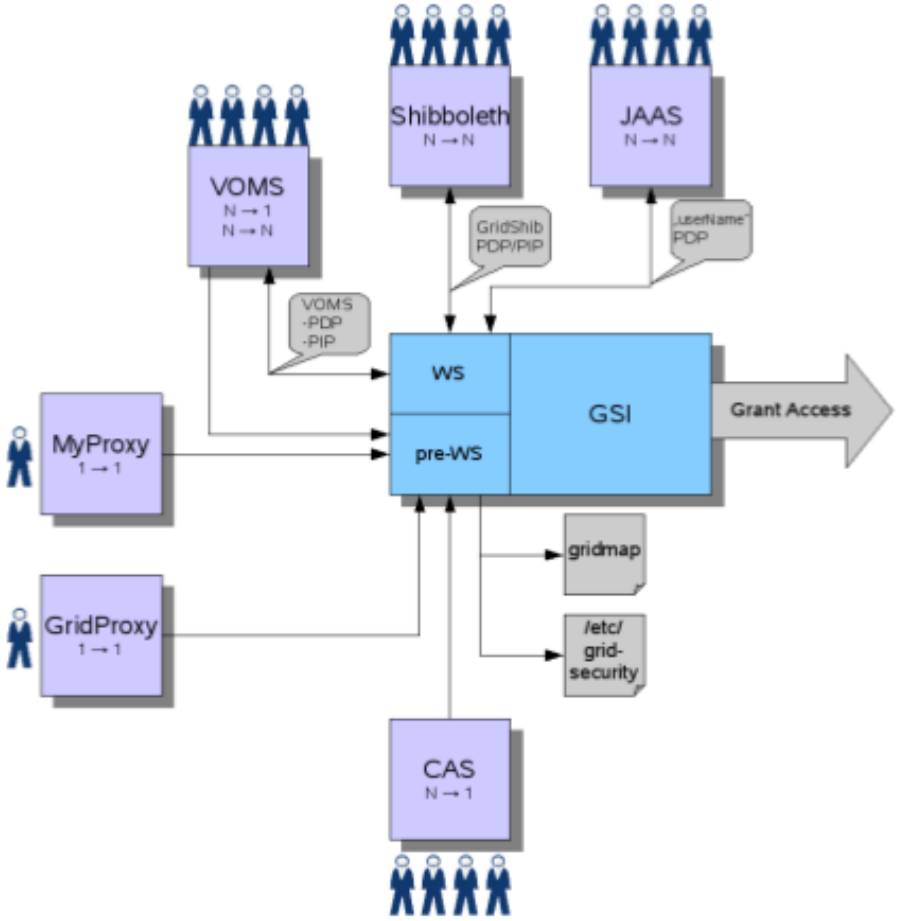



Abbildung 4.1: AAI in Globus Toolkit 4 [DEF+ 06a]

### 4.2.2 Autorisierung in UNICORE

In UNICORE wird die so genannte UNICORE User Database (UUDB) zur Autorisierung verwendet. In dieser Datenbank werden alle Informationen zu den Benutzern der verwalteten USite gespeichert, insbesondere deren Zertifikate. Nach der Authentifizierung bildet UNICORE die Zertifikate auf lokale Accounts ab. Somit werden anhand von unterschiedlichen Zertifikaten, unterschiedliche Zugangsberechtigungen vergeben, da jedes Zertifikat auf einen anderen Account abgebildet werden kann. Der Nachteil dabei ist, dass der Aufwand für die Pflege der UUDB bei größeren Benutzerzahlen stark ansteigt. Abbildung 4.2 zeigt die grundlegende AAI von UNICORE.

Genau wie GT4 besitzt auch UNICORE kein VO-Konzept, allerdings wird auch hier an der Integration von attribut- und rollenbasierter Autorisierung gearbeitet. Alle diese Arbeiten finden im Rahmen des IVOM Projektes im D-Grid statt. Eine Einführung in IVOM gibt das Kapitel 4.3.2.

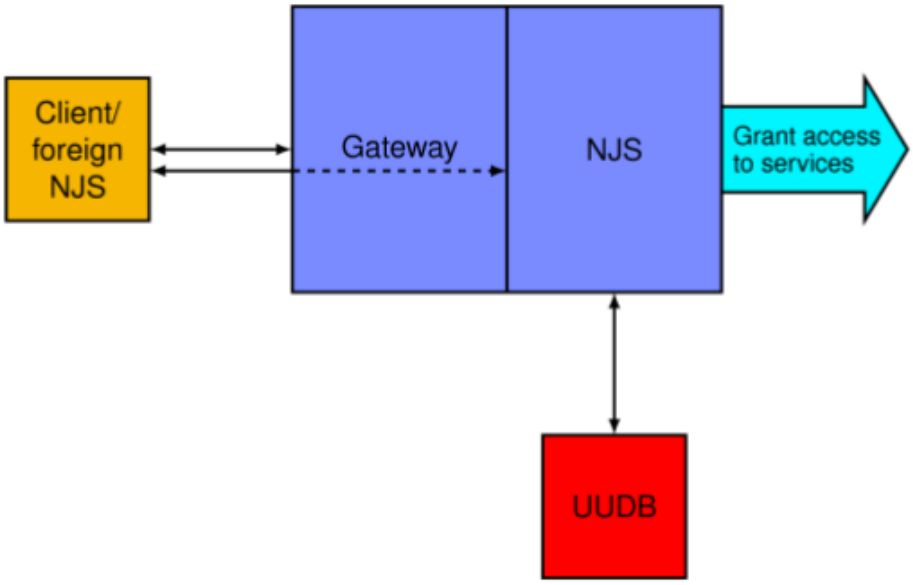



Abbildung 4.2: AAI in UNICORE 6 [DEF+ 06a]

### 4.2.3 Autorisierung in gLite

Von allen hier betrachteten Grid Middleware-Technologien ist gLite die einzige, die ein VO-Konzept integriert. gLite setzt dabei auf VOMS und unterstützt somit rollenbasierte Autorisierung. Ähnlich wie Globus verwendet auch gLite ein gridmap file, welches von einigen Diensten als Autorisierungsmechanismus verwendet wird. Die Einträge im gridmap file werden durch regelmäßige Anfragen an den VOMS-Server auf den neuesten Stand gebracht. Somit ist VOMS das zentrale Autorisierungsinstrument in gLite und bietet dem Benutzer rollenbasierte Autorisierung.

Abbildung 4.3 zeigt die AAI von gLite. Dabei kann man sehen, dass einige der Autorisierungsdienste bereits Web-Services (WS) unterstützen, andere wiederum nicht (pre-WS). In [DEF+ 06a] werden die hier abgebildeten aber nicht vorgestellten Konzepte genauer erklärt.

## 4.3 VO-Management im D-Grid

Das VO-Management im D-Grid ist sehr uneinheitlich, und wird zur Zeit von jeder Community separat verwaltet. Diese Heterogenität ist historisch bedingt und darin begründet, dass die verschiedenen Communities oft ein unterschiedliches VO-Verständnis besitzen. So gibt es beispielsweise Communities, die überhaupt kein VO-Konzept unterstützen, andere wiederum unterstützen VOs und wieder andere verwenden VOs mit gruppen- und rollenbasierter Autorisierung. Das unterschiedliche VO-Verständnis der Communities liegt zum Teil auch an der verwendeten Grid-Middleware, da nicht alle eingesetzten Middleware-Technologien VOs unterstützen. So bietet nur gLite von Grund auf eine Unterstützung für VOs, Globus

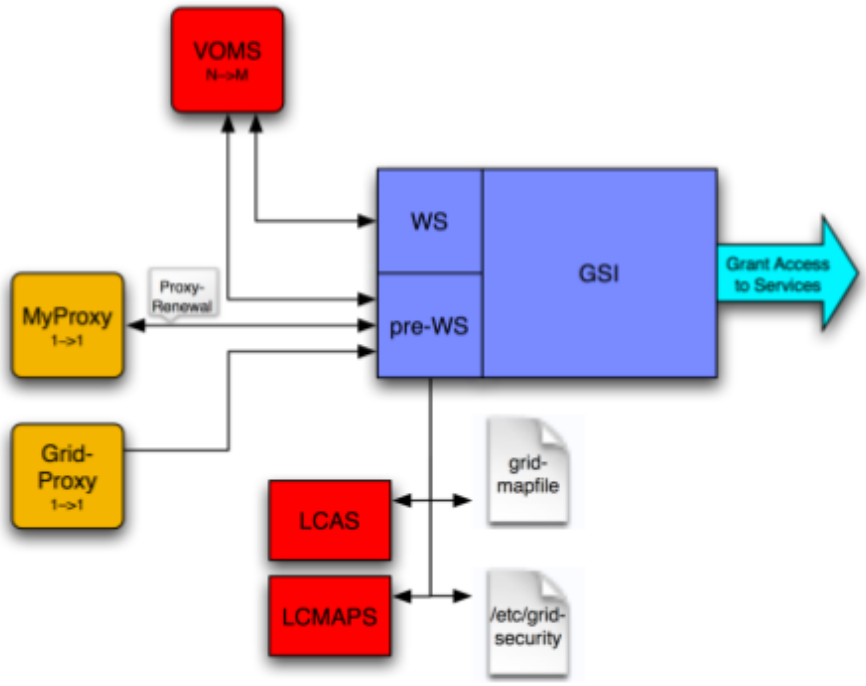



Abbildung 4.3: AAI in gLite 3.1 [DEF$^+$ 06a]

Toolkit kann nur mit Erweiterungen VOs unterstützen. UNICORE unterstützt keine VOs und es existiert zur Zeit auch keine Erweiterung dafür.

### 4.3.1 Das DGI Projekt

Die Aufgabe des D-Grid Integrationsprojektes (DGI) ist es, eine nachhaltige Grid-Plattform für e-Science in Deutschland zu etablieren. Das DGI stellt dafür Ressourcen bereit, die für die Etablierung einer deutschen Grid-Infrastruktur notwendig sind. Dazu werden unter anderem Entwicklungen aus den Community-Projekten als einheitliche Grid-Dienste für die gesamte D-Grid Community integriert.

In einem Fachgebiet hat sich das DGI bereits mit dem VO-Management im D-Grid auseinander gesetzt und ein Thesenpapier dazu ausgearbeitet [VOM 06]. Im Rahmen der beiden Projekte IVOM und VO-Management wurden die D-Grid Communities über ihre Anforderungen an das VO-Management befragt (siehe dazu [GHPS 07]). Daraus wurde ein Rahmenkonzept für das VO-Management im D-Grid erstellt [MSZ 08]. Das im folgenden Abschnitt beschriebene IVOM Projekt soll den D-Grid Communities eine Software zur Verfügung stellen, welche dieses Rahmenkonzept implementiert.

### 4.3.2 Das IVOM Projekt

Das D-Grid Projekt IVOM beschäftigt sich mit der Interoperabilität und Integration der VO-Management Technologien im D-Grid. Dieses Projekt hat mehrere Ziele, welche auf ein verbessertes und einheitliches VO-Management und AAI abzielen. Einen kurzen Überblick über die Projektziele und durchzuführenden Arbeiten liefert [ZiGr 06]. Im Folgenden sind die vorrangigen Ziele des IVOM Projektes aufgelistet (Quelle: IVOM Webseite[2]):

- Die Implementierung eines übergreifenden VO-Managements, einschließlich der Integration in Globus Toolkit 4, gLite und UNICORE, ermöglicht eine einheitlichen Nutzer- und Rechteverwaltung zur Authentifizierung und Autorisierung in den Communities, unabhängig von der jeweils verwendeten Grid-Middleware.
- Die Authentifizierung der Nutzer in "shibbolisierten" Grid-Umgebungen wird unmittelbar gegenüber deren Heimateinrichtung erfolgen und ist somit eng an die dort zugrunde liegenden organisatorischen Verfahren für Eintritt und Austritt gekoppelt. Die Heimateinrichtung wird dabei festlegen können, mit welchen technischen Verfahren ihre Nutzer sich authentifizieren. Bei Verwendung alternativer Verfahren werden die bisher verwendeten PKI-basierten Ansätze entlastet.
- Communities mit bestehenden Nutzerdatenbanken werden durch die Einführung von Shibboleth in die Lage versetzt, einer großen Zahl von Anwendern den Zugang zu Grid-Diensten zu ermöglichen. Der bisher notwendige Aufwand für ein Massen-Rollout von Nutzer-Zertifikaten entfällt.
- Der Verwaltungsaufwand für die Verlängerung oder den gegebenenfalls notwendigen Rückruf von Nutzerzertifikaten entfällt. Für den Nutzer entfällt die regelmäßige Erneuerung des Zertifikats.

In der ersten Phase des Projektes wurden vorhandene Lösungen wie VOMS und Shibboleth untersucht und durch Interviews die Anforderungen der Communities bestimmt. Damit konnte anschließend ein Konzept für das VO-Management im D-Grid entwickelt werden. Die Ergebnisse der Analyse sind in [DEF$^+$ 06a], die Ergebnisse der Interviews mit den Communities in [PGGH 07] nachzulesen. Mit diesen Daten wurde im Anschluss ein AAI-Prototyp für die D-Grid Infrastruktur entworfen und in der Arbeit [PGG$^+$ 07] festgeschrieben.

Das wichtigste Ziel dieses Projektes ist es, ein Rahmenkonzept für die Bildung virtueller Organisationen zu erstellen. Dazu gehört auch die Bereitstellung von Hilfsmitteln für das Management und die Umsetzung in den D-Grid Communities. Dazu wurde vor kurzem ein Integrations- und Deployment-Leitfaden ausgearbeitet, der beschreibt wie die vorgeschlagene Lösung in die existierenden Installationen integriert werden kann und wie die Dienste in der heterogenen Middleware-Umgebung des D-Grids ausgebracht werden können [GGG$^+$ 08].

Auf den ersten Blick sind die Ziele des IVOM Projektes und dieser Diplomarbeit sehr ähnlich. Die genauen Unterschiede erschließen sich erst bei näherer Betrachtung der Anforderungen aus Kapitel 3. Im Folgenden werden die wichtigsten Unterschiede zwischen dem IVOM Projekt und dieser Diplomarbeit aufgezeigt:

[2] http://www.d-grid.de/index.php?id=314

- Das IVOM Projekt hat das Ziel, das VO-Management im D-Grid zu vereinheitlichen. Diese Arbeit hingegen hat das Ziel, eine einheitliche AAI für die Middleware-Technologien im D-Grid zu entwickeln.
- Das IVOM Projekt erweitert die im D-Grid verwendeten Middleware-Technologien um bereits vorhandene Konzepte (wie VOMS oder Shibboleth). In dieser Arbeit hingegen wird eine neue Schnittstelle mit einer generischen VO-Struktur entwickelt, die als AAI dient und die auf vorhandene Dienste der Middleware-Technologien zugreift, diese also nicht um neue Funktionen erweitert.
- Das IVOM VO-Konzept unterscheidet sich von dem dieser Arbeit dadurch, dass das hier beschriebene VO-Konzept auch die Aufnahme von Ressourcen in VOs unterstützt und dadurch die Autorisierung vereinfacht. Somit müssen keine zusätzlichen ACLs gespeichert werden.

## 4.4 VOMS

Der *Virtual Organisation Membership Service* kurz VOMS ermöglicht das Erstellen von Benutzer-Autorisierungsinformationen einer VO. Mit VOMS werden allerdings nur die grundlegenden Informationen in einer Datenbank gespeichert, die die Beziehung eines Benutzers zu seiner VO wiedergeben. Dazu gehören die Gruppen und Rollen denen der Benutzer angehört, Informationen zu den Zertifikaten. Das VOMS System besteht aus folgenden Teilen:

**User Server** Beantwortet Autorisierungsanfragen der Clients und liefert die gewünschten Informationen über die Zugriffsberechtigungen des Benutzers, in Form eines Zertifikats, zurück.

**User Client** Baut mit Hilfe eines Benutzerzertifikats eine Verbindung zum Server auf und erhält von diesem eine Liste mit den Gruppen und Rollen des Benutzers.

**Administrator Server** Beantwortet Autorisierungsanfragen des Clients und propagiert Änderungen in der VOMS Datenbank.

**Administrator Client** Wird von den VO-Administratoren verwendet um VOs zu verwalten (beispielsweise Benutzer hinzufügen, Gruppe erstellen, Rolle ändern u.s.w.).

Wenn ein Benutzer VOMS kontaktiert, muss er den Befehl `voms-proxy-init` aufrufen. Durch diesen Aufruf wird eine Anfrage an den VOMS Server gestellt und der Benutzer erhält durch Vorzeigen seines Benutzerzertifikates ein Proxy-Zertifikat, welches Informationen zu den Gruppen und Rollen des Benutzer beinhaltet. Das Proxy-Zertifikat enthält des Weiteren den Berechtigungsnachweis des Benutzers und des VOMS Servers. Zudem ist das Proxy-Zertifikat nur eine bestimmte Zeit lang gültig und ist vom VOMS Server signiert. Abbildung 4.4 zeigt die Architektur von VOMS, welche im Wesentlichen aus einem Server mit Datenbankanbindung und mehreren Clients besteht. Für ausführlichere Informationen zum VOMS System sei auf [ACC$^+$ 03] verwiesen. Eine ausführliche Beschreibung für Benutzer und Administratoren findet man in [Cias 06] und [Lore 05].

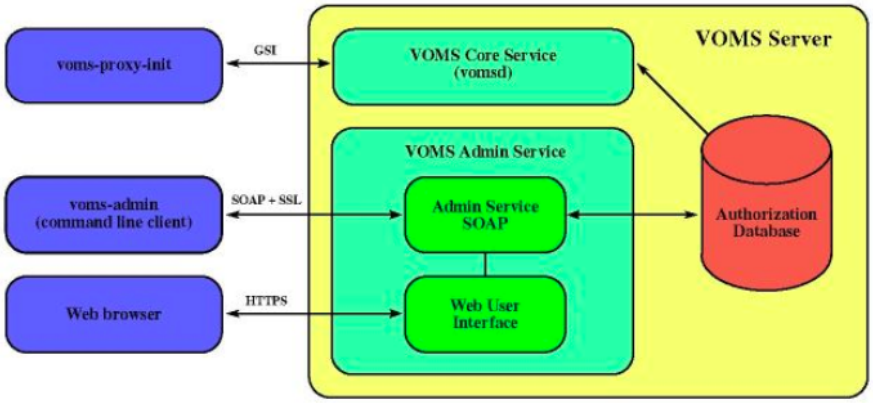



Abbildung 4.4: VOMS Architektur [Lore 05]

### 4.4.1 VOMRS

VOMS hat einige Schwachstellen, insbesondere in Hinblick auf das Management von VOs. So muss ein Benutzer, welcher er sich nicht bei VOMS registrieren konnte, dies dem VO-Manager mitteilen, sodass dieser den Benutzer manuell registriert. Diese, im ersten Augenblick kleine Unannehmlichkeit, kann bei kleinen VOs noch hingenommen werden, bei großen VOs mit sehr vielen Benutzern wird daraus ein administratives Problem, welches nur durch hohen Zeit- und Personalaufwand in den Griff zu bekommen ist. Aus diesem Grund wurde beschlossen, VOMS zu erweitern und verbessern. Dieses Projekt trägt den Namen *VOMS eXtension* kurz VOX, und wurde von Fermilab[3] in Zusammenarbeit mit dem U.S. CMS[4] initiiert. Eine Komponenten dieser Erweiterung ist der so genannte *Virtual Organization Membership Registration Service* (VOMRS), der eine vereinfachten Benutzerverwaltung unterstützt und zusätzlich noch weitere Informationen wie Benutzerzertifikate, Benutzerprofil, abgeschlossene Verträge etc. in der VOMRS Datenbank speichert. Eine ausführliche Dokumentation zu VOX und VOMRS findet man in [VOX]. Die VOMRS Architektur ist in Abbildung 4.5 dargestellt. Dort kann man gut erkennen, dass das System im Vergleich zu VOMS stark erweitert wurde.

### 4.4.2 VOMS in Grid Middleware

VOMS/VOMRS ist eine gute Möglichkeit, VOs in Grids zu etablieren und zu verwalten. Für das Globus Toolkit existiert bereits eine Erweiterung, womit auch Globus VOs erstellen kann. In gLite ist VOMS bereits von Grund auf integriert und es besteht die Möglichkeit, VOMRS als Erweiterung zu installieren. Für UNICORE ist die Integration noch nicht sehr weit fortgeschritten, allerdings kümmert sich das IVOM Projekt darum. Die aktuellen Er-

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

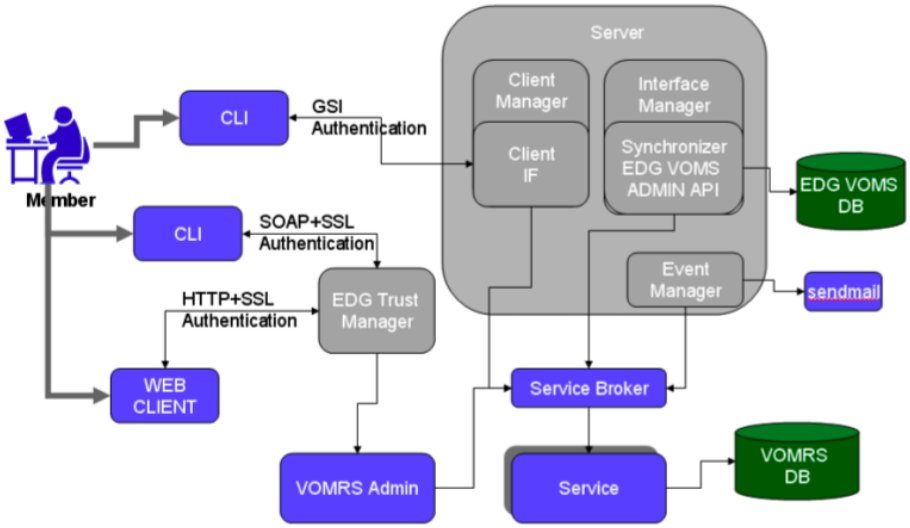



Abbildung 4.5: VOMRS Architektur [VOX]

gebnisse der VOMS Integration in UNICORE können unter [FaZi a] nachgelesen werden.

Die Anforderungen an ein einheitliches AAI, aus Kapitel 3 kann auch VOMS/VOMRS nicht erfüllen. Zum einen scheitert es daran, dass VOMS keine Ressourcen in den VOs unterstützt und zum anderen muss VOMS in jede Grid-Middleware separat integriert werden, weswegen mit VOMS kein übergeordnetes und einheitliches AAI erstellt werden kann.

## 4.5 Shibboleth

Shibboleth ist ein vom Internet2/MACE[5] entwickeltes Verfahren zur verteilten Authentifizierung und Autorisierung für Web-Services [Cant 05]. Shibboleth basiert auf der OASIS *Security Assertion Markup Language* Version 1.1 (SAML) [JH 04] und bietet dem Benutzer eine attributbasierte Zugriffskontrolle sowie Single-Sign-On bei der Authentifizierung. Shibboleth wurde in erster Linie nicht für die Anwendung in Grids entwickelt, wird aber als mögliche Ergänzung beziehungsweise als Ersatz für PKI-Strukturen in Grids angesehen. Die Architektur von Shibboleth besteht aus drei verschiedenen Teilen:

**Identity Provider (IdP)** Der IdP ist die Heimateinrichtung des Benutzers. Jede Einrichtung, die Mitglied einer Föderation wird, benötigt einen IdP. Dort sind alle Informationen des Benutzers gespeichert, insbesondere seine Attribute mit denen er berechtigt ist auf die verschiedenen Ressourcen zuzugreifen.

**Service Provider (SP)** Stellt seine Ressourcen den Benutzern zur Verfügung. Jede Einrichtung, die eine Ressource in einer Föderation bereitstellt, benötigt einen SP. Der Zugriff auf die Ressourcen wird nur gestattet, wenn der Benutzer die entsprechenden Attribute

[5] http://middleware.internet2.edu/MACE/

besitzt. Die Attribute werden vom IdP in Form einer SAML-Assertion an den SP übertragen.

**WAYF-Service** Der *Where-Are-You-From*-Service ist die zentrale Stelle einer Föderation, die alle teilnehmenden IdPs und deren Adressen kennt. Dieser Dienst ist optional und wird dazu verwendet, den IdP des Benutzers zu lokalisieren. Ein Benutzer kann zu mehreren IdP gehören, da er verschiedenen Rollen inne haben kann.

Abbildung 4.6 stellt einen Überblick über die Transaktionen in Shibboleth, beim Zugriff auf eine Ressource dar. Dabei werden folgende Schritte ausgeführt [DEF+ 06a]:

**1)** Ein Nutzer möchte Zugriff auf eine durch Shibboleth geschützte Ressource eines Service Providers (SP) erhalten.

**2)** Der Nutzer wird zum WAYF-Service umgeleitet.

**3), 4)** Der Nutzer wählt seine Heimat-Institution (IdP) aus.

**5)** Der Nutzer wird zum Handle-Service (HS) seines IdP weitergeleitet.

**6), 7)** Der Nutzer authentifiziert sich auf vertrautem Wege gegenüber seinem IdP.

**8)** Der Handle Service (HS) generiert eine eindeutige ID (Handle) und leitet den Nutzer zurück zum Assertion-Consumer-Service (ACS) des SP. Der ACS überprüft die mitgelieferte Assertion, generiert eine Session und übergibt an den Attribute-Requester (AR).

**9), 10)** Der AR nutzt das Handle, um Attribute des Nutzers bei der Attribute-Authority (AA) des IdP abzufragen. Die AA liefert, unter Berücksichtigung der Attribute-Release-Policy (ARP), eine Attribute Assertion an den AR zurück. Der SP entscheidet, anhand der erhaltenen Attribute, über die Gewährung und Art des Zugangs.

Shibboleth unterstützt in der Grundversion keine VOs, sondern ist lediglich ein attributbasierter Autorisierungsdienst. Allerdings existiert für Shibboleth die Erweiterung *myVocs*, mit welcher der Shibboleth Autorisierungsdienst um ein VO-Konzept erweitert werden kann. Näheres zur myVocs ist in [GRSW 06] zu finden.

### 4.5.1 Shibboleth in Grid Middleware

Zur Zeit existieren mehrere Vorschläge und Projekte, wie Shibboleth Mechanismen in Grid-Umgebungen integriert werden können. Das am weitesten fortgeschrittene Projekt ist dabei **GridShib**[6], dessen Ziel es ist, das Globus Toolkit und Shibboleth zu vereinen. GridShib ist ein Projekt des National Center for Supercomputer Applications (NCSA[7]) und der University of Chicago[8]. Mit GridShib sollen VOs durch verteilte Nutzerattribute erstellt werden können. Die Service Provider müssen dadurch nicht mehr die Identität des Benutzers prüfen, sondern sie gestatten den Zugriff aufgrund der Attribute des Benutzers. Ein erster Ansatz zur Integration von Shibboleth Mechanismen in Globus-Toolkit ist in [WBKS 05] beschrie-

[6] http://gridshib.globus.org/
[7] http://www.ncsa.uiuc.edu/
[8] http://www.uchicago.edu/

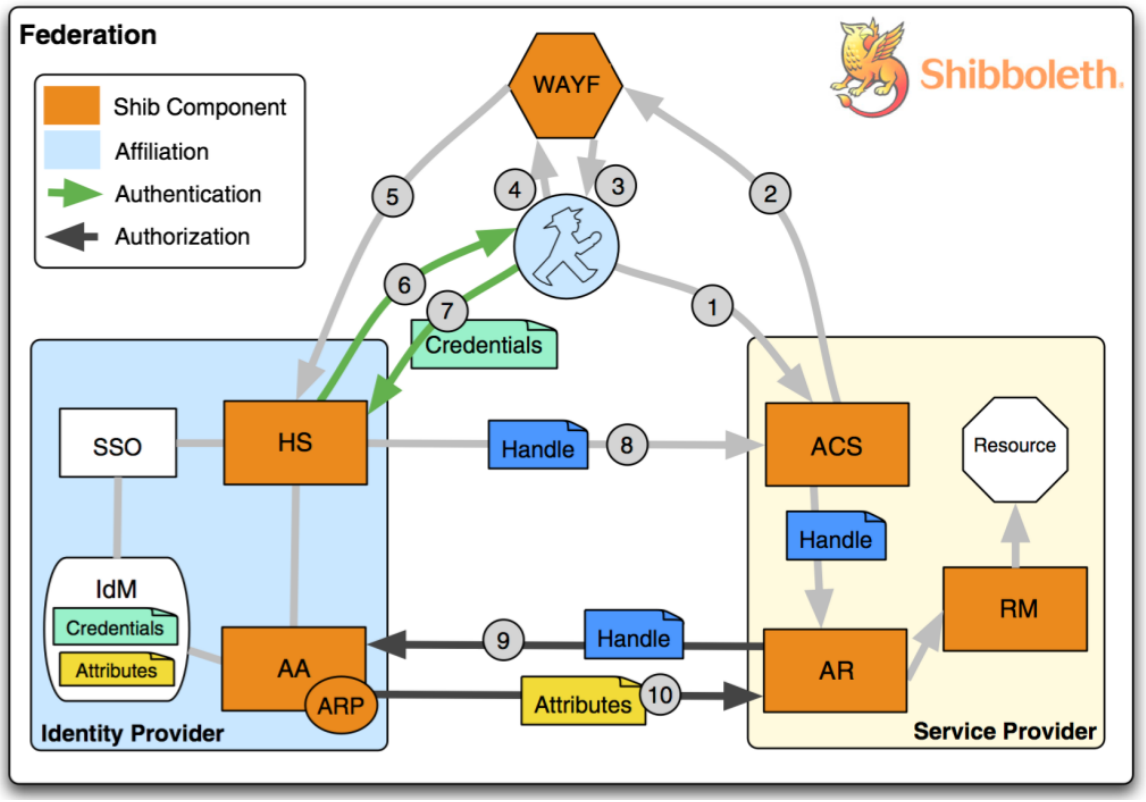



Abbildung 4.6: Shibboleth Architektur [DEF+ 06a]

ben. Weitere Publikationen zu Thema GridShib sind auf der GridShib Webseite[9] zu finden. Eine ausführliche Analyse von Shibboleth beziehungsweise GridShib auf seine Tauglichkeit als Autorisierungsdienst in einer Grid-Umgebung ist in der Arbeit [Mari 06] zu finden.

Die Integration von Shibboleth in anderen Middleware-Technologien ist noch nicht so weit fortgeschritten wie beim Globus Toolkit, allerdings beschäftigt sich im D-Grid das IVOM Projekt mit der Integration von Shibboleth in UNICORE. Die Ergebnisse können in [FaZi b] nachgelesen werden. Die Integration von Shibboleth Mechanismen in gLite wird vom Betreiber des Schweizer Wissenschaftsnetzes, SWITCH[10] vorangetrieben [Witz 08].

Die in Kapitel 3 gestellten Anforderungen an ein AAI werden auch von Shibboleth und seinen Middleware-Adaptionen nicht erfüllt. Zunächst unterstützt Shibboleth in der Grundversion gar keine VOs, sondern ist lediglich ein attributbasierter Autorisierungsdienst. Auch mit der Erweiterung myVocs bleiben die selben Nachteile wie bei VOMS. Zum einen können keine VOs mit Ressourcen etabliert werden, zum anderen kann Shibboleth nicht als übergeordnete AAI verwendet werden, sondern muss für jede Middleware separat angepasst werden.

[9] http://gridshib.globus.org/documents.php
[10] http://www.switch.ch

## 4.6 Das D-Grid Betriebskonzept

Die Zentrale Registrierung von Benutzern, VOs und Ressourcen erfolgt im D-Grid mit Hilfe der Dienste GRRS (Grid Resource Registry Service) und VOMRS. VOMRS wurde bereits in Kapitel 4.4 behandelt und wird hier nur kurz auf die Relevanz in Bezug auf das D-Grid hin untersucht. Eine ausführliche Beschreibung des D-Grid Betriebskonzeptes liefert [BDE+ 07].

### 4.6.1 VOMRS

Im D-Grid werden VOs und Benutzer mit Hilfe von VOMRS, entweder vom DGI oder von den Communities selbst, verwaltet. Für jede VO werden aus der zugehörigen Community ein oder mehrere Verantwortliche ausgewählt, die mit Hilfe dieses zentralen Dienstes neue Benutzer als VO-Mitglieder aufnehmen, ablehnen, sperren oder entfernen können. Abbildung 4.7 zeigt die Architektur dieses zentralen VO- und Benutzerdienstes.

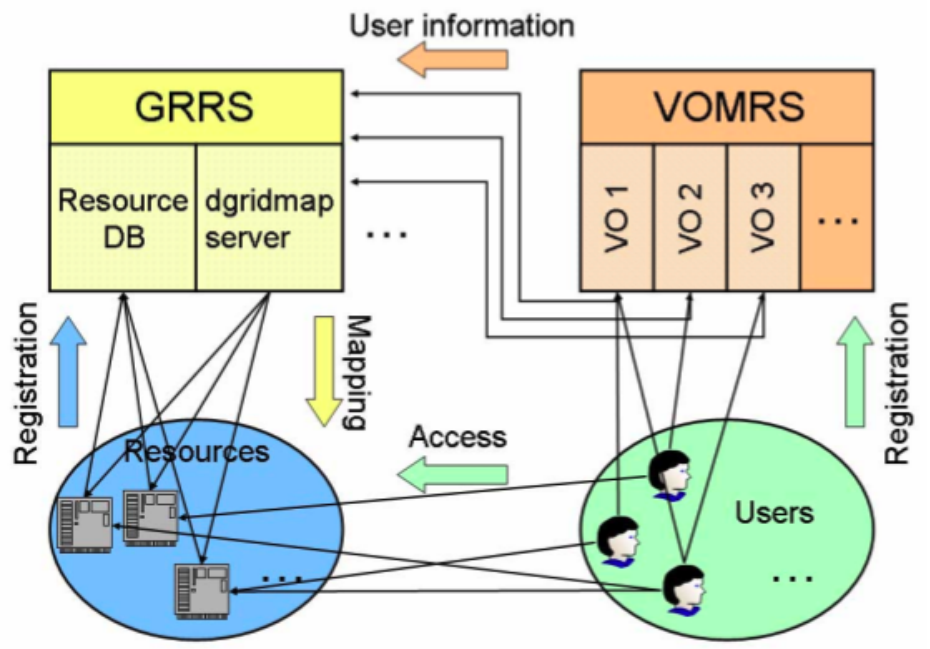



Abbildung 4.7: Betrieb der VO- und Benutzerdienste und der Ressourcen [BDE+ 07].

### 4.6.2 GRRS

Der *Grid Resource Registry Service* ist ein D-Grid Dienst, der dazu verwendet wird, alle Ressourcen im D-Grid zentral zu erfassen und die wichtigsten Informationen zu jeder Ressource in einer Datenbank zu hinterlegen. Die wichtigsten Daten zu den Ressourcen sind dabei:

- Netzwerkname der Ressource.

- Beschreibung der Ressource.
- E-Mail Adressen und DNs der Administratoren.
- Die Grid Host-Zertifikate für die Identifikation der Ressource im Grid.

Damit eine Ressource im D-Grid verwendet werden kann, muss diese über GRRS registriert werden. Die Abbildung 4.7 zeigt die Architektur von GRRS und VOMRS im D-Grid.

Da das D-Grid bereits eine zentrale VO-, Benutzer- und Ressourcen-Registrierungsstelle besitzt, könnte mit Hilfe dieser zentralen Dienste auch eine einheitliche AAI etabliert werden. Allerdings ist die Registrierung von Benutzern und Ressourcen streng getrennt und es können keine Ressourcen in VOs aufgenommen werden (aufgrund der Einschränkungen von VOMRS). Aus diesem Grund erfüllt auch die Konstellation GRRS/VOMRS die Anforderungen aus Kapitel 3 nicht.

## 4.7 Zusammenfassung

In diesem Kapitel wurde zunächst die Authentifizierung im D-Grid - welche auf X.509 Zertifikaten basiert - untersucht. Des Weiteren wurden alle Middleware-Technologien im D-Grid auf ihre Autorisierungsmechanismen hin untersucht und festgestellt, dass diese sehr unterschiedlich sind. Globus und UNICORE setzen im Wesentlichen auf ACLs (gridmap files, UUDBs), lediglich gLite bringt von Haus aus ein VO-Verständnis mit, setzt aber zusätzlich auch ACLs in Form von gridmap files ein.

Ein weiterer Abschnitt hat sich mit dem VO-Management im D-Grid beschäftigt und es wurde festgestellt, dass sich das IVOM Projekt mit der Vereinheitlichung des VO-Managements und der AAI beschäftigt. Allerdings erfüllt dieses Projekt nicht die Anforderungen aus Kapitel 3.

In den nächsten Abschnitten wurde die Funktionsweise von VOMS/VOMRS und Shibboleth erklärt und ihre Eignung bezüglich einer einheitlichen AAI untersucht. Aber auch diese beiden Konzepte konnten die an sie gestellten Anforderungen nicht erfüllen.

Im letzen Abschnitt wurde das D-Grid Betriebskonzept erklärt und festgestellt, dass es zentrale Registrierungsdienste für VOs und Benutzer (VOMRS), als auch für Ressourcen (GRRS) gibt. Aufgrund der strikten Trennung von VO- und Benutzerregistrierung auf der einen und Ressourcenregistrierung auf der anderen Seite, kommt auch dieses Konzept für eine einheitliche AAI mit den hier gestellten Anforderungen nicht in Frage.

Die Tabelle 4.1 stellt nochmals die wichtigsten Eigenschaften der Middleware-Technologien bezüglich AAI und VO-Konzept dar. Tabelle 4.2 fasst die Bewertung, der in den Grid Middleware-Technologien eingesetzten VO-Management- und AAI-Konzepten zusammen. Der Tabelle 4.3 ist zu entnehmen, dass keines der in diesem Kapitel untersuchten AAI-Konzepte alle Anforderungen aus Kapitel 3 erfüllt. Die Authentifizierung der Benutzer ist mit allen untersuchten Konzepten einheitlich möglich, allerdings scheitern sie an der Vereinheitlichung der Autorisierung. Der Grund dafür liegt in der Anforderung, dass der VO-Layer die Autorisierung der Benutzer übernimmt und somit müssen auch die Ressourcen in die

VO-Struktur aufgenommen werden. Keines der in diesem Kapitel untersuchten Konzepte die Ressourcen in die VO-Struktur aufnehmen und deshalb werden zusätzliche ACLs (in Form von gridmap files und UUDBs) benötigt, um den Zugriff der Benutzer auf die Ressourcen zu steuern.

Tabelle 4.1: Unterschiede der einzelnen Grid Middleware-Technologien

| | GT4 | UNICORE 5 | gLite 3 |
|---|---|---|---|
| Authentifizierung | X.509 | X.509 | X.509 |
| Autorisierung | gridmap file | UUDB | VOMS-Rollen |
| VO-Konzept | nein | nein | ja |
| VOMS | optional | geplant (IVOM) | ja |
| VOMRS | optional | Diskussion (IVOM) | optional |
| Shibboleth | GridShib | geplant (IVOM) | geplant (SWITCH) |

Tabelle 4.2: Bewertung der VO-Management- und AAI-Konzepte

| | VOMS/VOMRS | Shibboleth | GRRS+VOMRS |
|---|---|---|---|
| VO-Konzept | ja | myVocs | ja (VOMRS) |
| VOs mit Ressourcen | nein | nein | nein |
| Gruppen | ja | nein | ja (VOMRS) |
| Rollen | ja | nein | ja (VOMRS) |
| Capabilities | ja | nein | ja (VOMRS) |
| Attribute | nein | ja | nein |
| Verknüpfung VOs und Benutzer mit Ressourcen | nein | nein | ja |

Tabelle 4.3: Bewertung der AAI-Konzepte anhand der Use Cases aus Kapitel 3

| | VOMS/VOMRS | Shibboleth | GRRS+VOMRS |
|---|---|---|---|
| Use Case: Authentifizieren | ja | ja | ja |
| Use Case: Autorisieren | nein (zusätzliche ACLs nötig) | nein (zusätzliche ACLs nötig) | nein (zusätzliche ACLs nötig) |
| Use Case: Management | nur VOMRS | nein | ja |

Da keine der in diesem Kapitel untersuchten Lösungen die Anforderungen an eine einheitliche AAI für heterogene Grids erfüllt, wird in den folgenden Kapiteln ein Konzept vorgestellt, wie ein solches AAI aussehen und wie es in eine bestehende heterogene Grid Umgebung - wie jener des D-Grids - integriert werden kann.

# 5 Systementwurf

In diesem Kapitel wird der VO-Layer modelliert und an die Anforderungen aus Kapitel 3 angepasst. Damit der VO-Layer mit möglichst vielen Grid Middleware-Technologien zusammenarbeiten kann, ist es wichtig, dass er eine geeignete, generische VO-Struktur besitzt, in der alle im D-Grid verwendeten Autorisierungskonzepte abgebildet werden können. Diese VO-Struktur wird im ersten Abschnitt dieses Kapitels eingeführt.

Im zweiten Abschnitt werden die grundlegenden Design Entscheidungen bezüglich Datenverwaltung und globaler Kontrollfluss getroffen. Die anschließenden Abschnitte beschäftigen sich mit der Zerlegung des Systems in Subsysteme, sowie der Systemarchitektur.

Der letzte Teil dieses Kapitels befasst sich mit dem Objektentwurf. Alle zuvor getroffenen Entscheidungen werden in den Objektentwurf aufgenommen und dort zu einem Objektmodell verfeinert. Dazu gehört insbesondere die Umsetzung der zuvor getroffenen Design Entscheidungen in ein Objektmodell und die Spezifizierung der Schnittstellen zwischen den Objekten und den verschiedenen Subsystemen.

## 5.1 VO-Struktur

Die für den VO-Layer entwickelte, generische VO-Struktur wird in diesem Abschnitt eingeführt. Zunächst werden alle Eigenschaften dieser VO-Struktur identifiziert, anschließend wird eine geeignete Darstellung für die Berechtigungsnachweise der Benutzer vorgestellt. Diese Berechtigungsnachweise dienen im VO-Layer zur Autorisierung. Die für diese Diplomarbeit entwickelte, generische VO-Struktur hat folgende Eigenschaften:

- Unterstützung von VOs und sub-VOs. Eine VO kann beliebig viele sub-VOs enthalten und selbst auch Teil von beliebig vielen super-VOs sein.
- Eine VO besteht aus beliebig vielen Benutzern und Ressourcen. Diese können Mitglied von beliebig vielen VOs sein.
- Den Benutzern und Ressourcen in einer VO wird eine Menge von Berechtigungsnachweisen zugeordnet. Diese zeigen an, welcher Benutzer auf welche Ressourcen zugreifen darf. Nur wenn der Benutzer den selben Berechtigungsnachweis wie die angeforderte Ressource hat, kann er auf diese zugreifen.
- Ein Berechtigungsnachweis ist stets eindeutig und wird durch folgende vier Eigenschaften bestimmt:
  - Der VO, der sie zugeordnet ist,
  - Dem Gruppentyp,

  - Dem Rollentyp und
  - Dem Fähigkeitentyp.

- Damit auch VO-Strukturen unterstützt werden, welche auf Gruppen, Rollen oder Fähigkeiten verzichten, gibt es für diese drei Eigenschaften jeweils auch einen *NULL* Eintrag. Somit wird, in Umgebungen welche beispielsweise keine Gruppen, jedoch Rollen unterstützen, der Gruppentyp *NULL* dem FQAN zugewiesen. Gleiches gilt für Rollen oder Fähigkeiten, welche ebenfalls *NULL* sein können.

### 5.1.1 Berechtigungsnachweis

Die Attribute der Benutzer und Ressourcen müssen im VO-Layer auf einheitliche Weise dargestellt werden. Hierzu wird eine einheitliche Schreibweise zur Darstellung der Zugehörigkeit von Gruppen, Rollen und Fähigkeiten eingeführt. Die Struktur dieser Schreibweise sieht folgendermaßen aus:

/VO[/sub-VO(s)][/Group=group][/Role=role][/Capability=cap]

Wenn beispielsweise ein Benutzer Mitglied in der VO *AstroGrid-D*, der sub-VO *München*, der Gruppe *Researcher* ist und die Rolle *Admin* besitzt, sehen die Berechtigungsnachweise für diesen Benutzer folgendermaßen aus:

/AstroGrid-D/München/Group=Researcher/Role=Admin
/AstroGrid-D/München/Group=Researcher
/AstroGrid-D/München
/AstroGrid-D

Die Berechtigungsnachweise bauen aufeinander auf, da jeder der Eigenschaften in diesem Beispiel Teil der vorhergehenden Eigenschaft ist. Der Benutzer kann somit auf Ressourcen zugreifen, die einen dieser vier Berechtigungsnachweise akzeptieren. Nicht aber auf Ressourcen mit folgenden Berechtigungsnachweisen:

/AstroGrid-D/München/Group=NULL/Role=Admin
/AstroGrid-D/Group=Researcher/Role=Admin

Ein Benutzer besitzt in der Regel mehrere Berechtigungsnachweise, welche aufeinander aufbauen. Die Ressourcen hingegen besitzen nur wenige Berechtigungsnachweise, welche genau spezifizieren welche Mitgliedschaften ein Benutzer für den Zugriff benötigt. Sollte eine Community keine Gruppen unterstützen, jedoch aber Rollen, so werden alle Berechtigungsnachweise mit den Gruppentyp *NULL* spezifiziert. Gleiches gilt für Communities ohne Unterstützung von Rollen oder Fähigkeiten.

### 5.1.2 Autorisierung

Ziel der VO-Struktur ist es, den Zugriff der Benutzer auf die Ressourcen einfach steuern und administrieren zu können. In dieser VO-Struktur wird nicht nur jedem Benutzer, sondern

auch jeder Ressource eine Menge von Berechtigungsnachweisen zugeordnet. Wenn ein Benutzer nun Zugriff auf eine bestimmte Ressource haben möchte, muss überprüft werden, ob es einen Berechtigungsnachweis gibt, den sowohl der Benutzer, als auch die Ressource besitzt. Falls es keinen derartigen Berechtigungsnachweis gibt, wird dem Benutzer der Zugriff auf die Ressource vom VO-Layer verweigert. Falls die Ressource keinen Berechtigungsnachweis besitzt, genügt einem Benutzer die Mitgliedschaft in derselben VO, um auf die Ressource zugreifen zu können.

Der VO-Layer kann somit bereits im Vorfeld Entscheidungen bezüglich der Autorisierung treffen und kann den Zugriff auf eine Ressource verweigern, noch bevor der Benutzer zur eigentlichen Grid-Middleware weitergeleitet wird. Dadurch ist es möglich, den Ressourcenzugriff feiner zu strukturieren, da der VO-Layer Gruppen, Rollen und Fähigkeiten unterstützt, die darunter liegende Grid-Middleware aber nicht zwingend.

Falls der VO-Layer festgestellt hat, dass der Benutzer Zugriff auf die gewünschte Ressource haben darf, wird die Anfrage an die Grid Middleware weitergeleitet, welche die Ressource bereitstellt. Diese muss den Benutzer erneut autorisieren, erst danach kann auf die Ressource zugegriffen werden.

### 5.1.3 VO-Struktur in VOMS

Die soeben eingeführte VO-Struktur hat zum Ziel möglichst generisch zu sein, damit die verschiedenen Autorisierungskonzepte der D-Grid Communities darin abgebildet werden können. Aus diesem Grund muss es auch möglich sein, die VO-Struktur von VOMS in der des VO-Layers abzubilden. Die folgenden Abschnitte führen die VO-Struktur von VOMS kurz ein. Anschließend wird untersucht wie die VO-Struktur von VOMS in der des VO-Layers abgebildet werden kann. Zum grundlegenden Aufbau des VOMS-Systems sei auf Kaptitel 4.4 verwiesen.

Im folgenden Abschnitt wird erklärt, wie die Struktur der VOs in VOMS aussieht und wie diese gespeichert wird. Außerdem werden die wichtigsten Erkenntnisse bezüglich der VO-Struktur und deren Aufbau wie von [ACC+ 04] detailliert beschrieben wurde, kurz zusammengefasst.

#### Benutzerattribute

Eine VO in VOMS ist eine komplexe, hierarchisch aufgebaute Struktur mit Gruppen und Untergruppen. Dabei können die Gruppen unabhängig voneinander von verschiedenen Administratoren verwaltet werden. Die Gruppenadministratoren können dabei weitere Untergruppen erstellen und dort bestimmte Attribute an die Benutzer vergeben, allerdings können sie die Attribute in anderen Gruppen nicht verändern. Jede VO hat einen VO-Administrator und nur dieser ist berechtigt neue Benutzer in die VO aufzunehmen.

Eine Gruppe besteht in VOMS aus einer Menge von Benutzern und anderen Gruppen. Ein Benutzer kann dabei Mitglied in beliebig vielen Gruppen sein. Die Mitgliedschaft eines

Benutzers in einer Gruppe impliziert die Mitgliedschaft in allen Obergruppen, bis hin zur VO. Das geschieht aus dem Grund, dass nur der VO-Administrator das Recht hat, neue Benutzer in die VO aufzunehmen. Damit die Autorisierung flexibler gestaltet werden kann, können Benutzer noch zwei Arten von Attributen haben: Rollen (engl. roles) und Fähigkeiten (engl. capabilities).

Rollen werden benutzt, um die Eigenschaften des Benutzers genauer zu spezifizieren, welche der Benutzer als Mitglied in einer Gruppe besitzt. Aus diesem Grund ist der Gültigkeitsbereich einer Rolle auf eine bestimmte Gruppe beschränkt. Der Hauptunterschied zwischen Gruppen und Rollen ist, dass der Benutzer entscheiden kann, welche seiner Rollen in seinem Berechtigungsnachweis aufgelistet sind. Die Gruppen hingegen sind für einen Benutzer immer spezifiziert. Das ermöglicht die Erstellung von Gruppen, welche die Rechte des Benutzers einschränken. Die Fähigkeiten werden verwendet, um die Charakteristiken eines Benutzers zu beschreiben.

Ein Benutzerattribut wird in VOMS als eine Zeichenkette dargestellt, welche aus einer Gruppenmitgliedschaft, Rollen und Fähigkeiten im Rahmen einer VO besteht. Diese Attribute können beschränkt oder unbeschränkt gültig sein. Der Resource Provider (RP) ist für die Umsetzung dieser VO-Policies auf dem lokalen System verantwortlich.

#### Berechtigungsnachweis

Der so genannte *Fully Qualified Attribut Name* (FQAN) ist eine kompakte Schreibweise, um die Mitgliedschaft eines Benutzers in einer Gruppe mit einer dazugehörigen Rolle und Fähigkeit darzustellen. Er dient als Berechtigungsnachweis und wird vom VOMS-System dazu benutzt, um die Zugriffsberechtigung eines Benutzers auf eine Ressource zu überprüfen. Ein FQAN wird durch eine Sequenz, bestehend aus dem Gruppennamen, gefolgt von einer Rolle und einer Fähigkeit, dargestellt. Im Allgemeinen hat ein FQAN folgende Form:

/VO[/group[/subgroup(s)]][/Role=role][/Capability=cap]

Somit sieht beispielsweise der FQAN zur Rolle *Administrator*, in der Gruppe *Forscher*, der VO *AstroGrid-D* folgendermaßen aus:

/AstroGrid-D/Forscher/Role=Administrator

Die gesamte VO-Struktur - und somit auch jeder Berechtigungsnachweis - wird von VOMS in einer relationalen Datenbank gespeichert. Das Datenbankschema von VOMS ist in Abbildung 5.1 dargestellt. Anhand der Datenbanktabellen generiert VOMS die FQANs für die Benutzer und integriert diese in ein Proxy-Zertifikat mit welchem die Benutzer Zugriff auf die Ressourcen erhalten.

### 5.1.4 Abbildung der VO-Konzepte

Die VO-Struktur des VO-Layers ist stark an jener von VOMS angelehnt. Das hat den Grund, dass VOMS eines der komplexesten Autorisierungskonzepte des D-Grids ist. Deshalb muss

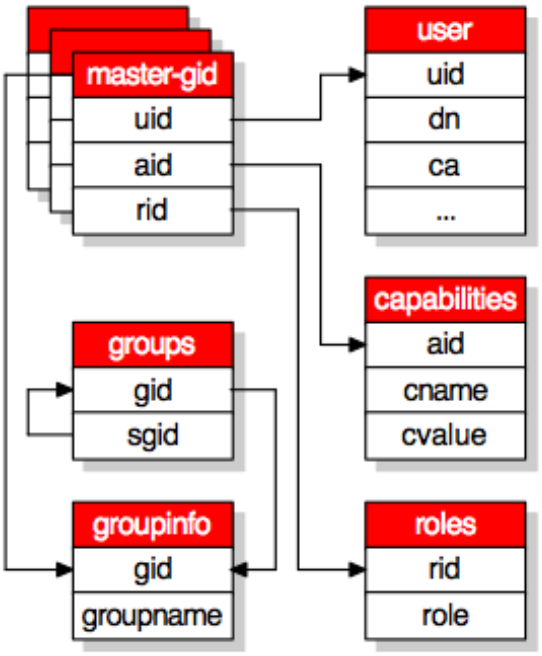



Abbildung 5.1: VOMS Datenbankschema [ACC+ 04]

der VO-Layer mindestens die VO-Struktur von VOMS in seiner eigenen VO-Struktur abbilden können. Eine VO-Struktur aus VOMS kann in eine VO-Struktur des VO-Layers überführt werden, indem alle Subgruppen der VOs in VOMS in sub-VOs überführt werden. Die Unterstützung für Gruppen, Rollen und Fähigkeiten ist im VO-Layer ebenso wie in VOMS vorhanden.

Der VO-Layer unterstützt zusätzlich noch all jene VO-Konzepte, die eine beliebige Teilmenge der drei Benutzerattribute verwenden. Somit kann der VO-Layer als Ersatz für alle VO-Konzepte der D-Grid Communities verwendet werden, selbst für jene Communities die keine VOs unterstützen. In diesem Fall werden die Benutzer, Ressourcen und Berechtigungsnachweise in eine einzige VO eingetragen, welche die Community selbst darstellt.

## 5.2 Entwurfsziele

Das gesamte System wird in Abschnitt 5.4 in mehrere Subsysteme zerlegt, die unabhängig voneinander entwickelt werden können. Die systemübergreifenden Punkte *Datenverwaltung* und *globaler Kontrollfluss* werden zuvor im folgenden Abschnitt geklärt.

### 5.2.1 Datenverwaltung

Das Datenmodell des VO-Layer basiert auf dem Composite-Pattern [BrDu 04] und kann dadurch eine flexieble Struktur aus VOs, mit dazugehörigen sub-VOs, Benutzern und Ressourcen annehmen. Zusätzlich werden auch noch Gruppen, Rollen und Fähigkeiten unterstützt, die zu Berechtigungsnachweisen zusammengefasst werden. Abbildung 5.2 zeigt das Datenmodell des VO-Layers als Klassendiagramm. Darin sind auch die Attribute und Operationen spezifiziert, welche die einzelnen Klassen unterstützen.

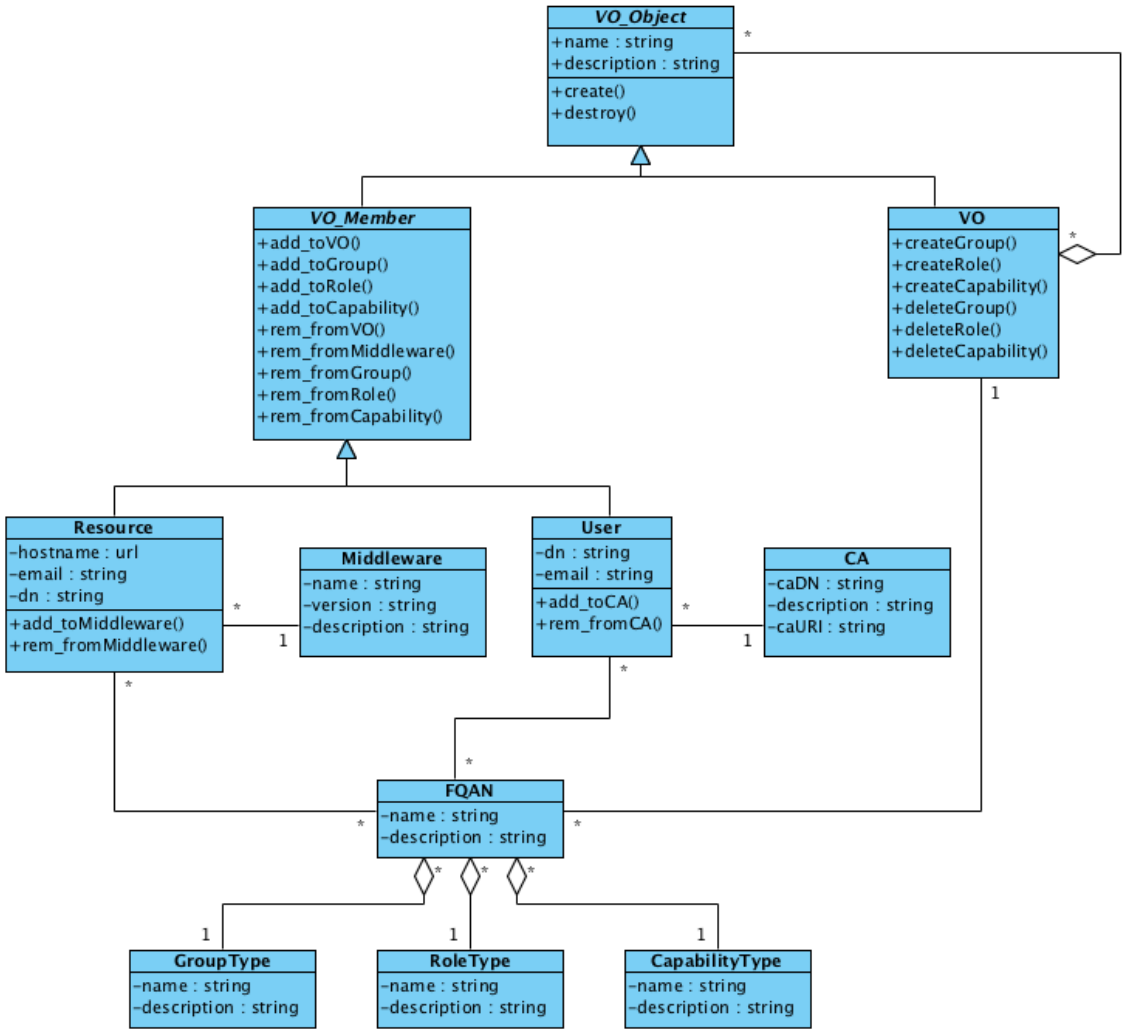



Abbildung 5.2: Das Datenmodell des VO-Layers zur Darstellung der VO-Struktur.

Die Abstrakte Klasse `VO_Object` bezeichnet die Komponente des Composite-Patterns. Davon abgeleitet ist die Klasse `VO`, welche als Kompositum fungiert, sowie die Klassen `User` und `Resource`, welche die Blätter des Composite-Patterns darstellen. Die abstrakte Klasse `VO_Member`, die direkt von `VO_Object` abgeleitet ist, stellt gemeinsame Ressourcen für die Klassen `User` und `Resource` zur Verfügung.

Die Klassen `VO`, `User` und `Resource` besitzen beliebig viele Berechtigungsnachweise, welche im Folgenden - analog zu VOMS - als *FQANs* bezeichnet werden. Die Klasse `FQAN` ist mit je einem Objekt der Klassen `GroupType`, `RoleType` und `CapabilityType` assoziiert. Des Weiteren kann ein FQAN beliebig vielen Ressourcen und Benutzern zugeordnet werden, aber nur genau einer VO.

Jede Ressource ist mit genau einer `Middleware` Klasse assoziiert, wobei eine Middleware beliebig viele Ressourcen bereitstellen kann. Das gleiche gilt für die Benutzer, die alle genau ein `CA` Objekt besitzen.

Zur persistenten Speicherung des Datenmodells des VO-Layers wird in dieser Diplomarbeit ein Datenbankschema entworfen, welches das Klassendiagramm aus Abbildung 5.2 in ein relationales Datenbankschema umsetzt. In Abbildung 5.4 ist das Datenbankschema dargestellt.

Für die Gruppen, Rollen und Fähigkeiten existiert jeweils eine Tabelle, welche den Typ spezifiziert (*GroupType*, *RoleType* und *CapabilityType*). Aus diesen wird der FQAN generiert, welcher stets aus einer Kombination von VO, Gruppentyp, Rollentyp und Fähigkeitentyp zusammengesetzt ist. Das hat den Vorteil, dass eine Reihe von Gruppentypen spezifiziert werden kann, welche in unterschiedlichen VOs verwendet werden können (beispielsweise kann der Gruppentyp *Researcher* in vielen VOs vorkommen). Sobald ein Benutzer Zugriff auf eine Ressource haben möchte, muss der VO-Layer überprüfen, ob der Benutzer und die Ressource in der selben VO sind und ob der Benutzer mindestens einen gleichen FQAN wie die angeforderte Ressource besitzt.

Nicht nur die VO-Struktur muss persistent gespeichert werden, sondern auch die Zertifikate der Benutzer. Hierzu wird auf jedem System, auf dem der VO-Layer installiert wird, das neue Verzeichnis `.vo-layer` im Home-Verzeichnis des ausführenden Benutzers angelegt. Im Unterverzeichnis `certs` werden alle Zertifikate des Benutzers in separaten Ordnern gespeichert. Neben den Benutzerzertifikaten werden - in regelmäßigen Abständen - die Änderungen der VO-Struktur, in Form von gridmap files und UUDBs in weiteren Verzeichnissen abgelegt. Diese werden anschließend an die entsprechenden Grid Middleware-Technologien weitergeleitet um dort die Autorisierungsinformationen zu aktualisieren (siehe dazu auch Kapitel 6.2.4). Abbildung 5.3 stellt diese Verzeichnisstruktur graphisch dar.

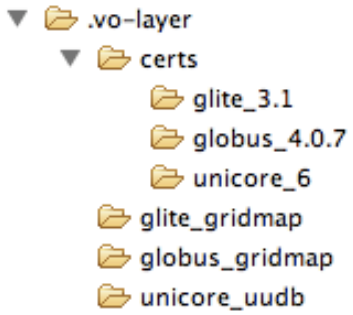



Abbildung 5.3: Verzeichnisstruktur zur Speicherung der Zertifikate, gridmap files und UUDBs.

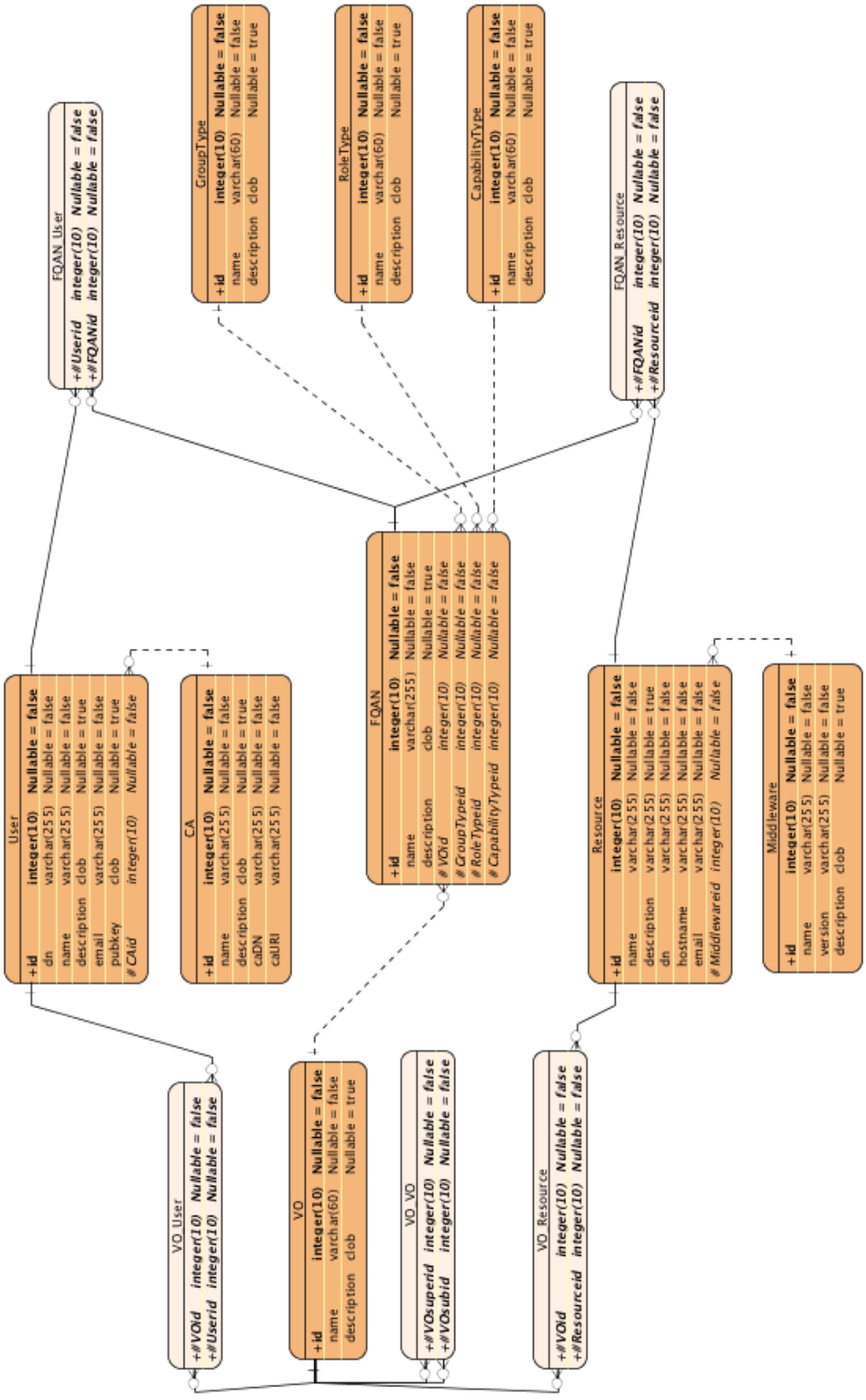



Abbildung 5.4: Das Datenbankschema des VO-Layers.

### 5.2.2 Globaler Kontrollfluss

Der VO-Layer ist darauf ausgelegt, dass mehrere Benutzer und auch VO-Administratoren parallel damit arbeiten können. Das wird durch die Unterstützung von Transaktionen in der VO-Layer Datenbank erreicht. Dadurch ist die Datenstruktur stets in einem konsistenten Zustand. Der VO-Layer implementiert keine Aufgaben der eigentlichen Middleware-Technologie, sondern leitet die Anfragen - nach kurzer Überprüfung - an die eigentliche Middleware weiter. Die Grid-Middleware ist es auch, die den Benutzer authentifiziert. Der VO-Layer sendet hierzu lediglich die Anfrage, mit allen Parametern des Benutzers (wie etwa dem Benutzerzertifikat), an die Grid-Middleware. Diese überprüft das Zertifikat und stellt dem Benutzer - nach erfolgreicher Prüfung - ein Proxy-Zertifikat aus. Diesen Ereignisfluss kann man im UML-Sequenzdiagramm in Abbildung 5.5 sehen.

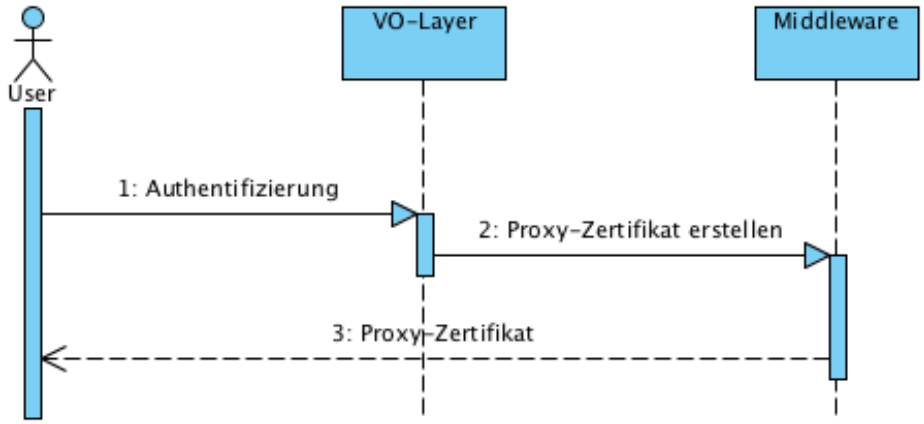



Abbildung 5.5: Globaler Kontrollfluss für den Use Case: *Authentifizieren*.

Die Autorisierung eines Benutzers für den Ressourcenzugriff läuft auch über den VO-Layer. Dieser überprüft zunächst die Berechtigungen des Benutzers und im Erfolgsfall leitet er den Grid-Job an die Middleware-Technologie weiter, wo die Autorisierung erneut geprüft und anschließend der Job gestartet wird. In Abbildung 5.6 ist dieser Ablauf in Form eines UML-Sequenzdiagramms zu sehen.

Für die Verwaltung des VO-Layers gibt es verschiedene Management Use Cases, welche in Kapitel 3.2.3 eingeführt wurden. Abbildung 5.7 zeigt das UML Sequenzdiagramm für den globalen Kontrollfluss eines ausgewählten Management Use Cases. Der Administrator führt eine Aktion zur Änderung der VO-Struktur aus. Diese Änderung wird in der VO-Layer Datenbank gespeichert, gleichzeitig muss die Änderung auch an die Grid Middleware-Technologien weitergegeben werden. Diese müssen dann eventuell das gridmap file anpassen (Globus und gLite) oder die UUDB aktualisieren (UNICORE).

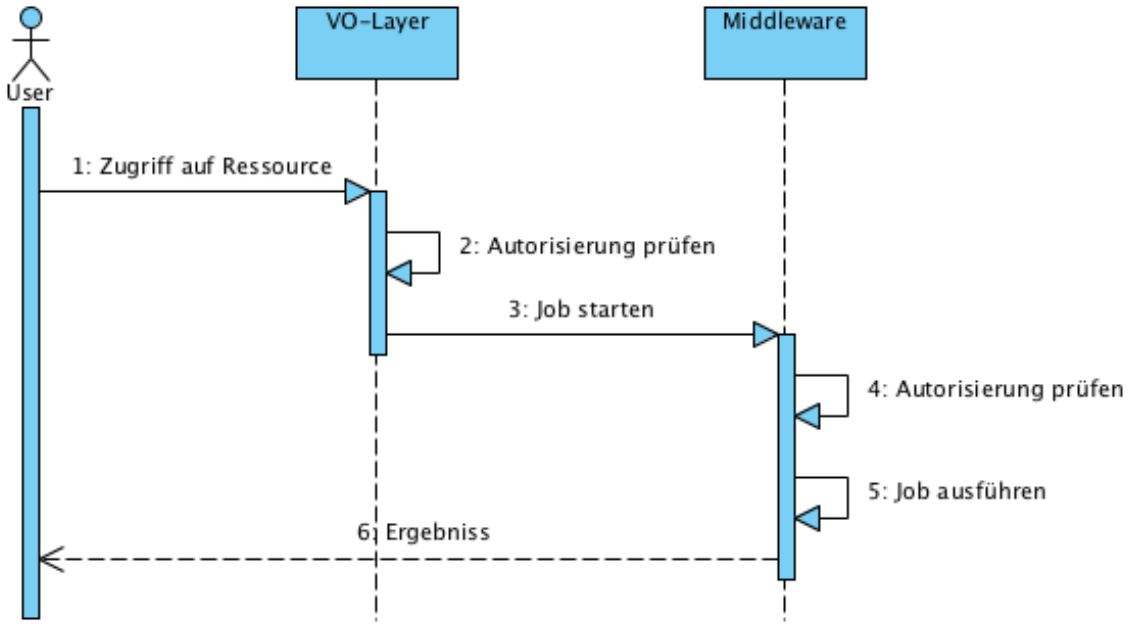



Abbildung 5.6: Globaler Kontrollfluss für den Use Case: *Autorisieren*.

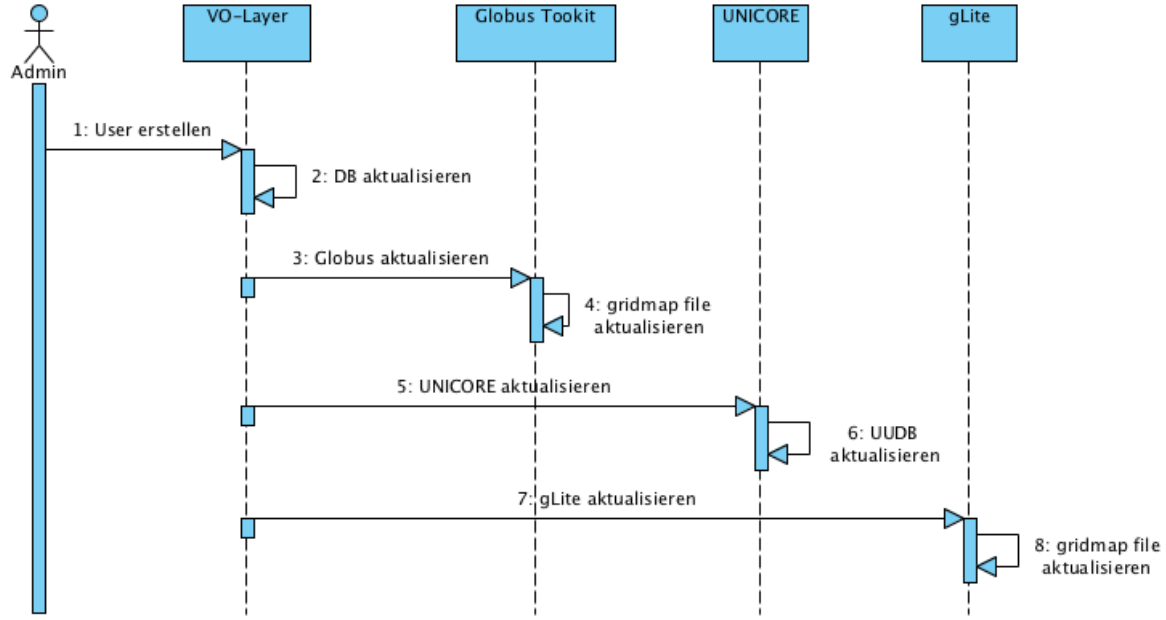



Abbildung 5.7: Globaler Kontrollfluss für das Management am Beispiel des Use Case: *Benutzer erstellen*.

## 5.3 Softwarearchitektur

Die Architektur eines Softwaresystems beschreibt dessen grundlegenden Elemente und Strukturen. Sie beschreibt die hierarchische und strukturierte Anordnung der Systemkomponenten, sowie ihre Beziehungen zueinander.

Die Softwarearchitektur des VO-Layer basiert auf dem *Model-View-Controller* (MVC) Architekturstil (Abbildung 5.8) [BrDu 04], der aus den drei Komponenten *Model*, *View* und *Controller* besteht. Das Model vereint das Wissen über die Objekte des Systems. Der View ist für die Beschaffung, Darstellung und Aktualisierung der relevanten Daten aus dem Model, nicht aber für die Interaktion mit dem Benutzer zuständig. Der Controller verwaltet den View, nimmt Benutzeraktionen entgegen, wertet diese aus und agiert entsprechend. Der Controller erhält somit die Intelligenz und steuert den Ablauf der Views. Dadurch erhält man ein flexibles Programmdesign, in dem es recht einfach ist, Änderungen vorzunehmen oder Erweiterungen zu implementieren. Das Model ändert sich nach der Spezifikation kaum noch, aber der Controller und der View können sich ständig ändern, wobei auch mehrere Controller und Views für das gleiche Model existieren können. Im Rahmen dieser Diplomarbeit wird das Model und der Controller prototypisch implementiert. Der View hingegen wird nicht implementiert.

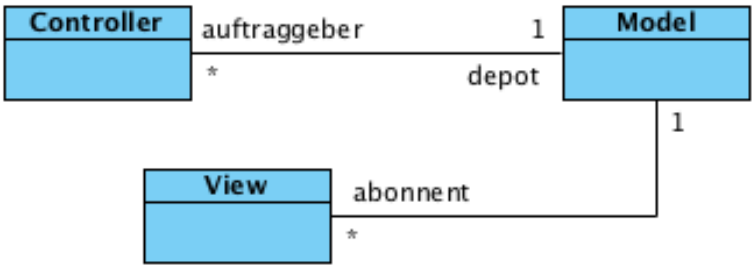



Abbildung 5.8: Model-View-Controller Architekturstil [BrDu 04]

## 5.4 Subsysteme

Nach der Definition der Anforderungen kann der VO-Layer in mehrere Subsysteme zerlegt werden. Dabei wird Wert darauf gelegt, dass die Kopplung zwischen den Subsystemen möglichst gering ist. Abbildung 5.9 zeigt die Subsysteme des VO-Layers und dessen Abhängigkeiten untereinander. Alle Subsysteme werden vom `Main` Subsystem aus aufgerufen und können dann ihrerseits auf weitere Subsysteme zugreifen. Im Folgenden werden die einzelnen Subsysteme kurz eingeführt.

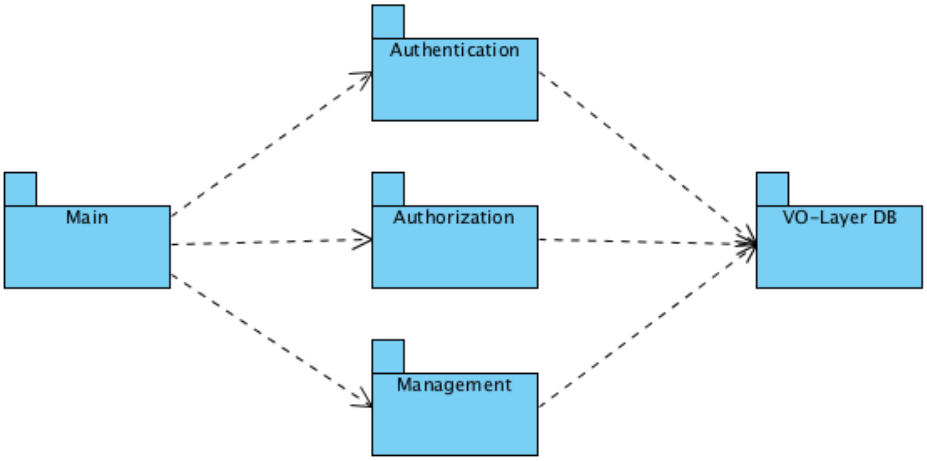



Abbildung 5.9: Systemzerlegung des VO-Layers.

### Authentication

Das `Authentication` Subsystem leitet sich aus dem Use Case *Authentifizieren* aus Kapitel 3.2.1 ab. Die Aufgabe dieses Subsystems ist es, den Benutzer zu authentifizieren und ein Proxy-Zertifikat mit Hilfe der Grid-Middleware zu generieren. Da jede Grid-Middleware unterschiedliche Grid-Befehle zur Authentifizierung verwendet, muss das Subsystem für jede eingesetzte Middleware eine eigene Methode bereitstellen. Diese Methode setzt den Grid-Befehl bei der entsprechenden Middleware-Technologie ab und die Middleware erzeugt daraufhin das Proxy-Zertifikat.

### Authorization

Das `Authorization` Subsystem wird aus dem Use Case *Autorisieren* aus Kapitel 3.2.2 abgeleitet. Wie das `Authentication` Subsystem, benötigt auch das `Authorization` Subsystem Zugang zur Grid-Middleware und muss für jede eingesetzte Middleware-Technologie eine eigene Methode bereitstellen, welche für die Autorisierungsfunktion der entsprechenden Middleware-Technologie aufruft. Des Eeiteren werden Informationen aus der VO-Layer Datenbank benötigt, um entscheiden zu können ob ein Benutzer Zugang zu einer Ressource hat oder nicht. Dazu wird das `VO-Layer DB` Subsystem verwendet.

### Management

Das `Management` Subsystem implementiert die Use Cases aus Kapitel 3.2.3. Die Änderungen, welche der Administrator an der VO-Struktur vornimmt, werden an das `VO-Layer DB` Subsystem weitergeleitet. Sobald die Änderungen in der Datenbank festgeschrieben wurden, müssen sie an die jeweilige Grid Middleware-Technologie weitergeleitet werden. Dazu enthält das Subsystem Methoden, welche regelmäßig die VO-Struktur in Form von grid-

map files (Globus Toolkit und gLite) und UUDBs (UNICORE) exportiert. Diese gridmap files und UUDBs werden anschließend in den entsprechenden Grid Middleware-Technologien importiert.

#### VO-Layer DB

Das `VO-Layer DB` Subsystem übernimmt die Speicherung der VO-Struktur in der zentralen VO-Layer Datenbank. Dieses Subsystem wird immer dann benötigt, wenn Änderungen an der VO-Struktur vorgenommen werden oder die Daten aus der VO-Layer Datenbank ausgelesen werden müssen. Dieses Subsystem muss sehr effizient implementiert werden, da es den gleichzeitigen Zugriff mehrerer VO-Administratoren und vieler Benutzer bewältigen muss.

## 5.5 Objektentwurf

In diesem Abschnitt wird das in dieser Diplomarbeit implementierte Objektmodell vorgestellt. Dazu wird auf das Datenmodell des VO-Layers aus Abbildung 5.2 zurückgegriffen und dieses an die Programmierumgebung angepasst, um die VO-Layer Daten persistent zu speichern. In Abbildung 5.10 ist das Objektmodell des VO-Layer Datenmodells zu sehen. Dieses wird, mit Hilfe der im `VO-Layer DB` Subsystem implementierten objektrationalen Abbildung (engl. object-relational mapping, ORM), von einem Objektmodell in ein relationales Datenbankschema übersetzt. Die Methoden zum Ändern der VO-Struktur, welche im ursprünglichen Datenmodell in Abbildung 5.2 noch enthalten sind, werden in den entsprechenden Controller-Klassen des VO-Layers implementiert. Das Objektmodell der VO-Struktur enthält nur noch die Attribute der Objekte, welche mit Hilfe des ORM in die Datenbanktabellen aus Abbildung 5.4 gespeichert werden.

Das Objektmodell aus Abbildung 5.11 speichert die Informationen zu den Zertifikaten, wobei für jede Grid Middleware-Technologie eine eigene Zertifikat-Klasse benötigt wird. Die Zertifikat-Objekte werden für die Authentifizierung der Benutzer benötigt und je nach verwendeter Grid-Middleware werden andere Daten für den Authentifizierungvorgang von der Middleware angefordert. Das in Abbildung 5.12 dargestellte Objektmodell wird benötigt um aus den Daten der VO-Layer Datenbank die gridmap files für Globus Toolkit und gLite, sowie die UUDBs für UNICORE zu erstellen.

Das Objektmodell des Hauptprogramms mit den aufgerufenen Klassen ist in Abbildung 5.13 dargestellt. Zunächst wird die `main`-Methode der Klasse `Start` aufgerufen, diese wiederum erzeugt eine `Application` und startet sie. Diese kann nun die Subsysteme `Authentication`, `Authorization` und `Management` ansteuern. Diese wiederum können über ihre Methoden auf die VO-Layer Datenbank zugreifen und erhalten über das `DBResultInterface` die Ergebnisse der Datenbankabfragen. Von den Subsystemen `Authentication`, `Authorization` und `Management` sind jeweils nur die Controller implementiert.

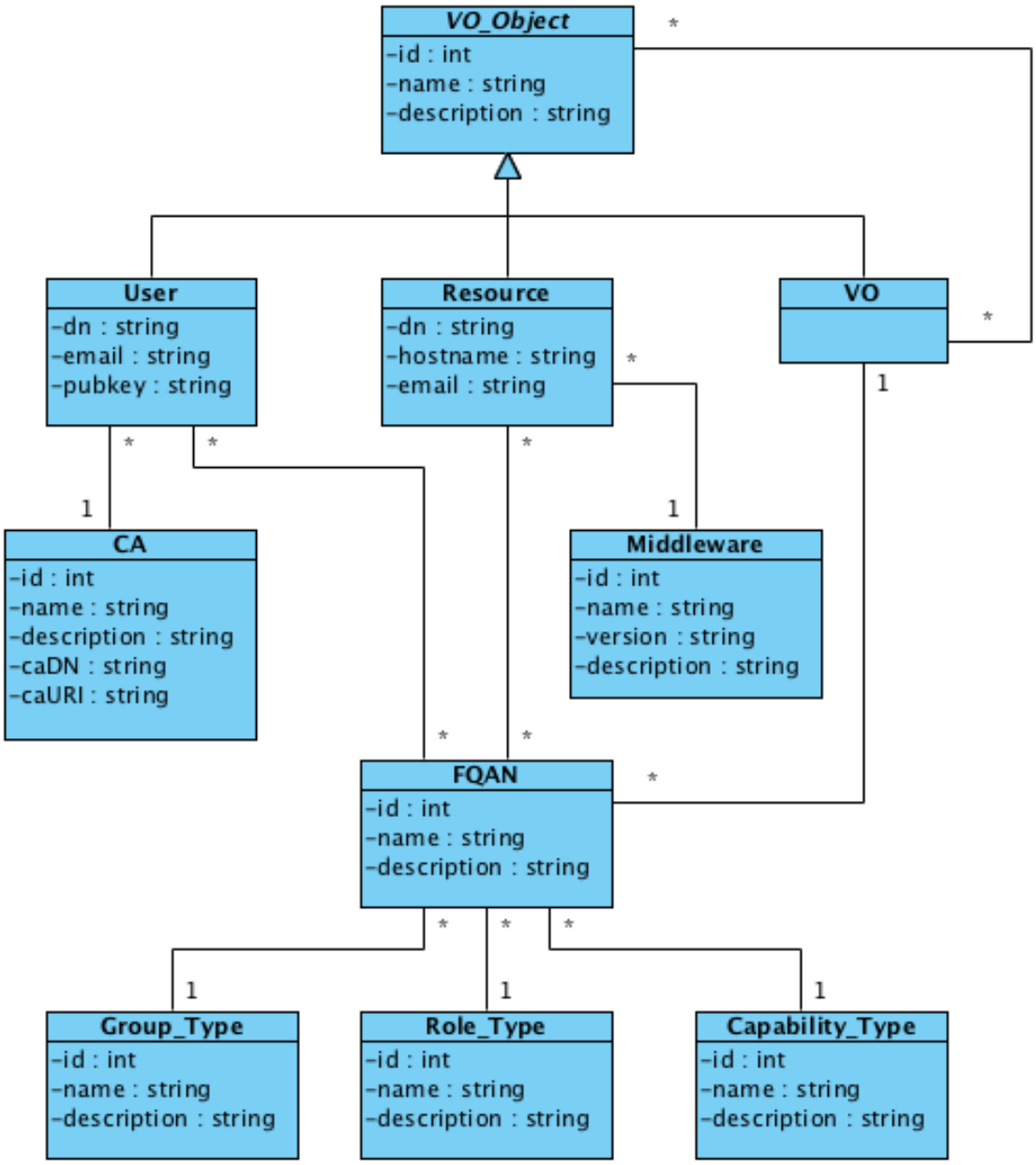



Abbildung 5.10: Objektmodell des VO-Layer Datenmodells.

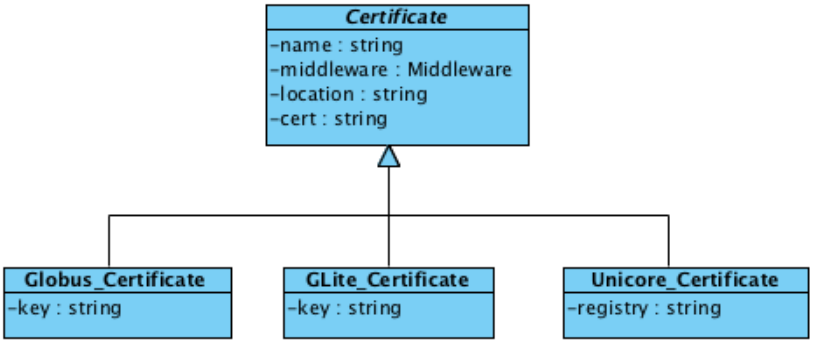



Abbildung 5.11: Objektmodell zur Speicherung der Zertifikate.

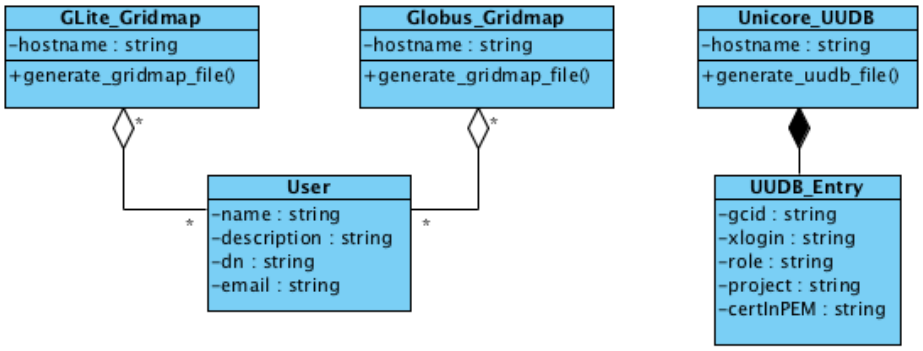



Abbildung 5.12: Objektmodell zur Speicherung der gridmap files und UUDBs.

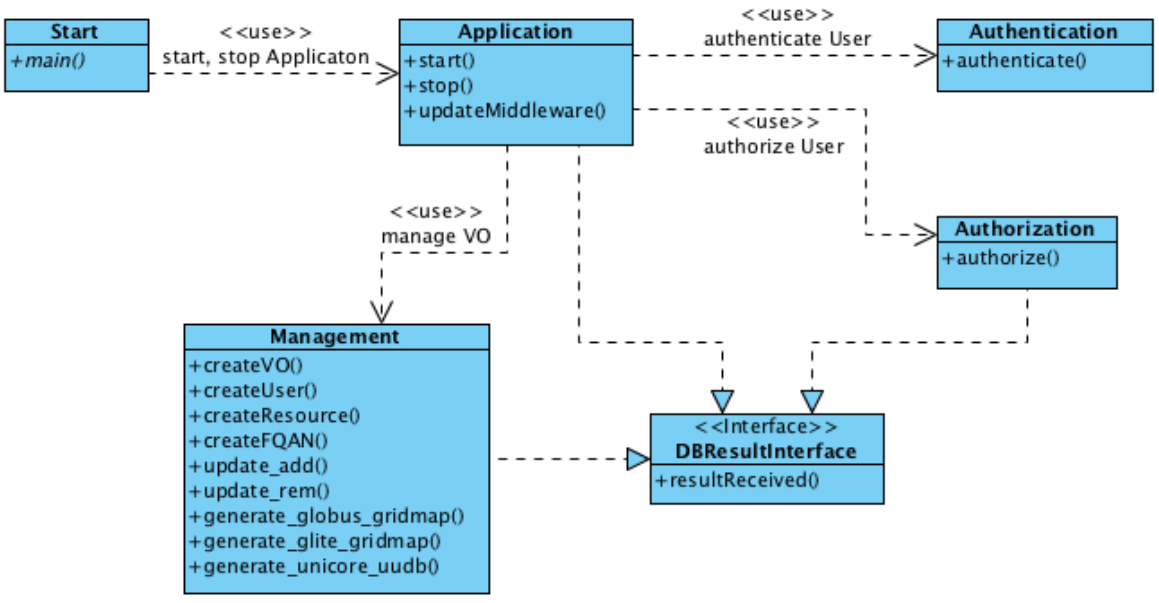



Abbildung 5.13: Objektmodell des Hauptprogramms und der aufgerufenen Klassen.

Des Weiteren wird ein Objektmodell für das `VO-Layer DB` Subsystem benötigt (Abbildung 5.14). Dieses Subsystem ist vollständig von den anderen Subsystemen und der Hauptanwendung abgekoppelt. Initiiert wird das `VO-Layer DB` Subsystem vom `DBManager`, der eine Reihe von statischen Methoden bereitstellt, welche in allen Subsystemen benutzt werden können, um Datenbank-Abfragen auszuführen. Bei der Ausführung einer solchen statischen Methode wird zunächst ein `VolDBConnection` Objekt erstellt, oder ein bestehendes verwendet. Anschließend wird die gewünschte Datenbankabfrage als `DBQuery` Objekt gespeichert und in der Liste der auszuführenden Abfragen hinzugefügt. Der `QueryThread` ist für das Ausführen dieser Datenbankabfragen zuständig und führt - je nach spezifizierten `Enum_QueryType` - eine bestimmte Methode aus der `VolDBConnection` Klasse aus. In dieser Methode wird mit Hilfe der `LoggerConnection` und des `LoggerStatements` eine SQL-Abfrage auf der Datenbank ausgeführt. Anschließend werden die Ergebnisse dieser Abfrage an das aufrufende Subsystem zurückgegeben. Zur Veranschaulichung stellt Abbildung 5.15 den Ablauf einer Datenbankabfrage als UML-Sequenzdiagramm dar.

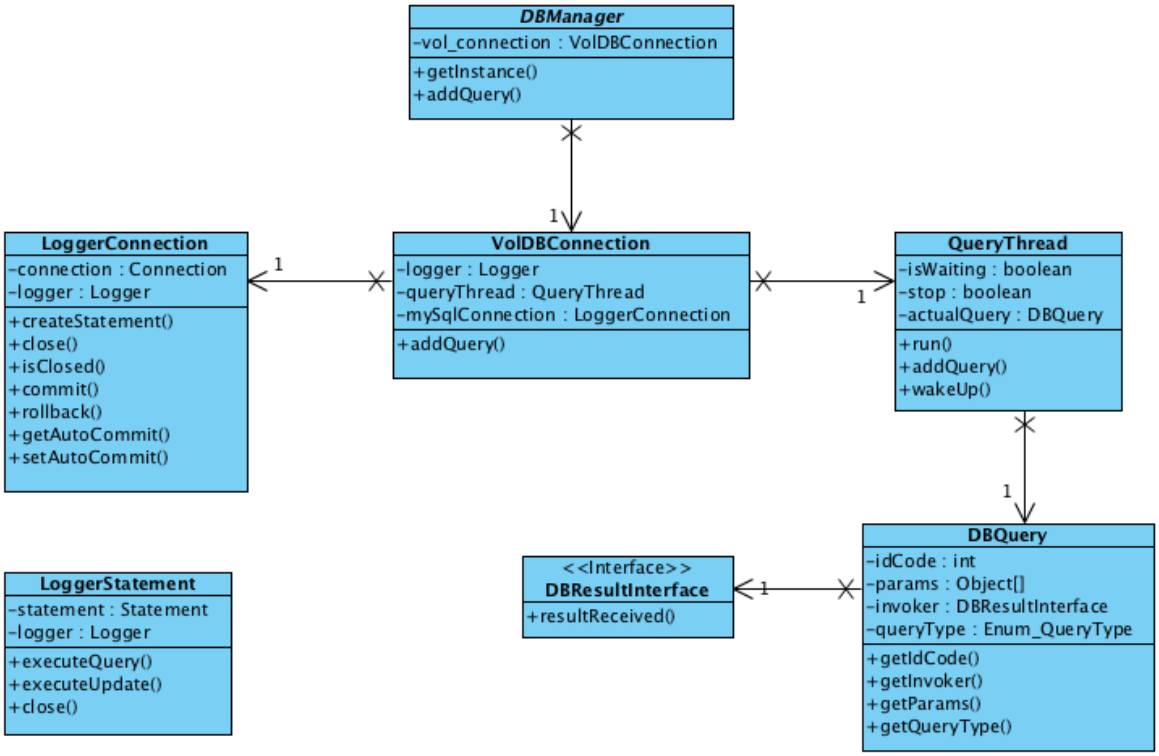



Abbildung 5.14: Objektmodell des `VO-Layer DB` Subsystems.

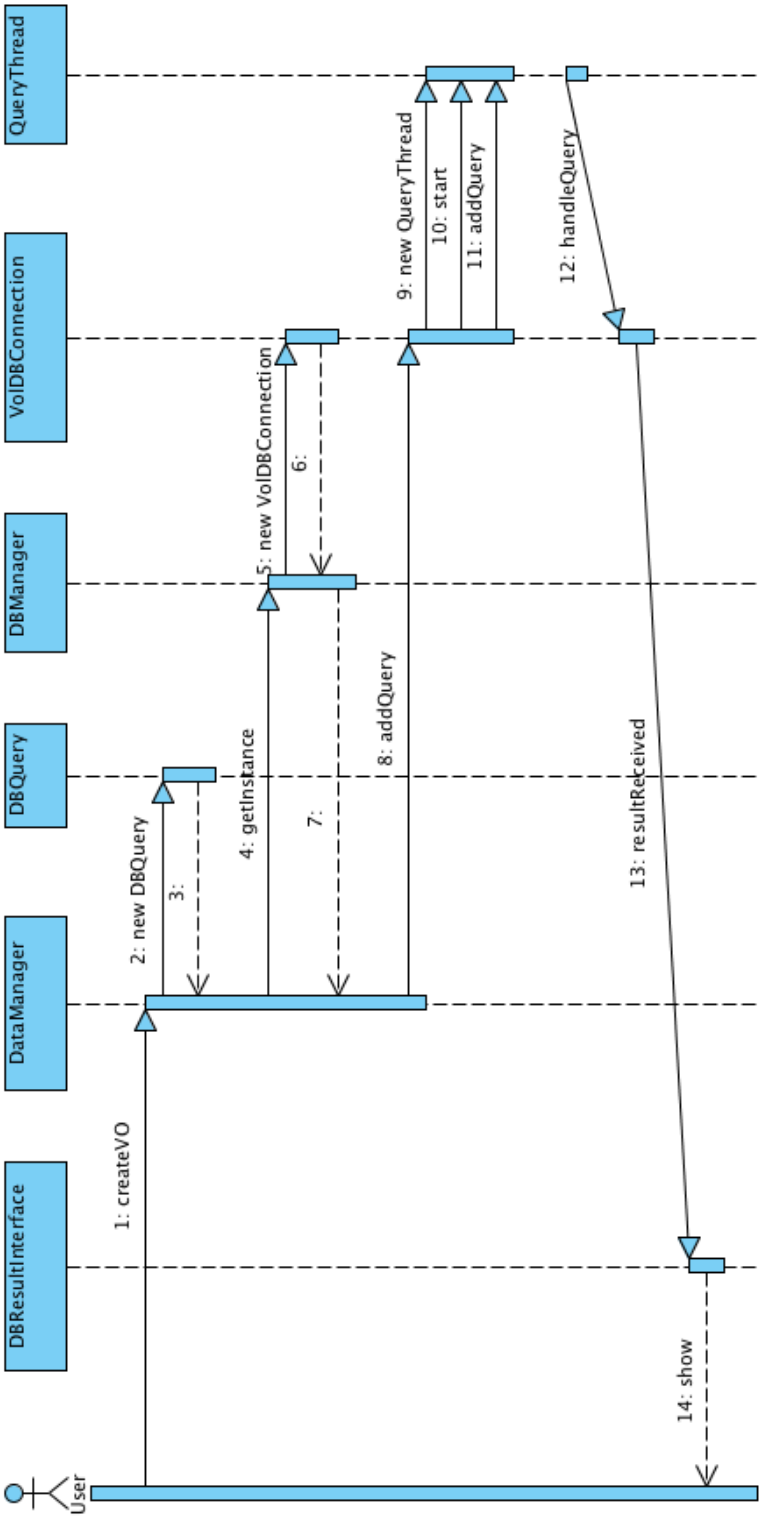



Abbildung 5.15: UML-Sequenzdiagramm des `VO-Layer DB` Subsystems am Beispiel des *createVO* Use Case. Der Schritt 5 wird nur durchgeführt falls noch keine `VolDBConnection` erstellt wurde. Die Schritte 9 und 10 werden nur durchgeführt falls noch kein `QueryThread` existiert.

## 5.6 Zusammenfassung

In diesem Kapitel wurde das System des VO-Layers, anhand der in Kapitel 3 identifizierten Anforderungen, entworfen. Dazu wurde zunächst die VO-Struktur von VOMS analysiert, um daraus eine geeignete, generische VO-Struktur für die Anforderungen dieser Diplomarbeit zu entwickeln.

Für die Datenverwaltung wurde ein Datenbankschema entwickelt, welches die VO-Struktur - in geeigneter Form - in eine relationale Datenbank speichert (siehe dazu Abbildung 5.4). Neben der Datenbank werden auch im Dateisystem des Rechners, auf dem der VO-Layer ausgeführt wird, Informationen zu den Benutzerzertifikaten, den gridmap files und den UUDBs gespeichert. Der globale Kontrollfluss wurde anhand mehrerer UML-Sequenzdiagramme dargestellt und zeigt die Interaktionen zwischen den Benutzern, dem VO-Layer und der darunter liegenden Grid-Middleware.

Als Softwarearchitektur wurde für den VO-Layer der MVC-Architekturstil gewählt, wobei im Rahmen dieser Arbeit ausschließlich das Model und der Controller prototypisch implementiert werden. Nach der Entscheidung für den MVC-Architekturstil wurde der VO-Layer in Subsysteme unterteilt, wobei es die Subsysteme `Main`, `Authentication`, `Authorization` `Management` und `VO-Layer DB` gibt.

Der letzte Teil diese Kapitels widmete sich dem Objektentwurf. Es wurden zunächst die Objektmodelle für die persistente Speicherung vorgestellt. Dazu gehören das Objektmodell des VO-Layer Datenmodells (Abbildung 5.10) sowie die Objektmodelle zur Speicherung der Zertifikate, der gridmap files und UUDBs (Abbildungen 5.11 und 5.12). Neben den Objektmodellen für die persistente Speicherung, wurde auch noch das Objektmodell des Hauptprogramms und die Beziehung zwischen diesem und den Subsystemen erläutert (Abbildung 5.13). Zuletzt wurde noch das `VO-Layer DB` Subsystem genauer vorgestellt und ebenfalls als Objektmodell eingeführt (Abbildung 5.14).

Im nächsten Kapitel wird, aus dem hier vorgestellten Systementwurf und aus dem Objektmodell, ein Prototyp in einer Testumgebung implementiert. Dazu wird zunächst die Testumgebung vorgestellt und anschließend die wichtigsten Teile des Prototypen genauer erklärt.

# 6 Implementierung

Die Implementierung beschäftigt sich mit der Erstellung des Quellcodes aus dem im vorherigen Kapitel entwickelten Systementwurf. Zunächst wird die Testumgebung vorgestellt, in der der VO-Layer entwickelt wird, anschließend werden die wichtigsten Teile aus dem Quellcode genauer beschrieben. Der letzte Abschnitt in diesem Kapitel beschäftigt sich mit dem Deployment des VO-Layers in einer Grid-Umgebung.

## 6.1 Testumgebung

Das in Kapitel 5 entwickelte System wird in einer Testumgebung implementiert. Diese Testumgebung beinhaltet alle Grid Middleware-Technologien, die auch im D-Grid vorhanden sind. Das bedeutet, dass das Globus Toolkit, UNICORE und gLite installiert sein müssen. Die Testumgebung wird auf einem 64-Bit System mit Gentoo Linux 2008.0[1] eingerichtet. Dabei werden folgende Softwarepakete für die Implementierung des VO-Layers benötigt:

| **Software** | **Version** | **Homepage** |
|---|---|---|
| Java SDK | 1.5 | http://www.java.com/ |
| GNU bash | 3.2.33 | http://www.gnu.org/software/bash |
| MySql | 5.0.54 | http://www.mysql.com/ |
| MySql Connector | 1.5.6 | http://www.mysql.de/products/connector/j/ |
| Apache log4j | 1.2.15 | http://logging.apache.org/log4j/ |
| Apache ant | 1.7.0 | http://ant.apache.org/ |
| VirtualBox | 1.6.4 | http://www.virtualbox.org/ |

Tabelle 6.1: Benötigte Softwarepakete

Neben den soeben genannten Softwarepaketen, müssen auch noch die Grid Middleware-Technologien in der Testumgebung installiert sein. Die Installation von Globus Toolkit und von UNICORE erfolgt nativ auf dem Testsystem. Für UNICORE wird, neben dem UNICORE Server auch noch der UNICORE Commandline Client (UCC) für den Zugriff auf den Server benötigt. Die Installation von gLite kann nur auf einer speziellen Linux Distribution, dem Scientific Linux[2] (SL), erfolgen. Dazu wird für gLite eine virtuelle Umgebung mit Hilfe der Software VirtualBox erstellt. Auf dieser VM wird Scientific Linux Cern[3] (SLC) - die Cern Version von SL - als Betriebssystem installiert und anschließend gLite. Bei der Installation von gLite kommt lediglich der VOMS-Knoten zum Einsatz, alle anderen gLite-Knoten

[1] http://www.gentoo.org/
[2] http://www.scientificlinux.org/
[3] http://linux.web.cern.ch/linux/scientific4/

werden in dieser Testumgebung nicht installiert. Dieser Knoten stellt einen VOMS-Server zur Verfügung und übernimmt die zentrale Autorisierung der Benutzer im gLite-Grid. Tabelle 6.2 fasst alle benötigten Grid Middleware-Komponenten zusammen und Abbildung 6.1 stellt die Komponenten der Testumgebung graphisch dar. Für eine genaue Anleitung der Installation von Globus Toolkit, UNICORE und gLite, sei an dieser Stelle auf Anhang A verwiesen.

| Software | Version | Homepage |
|---|---|---|
| Globus Toolkit | 4.0.7 | http://www.globus.org/ |
| UNICORE Server | 6.1.1 | http://www.unicore.eu/ |
| UCC | 1.1.1 | http://www.unicore.eu/ |
| gLite (VOMS server) | 3.1 | http://www.cern.ch/glite |

Tabelle 6.2: Middleware Software

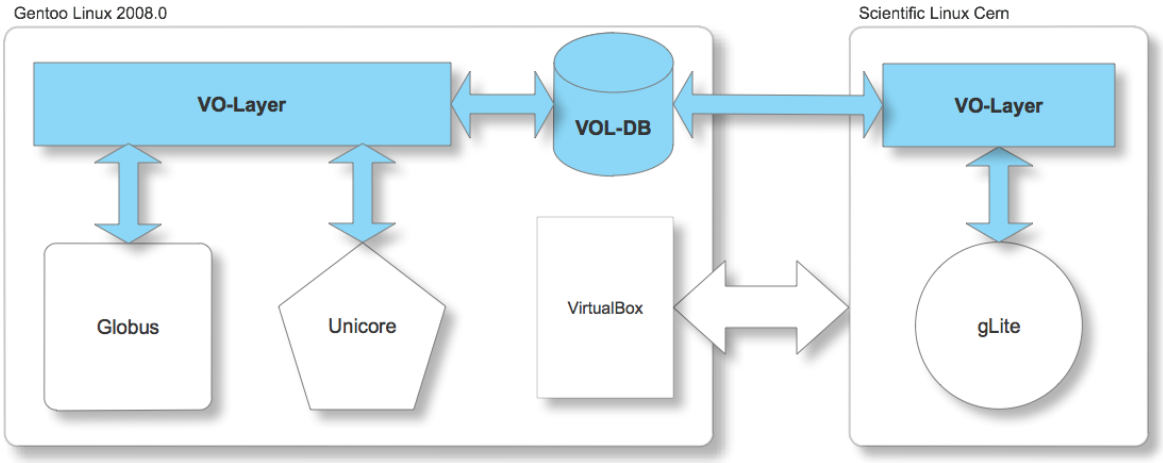



Abbildung 6.1: Testbett in dem der VO-Layer implementiert wurde.

## 6.2 VO-Layer System

In diesem Abschnitt werden die wichtigsten Teile der Konzeptimplementierung des VO-Layer besprochen. Dazu gehören insbesondere die Umsetzung der Authentifizierungs- und Autorisierungsmechanismen. Des Weiteren wird das Management-Subsystem des VO-Layers im Detail erklärt. Abbildung 6.2 zeigt das Objektdiagramm des VO-Layers, mit der Testtreiber Klasse `Test`, welche die Methoden der drei Subsysteme aufruft. Die Views der Subsysteme werden nicht implementiert, sind aber der Vollständigkeit halber ebenfalls in der Abbildung aufgeführt.

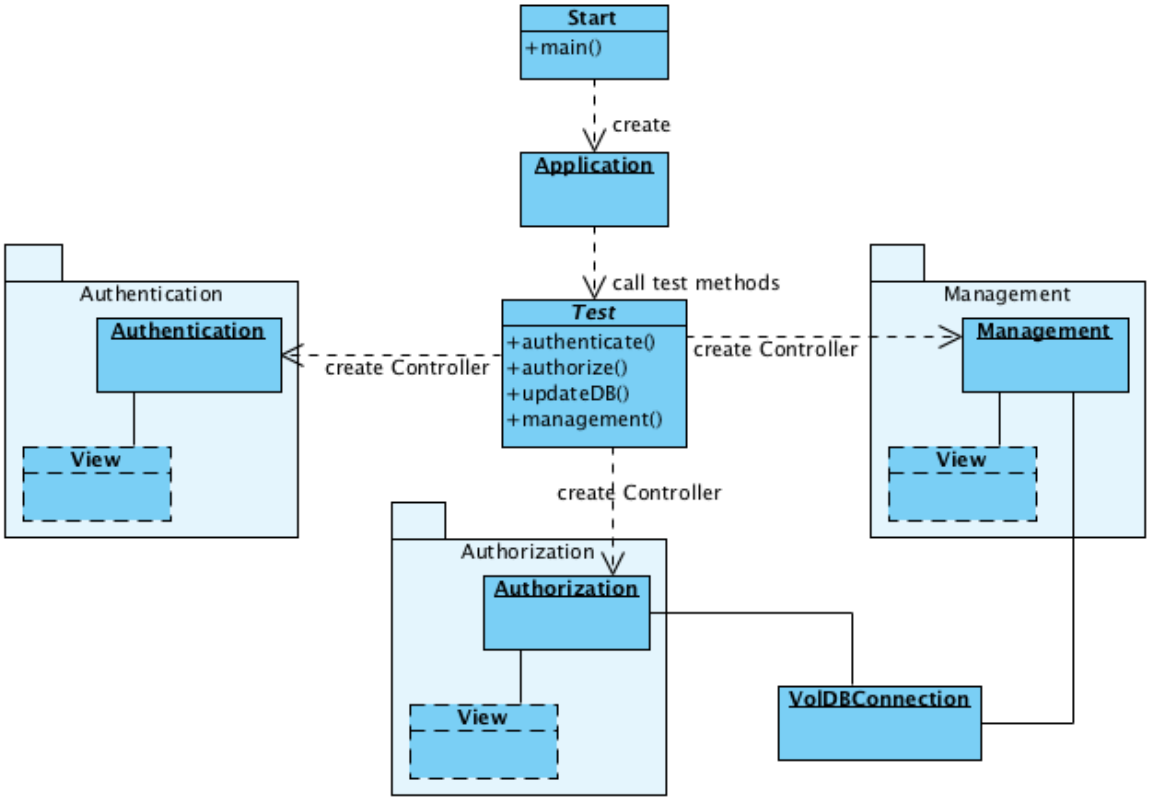



Abbildung 6.2: Das Objektdiagramm des VO-Layer mit der abstrakten Klasse `Test`, welche als Testtreiber für die drei Subsysteme fungiert.

### 6.2.1 Hauptanwendung

Der VO-Layer wird mit der Klasse `Start` ausgeführt. Diese Klasse enthält die statische Methode `main` (siehe Listing 6.1), welche die eigentliche Anwendung startet. Nach der Erstellung eines Objektes der Klasse `Application` werden von diesem Objekt aus die verschiedenen Testmethoden der Klasse `Test` ausgeführt. Diese wiederum erstellen die Controller Objekte der Subsysteme und rufen die entsprechenden Methoden für die Authentifizierung, die Autorisierung oder das Management auf.

```
public static void main(String[] args) {
  Application app = new Application();
  app.start();
  app.stop();
}
```

Listing 6.1: Hauptanwendung starten

### 6.2.2 Authentifizieren

Um einen Benutzer im Grid zu authentifizieren, wird ein Benutzerzertifikat benötigt. Dieses wird der `authenticate` Methode der Klasse `Authentication` übergeben (siehe Listing 6.2). Je nach dazugehöriger Middleware wird nun ein Befehl erstellt, der von der Java Klasse `Runtime` auf der Kommandozeile des Betriebssystems - in diesem Fall der Shell - ausgeführt wird (siehe Listing 6.3). Dieser Befehl startet ein Shell-Skript, welches den eigentlichen Grid-Befehl zur Authentifizierung des Benutzers ausführt.

```
public boolean authenticate(Certificate cert) {
  switch (cert.getMiddleware()) {
  ...
```

Listing 6.2: Authentication.java

```
String cmd = "globus_authenticate.sh "+
    cert.getLocation()+cert.getCert()+
    " "+cert.getLocation()+cert.getKey()+" "+passwd;
CommandExecuter.executeCmd(cmd);
```

Listing 6.3: Shell-Skript ausführen

Es existiert für jede Grid-Middleware ein eigenes Shell-Skript zur Authentifizierung der Benutzer. Für Globus wird das Skript *globus_authenticate.sh* (Listing 6.4) verwendet, für gLite das Skript *glite_authenticate.sh* (Listing 6.5) und für UNICORE das Skript *unicore_authenticate.sh* (Listing 6.6). Im Falle von Globus und gLite wird durch den entsprechenden Grid-Befehl (`grid-proxy-init` für Globus und `voms-proxy-init` für gLite) ein Proxy-Zertifikat erstellt und im Dateisystem für den Benutzer abgelegt. Für UNICORE wird mit dem Befehl `ucc connect` lediglich geprüft, ob das Zertifikat gültig ist und der Benutzer auf die angegebene *registry* zugreifen kann.

```
#!/bin/sh

# script to create a Globus proxy-cert.
# Parameter:
# 1 cert file
# 2 key file
# 3 password

grid-proxy-init -pwstdin -cert $1 -key $2 << EOF
$3
EOF
```

Listing 6.4: globus_authenticate.sh

```
#!/bin/sh

# script to create a gLite proxy-cert.
# Parameter:
# 1 cert file
# 2 key file
# 3 password

voms-proxy-init -pwstdin -cert $1 -key $2 << EOF
$3
EOF
```

Listing 6.5: glite_authenticate.sh

```
#!/bin/sh

# script to check if a UNICORE certificate is valid.
# Parameter:
# 1 key file
# 2 registry
# 3 password (opt.)

if [ $# -eq 3 ]
then
  ucc connect -k $1 -p $3 -r $2
elif [ $# -eq 2 ]
then
  ucc connect -k $1 -r $2
fi
```

Listing 6.6: unicore_authenticate.sh

Die Ausgabe dieser Grid-Befehle wird dem Benutzer des VO-Layers am Terminal angezeigt und im Erfolgsfall erhält er ein Proxy-Zertifikat (für Globus oder gLite) oder die Bestätigung für den Zugang zum UNICORE Grid. Falls die Authentifizierung nicht erfolgreich war, wird im Terminal der entsprechende Fehler angezeigt. Listing 6.7 zeigt die Ausgabe des VO-Layers bei der erfolgreiche Authentifizierung eines Benutzers an einem Globus und einem UNICORE Grid. Die Ausgabe für gLite ist, aufgrund der selben Befehle, gleich wie jene von Globus und wird deshalb hier ausgespart.

```
#########################
### Application start ###
#########################
Running: globus_authenticate.sh usercert.pem userkey.pem
OUTPUT> grid-proxy-init -pwstdin -cert usercert.pem -key userkey.pem
OUTPUT> Your identity: /O=Globus/OU=GT4 Examples/CN=Wolfgang Kirchler
OUTPUT> Creating proxy ..................................... Done
OUTPUT> Your proxy is valid until: Mon Sep  1 23:30:27 2008
Globus authentication success

Running: unicore_authenticate.sh keystore.jks
    https://localhost:8080/DEMO-SITE/services/
      Registry?res=default_registry
OUTPUT> ucc connect --keystore keystore.jks --registry
    https://localhost:8080/DEMO-SITE/services/
      Registry?res=default_registry
OUTPUT> You can access 1 target system(s).
UNICORE authentication success
```

Listing 6.7: Terminalausgabe für die Authentifizierung an einem Grid mit Globus und UNICORE Grid-Middleware.

### 6.2.3 Autorisieren

Die Autorisierung des Benutzers erfolgt immer dann, wenn dieser auf eine Ressource zugreifen möchte. Durch den VO-Layer wird nun, vor dem eigentlichen Ressourcenzugriff über die Grid-Middleware, die Autorisierung des Benutzers anhand der VO-Struktur des VO-Layers überprüft. Dazu wird zunächst durch die Methode `authorization` eine Datenbankabfrage ausgeführt, welche überprüft, ob der Benutzer *user* Zugang zur Ressource *resource* hat (Listing 6.8). Sobald das Ergebnis der Datenbankabfrage vorliegt, wird die Methode `resultReceived` ausgeführt, die das Ergebnis der Autorisierung verarbeitet (Listing 6.9). Ist dieses Ergebnis negativ, wird an dieser Stelle abgebrochen und der Benutzer erhält eine Meldung, dass sein Autorisierungsversuch fehlgeschlagen ist. Ist das Ergebnis positiv, wird anschließend der gewünschte Grid-Job gestartet, wobei dieser - je nach zugrunde liegender Grid Middleware-Technologie - anders ausgeführt werden muss.

```
public void authorize(User user, Resource resource, Certificate cert) {
  ...
  DataManager.authorize(user, resource, this, DB_AUTHORIZE);
}
```

Listing 6.8: Authorization.java

```
public void resultReceived(int idCode, Object result) {
  switch (idCode) {
  ...
  case DB_AUTHORIZE:
    boolean authorize = (Boolean)result;
    if (!authorize) {
      AuthorizationView.printResults("Access on Resource refused");
      return;
    }
    switch (cert.getMiddleware()) {
    case GLOBUS_4:
      String client = "org.globus.examples.clients.
            MathService_instance.Client";
      cmd = "globus_run.sh "+client+" "+
            resource.getHostname()+resource.getName();
      break;

    case UNICORE_6:
      String password = "";
      String jobfile = System.getProperty("user.home")+
          "/ucc-1.1.1/samples/date.u";
      // generate the command to run the script
      // with the specified parameters
      cmd = "unicore_run.sh "+jobfile+" "+
          cert.getLocation()+cert.getCert()+
          " "+((Unicore_Certificate)cert).getRegistry();
      if (password!= "")
        cmd += " "+password;
      break;

    case GLITE_3_1:
      String jdlfile = System.getProperty("user.home")+"jobfile.jdl";
      cmd = "glite_run.sh "+resource.getHostname()+resource.getName()+
          " "+jdlfile;
      break;

    }
  }
}
```

Listing 6.9: Autorisierungs-Ergebnis des VO-Layers

Für jede Grid Middleware-Technologie existiert ein eigenes Skript, welches den angegebenen Job mit dem entsprechenden Grid-Befehl ausführt. Für Globus-Jobs existiert das Skript *globus_run.sh* (Listing 6.10), welches einen Web-Service Client mit der Web-Service URL des Servers aufruft. Dazu muss der Web-Service Client mit der Java Runtime gestartet werden, als Argument erhält der Client die URL des Server (Beispiele zu dieser Art von Globus Jobs finden sich im Buch [SoCh 05]).

```
#!/bin/sh

# script to run a Globus web service client
# Parameter:
# 1 WebService Client
# 2 WebService URL

# set the classpath for the globus environment
source $GLOBUS_LOCATION/etc/globus-devel-env.sh

cd $HOME/globus/MathService
echo "java -classpath .build/stubs/classes/:\$CLASSPATH $1 $2"

# run the web-service client
java -classpath .build/stubs/classes/:$CLASSPATH $1 $2
```

Listing 6.10: globus_run.sh

Damit Jobs auf Ressourcen ausgeführt werden können, welche von der gLite Middleware-Technologie bereitgestellt werden, existiert das Skript *glite_run.sh* (Listing 6.11). Dieses Skript führt ein Job-File auf der gLite Ressource mittels des Befehls `globus-job-run` aus. Dieser Befehl `globus-job-run` ist ein Zeichen für die enge Verwandtschaft der beiden Middleware-Technologien Globus Toolkit und gLite.

```
#!/bin/sh

# script to run a job on a gLite site
# uses the 'globus-job-submit' command glite-job-submit is not tested
# Parameter:
# 1 contact string
# 2 executable

echo "globus-job-run $1 $2"
globus-job-run $1 $2
```

Listing 6.11: glite_run.sh

UNICORE Jobs werden mit Hilfe des Skriptes *unicore_run.sh* (Listing 6.12) ausgeführt. Dazu wird mit Hilfe des UNICORE Commandline Clients der Befehl `ucc run` ausgeführt, welcher einen Job startet, der durch ein Job-File definiert ist. Als weitere Parameter werden die Informationen zum Benutzerzertifikat übergeben, da die Authentizität des Benutzer von UNICORE bei jedem Zugriff überprüft wird.

Der VO-Layer zeigt die Ausgabe der Grid-Jobs im Terminal an. Je nach Job wird das Ergebnis sofort nach der Ausführung desselben angezeigt oder muss in einem weiteren Schritt von einem Web-Service abgefragt werden.

```
#!/bin/sh

# script to run a UNICORE job from a jobfile
# Parameter:
# 1 job file
# 2 key file
# 3 registry
# 4 password (opt.)

# run the unicore job
if [ $# -eq 4 ]
then
  echo "ucc run -k $2 -p $4 -r $3 -v $1"
  ucc run -k $2 -p $4 -r $3 -v $1
elif [ $# -eq 3 ]
then
  echo "ucc run -k $2 -r $3 -v $1"
  ucc run -k $2 -r $3 -v $1
fi
```

Listing 6.12: unicore_run.sh

In Listing 6.13 wird die Terminal Ausgabe für eine Jobausführung auf einer Globus Ressource und auf einer UNICORE Ressource gezeigt. Zunächst wird ein Globus-Job mit Hilfe des Skriptes *globus_run.sh* gestartet. Dieses startet einen Web-Service Client, welcher auf einen primitiven Web-Service zugreift und eine Addition auf einer Variablen durchführt. Dieser Grid-Service wurde aus dem Buch [SoCh 05] entnommen. Anschließend wird ein UNICORE Job, der im Job-File *date.u* definiert ist, mit Hilfe des Skriptes *unicore_run.sh* gestartet. Listing 6.13 zeigt die ausführliche Ausgabe für diese Jobausführung an. Der Start eines gLite Jobs läuft fast genauso ab wie der Start eines Globus Jobs, deshalb wird hier auf ein weiteres Beispiel zu gLite verzichtet.

```
#########################
### Application start ###
#########################
Running: globus_run.sh
    org.globus.examples.clients.MathService_instance.Client
    http://127.0.0.1:8080/wsrf/services/examples/core/first/MathService
OUTPUT> java -classpath .build/stubs/classes/:$CLASSPATH
    org.globus.examples.clients.MathService_instance.Client
    http://127.0.0.1:8080/wsrf/services/examples/core/first/MathService
OUTPUT> Current value: 15
OUTPUT> Current value: 10

Running: unicore_run.sh date.u keystore.jks
    https://localhost:8080/DEMO-SITE/services/
      Registry?res=default_registry
OUTPUT> ucc run -k keystore.jks -r
```

```
    https://localhost:8080/DEMO-SITE/services/
      Registry?res=default_registry -v date.u
OUTPUT> [ucc run] Verbose mode.
OUTPUT> [ucc run] Reading properties file <.ucc/preferences>
OUTPUT> [ucc run] Keystore = keystore.jks
OUTPUT> [ucc run] Keystore password given.
OUTPUT> [ucc run] Using default alias.
OUTPUT> [ucc run] Keystore type = jks
OUTPUT> [ucc run] Registry = https://localhost:8080/DEMO-SITE/services/
    Registry?res=default_registry
OUTPUT> [ucc run] Pinging registry.
OUTPUT> [ucc run] Registry PING OK,
    server reply: servertime=2008-09-01T11:41:31.773+02:00
OUTPUT> [ucc run] Output goes to <.>
OUTPUT> [ucc run] Output goes to <.>
OUTPUT> [ucc run] Synchronous processing = true
OUTPUT> [ucc run] Adding job id to output file names = true
OUTPUT> [ucc run] Assuming JSDL job file = false
OUTPUT> [ucc run] Read job from <date.u>
OUTPUT> [ucc run] Selected TSS at
    https://localhost:8080/DEMO-SITE/services/
    TargetSystemService?res=ad7bbc2c-38da-474e-ae28-d002fccea3a5
OUTPUT> [ucc run] Job submitted,
    job url=https://localhost:8080/DEMO-SITE/services/
    JobManagement?res=3cd21e15-7c14-41a1-9101-8106aa1218f7
OUTPUT> [ucc run] Job started.
OUTPUT> SUCCESSFUL exit code: 0
OUTPUT> [ucc run] Exporting USpace file 'stdout' to
    '3cd21e15-7c14-41a1-9101-8106aa1218f7.stdout'
OUTPUT> 3cd21e15-7c14-41a1-9101-8106aa1218f7.stdout
OUTPUT> [ucc run] Exporting USpace file 'stderr' to
    '3cd21e15-7c14-41a1-9101-8106aa1218f7.stderr'
OUTPUT> 3cd21e15-7c14-41a1-9101-8106aa1218f7.stderr
OUTPUT> 3cd21e15-7c14-41a1-9101-8106aa1218f7.properties
```

Listing 6.13: Terminalausgabe für den Start eines Globus und eines UNICORE Jobs auf einer Grid-Ressource.

### 6.2.4 Management

Mit Hilfe des Management-Subsystems kann die VO-Struktur angepasst, neue VOs, Benutzer und Ressourcen erstellt und in die bestehende Struktur integriert werden. Des Weiteren können den bestehenden Benutzer und Ressourcen neue FQANs zugeteilt oder vorhandene entzogen werden. Die Aufgabe des VO-Layers an dieser Stelle ist die gewünschten Änderungen in der VO-Layer DB festzuschreiben und die geänderten Zugriffsberechtigungen an die Middleware-Technologien weiterzuleiten.

Listing 6.14 zeigt zwei ausgewählte Methoden der Klasse `VOManagement`, welche dazu verwendet werden können, um die VO-Struktur des VO-Layer zu verändern. Alle diese Methoden

setzten einen Datenbankbefehl ab, der entweder einen neuen Datensatz in die Datenbank aufnimmt oder einen bestehenden aus dieser löscht.

```
public void createVO(VO vo) {
  DataManager.create_vo(vo, this, DB_CREATE_VO);
}
public void update_add(User user, VO vo) {
  DataManager.add_user_to_vo(user, vo, this, DB_ADD_USER_VO);
}
```

Listing 6.14: Methoden aus VOManagement.java

Die Änderungen, welche an der VO-Struktur des VO-Layers vorgenommen werden, müssen auch den darunter liegenden Grid Middleware-Technologien mitgeteilt werden. Dazu gibt es zwei Lösungswege, welche zunächst vorgestellt werden. Anschließend werden die Lösungen bewertet:

**Sofortige Änderung** nach der Aktualisierung der VO-Layer DB. Dazu werden nach jeder Änderung der VO-Struktur die neuen Zugriffsbedingungen an die Middleware-Technologien weitergeleitet.

Die Positiven Aspekte sind, dass zum einen die Änderungen der Zugriffsbedingungen sofort wirksam werden und zum anderen nur Änderungen übertragen werden. Somit verteilt sich der Aufwand für die Aktualisierung gleichmäßiger.

Auf der anderen Seite müssen trotzdem Methoden zur Übertragung der gesamten VO-Struktur an die Middleware-Technologien - für den Fall eines Datenverlustes - existieren. Die Änderungen müssen in bestehende gridmap files und UUDBs integriert werden. In UNICORE existieren Befehle für die Änderung der UUDB, allerdings gibt es in Globus und gLite keine Methoden zur Änderung der gridmap files. Diese Methoden müssen somit auch entwickelt werden. Des Weiteren erfordern nicht alle Änderungen an der VO-Struktur Änderungen in den Middleware ACLs. Deshalb müssen die relevanten Änderungen erst ermittelt werden. Insgesamt steigt der Programmieraufwand für diese Lösung erheblich, da stets eine direkte Verbindung zu den Middleware-Technologien benötigt wird, und zur Aktualisierung der gridmap files auch noch neue Methoden entwickelt werden müssen.

**Periodische Aktualisierung** der Zugriffsbedingungen in den Middleware-Technologien. Hierbei wird, stets nach einer festgelegten Zeitspanne (oder nach explizitem Aufruf), die gesamte VO-Struktur in die ACLs der Grid Middleware-Technologien übersetzt.

Dieser Lösungsvorschlag hat den Vorteil, dass Änderungen leicht auf neue Systeme übertragen werden können. Des Weiteren ist die Erstellung der ACLs (gridmap files und UUDBs), aus der VO-Struktur sehr einfach und die ACLs können jederzeit - etwa nach Datenverlust - komplett wiederhergestellt werden. Es wird keine direkte Verbindung zu den Middleware-Technologien benötigt, sondern die Aktualisierung der ACLs erfolgt über Dateitransfer.

Als negativ anzusehen ist, dass die Änderungen der Zugriffsbedingungen mitunter erst nach der Aktualisierung der Middleware ACLs wirksam werden. Des Weiteren müssen

die gridmap files und UUDBs auf den jeweiligen Host übertragen werden. Die Übertragungsmethoden hierfür müssen auch entwickelt werden.

Da bei der zweiten Lösung die positiven Aspekte überwiegen und diese auch einfacher in die Testumgebung integriert werden kann, wird für diese Diplomarbeit dieser Ansatz weiter verfolgt. Für jede der drei Middleware-Technologien wird eine Methode erstellt, welche aus der VO-Struktur die entsprechenden ACLs erstellt. In Listing 6.15 sind die öffentlichen Methoden aufgelistet, die das Generieren der ACLs initiieren. Zunächst wird dabei eine Datenbankabfrage gestartet, welche für jeden Host - auf dem die entsprechenden Grid-Middleware läuft - die benötigten Informationen aus der Datenbank holt und anschließend in eine entsprechende Datenstruktur (siehe Abbildung 5.12) speichert.

```
public void generate_globus_gridmap() {
  DataManager.get_all_globus_gridmap(this, DB_GET_GLOBUS_GRIDMAP);
}
public void generate_unicore_uudb() {
  DataManager.get_all_unicore_uudb(this, DB_GET_UNICORE_UUDB);
}
public void generate_glite_gridmap() {
  DataManager.get_all_glite_gridmap(this, DB_GET_GLITE_GRIDMAP);
}
```

Listing 6.15: Aufruf zur Erstellung der ACLs für die Middleware-Technologien.

Obwohl die Grid-Middleware gLite bereits VOMS integriert und somit über ein VO-Konzept verfügt, aktualisiert der VO-Layer die VOMS-Daten nicht, sondern schreibt die geänderten Autorisierungsinformationen in einem gridmap file fest. Die Entscheidung für die Aktualisierung der gridmap files und nicht der VOMS-Datenstruktur hat einige Gründe, welche im Folgenden dargelegt werden:

1. Wenn neben der VO-Layer Datenbank auch die VOMS-Datenbank aktualisiert werden muss, können leicht Inkonsistenzen zwischen den beiden Datenstrukturen entstehen. Die Pflege der zwei VO-Strukturen wird erschwert.
2. gLite verteilt die Autorisierungsinformationen aus der zentralen VOMS-Datenbank auf die einzelnen Hostrechner ebenfalls über gridmap files. Aus diesem Grund ist es einfacher, den Umweg über VOMS einzusparen und stattdessen direkt aus der VO-Layer Datenbank die gridmap files für die gLite Hostrechner zu erstellen.
3. Die Erstellung und die Pflege der gridmap files ist wesentlich einfacher als die Pflege einer VOMS-Datenbank. Zudem werden die gridmap files ständig komplett neu erzeugt und können, bei einem Datenverlust neu erstellt werden. Die Rekonstruktion der VOMS-Datenbank hingegen ist um einiges schwieriger.
4. Für die Autorisierung wird mit dem Einsatz des VO-Layers nur noch dessen VO-Struktur verwendet. Die Autorisierungskonzepte der darunter liegenden Grid Middleware-Technologien sind lediglich eine Absicherung dazu. Somit wäre VOMS nur eine (vereinfachte) Abbildung der VO-Struktur des VO-Layers und bietet deshalb keinen Mehrwert für das System.

5. Der direkte Zugriff auf die VOMS-Datenbank ist vom VO-Layer aus nicht so einfach möglich. Der VOMS-Administrator muss dazu direkt an dem Hostrechner mit dem VOMS-Server angemeldet sein. Die automatische Aktualisierung der VOMS-Datenbank ist, von einem anderen Rechner aus, nicht ohne weiteres möglich.

Alle diese Gründe haben zu der Entscheidung geführt, die Autorisierungsinformationen für die gLite Grid-Middleware direkt in die gridmap files der entsprechenden Hostrechner zu schreiben. Somit wird der Einsatz eines VOMS-Servers nach der Umstellung auf den VO-Layer obsolet.

Nachdem die Datenbankabfrage erfolgreich ausgeführt wurde, werden - aus der soeben erstellten Datenstruktur - die neuen ACLs für die Middleware-Technologien in Form von gridmap files (für Globus und gLite) und UUDBs (für UNICORE) mit Hilfe der Methoden `generate_gridmap_file` (Listing 6.16) und `generate_uudb_file` (Listing 6.17) erstellt.

Das gridmap file für Globus und gLite erhält dabei als Dateinamen den Hostnamen des Zielsystems, auf dem es importiert werden muss. Die Datei selbst besteht aus mehreren Zeilen, wobei jede Zeile das Tupel (DN, Benutzername) - durch Leerzeichen getrennt - enthält. Ein Beispiel für eine solche Zeile wäre:

"C=DE/O=GridGermany/OU=TU München/CN=Wolfgang Kirchler" kirchler

Das erstellte gridmap file muss daraufhin auf dem Zielsystem gespeichert werden. Beim Globus Toolkit wird das gridmap file standardmäßig unter `$GLOBUS_LOCATION/etc/gridmapfile` gespeichert, wobei `$GLOBUS_LOCATION` der Installationspfad von Globus ist (standardmäßig `/usr/local/gt4`). Für gLite muss das gridmap file unter `/etc/gridmap-security/gridmap` gespeichert werden.

```
public void generate_gridmap_file() {
  File file = new File(Application.globusPath+"/"+
      hostname.replace("http://", ""));
  try {
    BufferedWriter bw = new BufferedWriter(new FileWriter(file));
    for (User user : user_s) {
      bw.write(user.getDn()+" "+user.getName());
      bw.newLine();
    }
    bw.flush();
    bw.close();
  } catch (IOException e) {
    e.printStackTrace();
  }
}
```

Listing 6.16: Methode zum Erstellen der Globus gridmap files.

Für UNICORE muss eine Datei im Comma-Separated-Value (CVS) Format gespeichert werden. Die Datei erhält - wie die gridmap files - als Dateinamen den Hostnamen des Zielsystems

und besteht selbst wiederum aus mehreren Zeilen, wobei die Einträge in den Zeilen durch Semikolons voneinander getrennt werden. Als erste Zeile wird eine Spezifizierung der Spalteneinträge erwartet. Somit sieht eine UUDB-Datei mit einem Eintrag wie folgt aus:

Nr.;GcID;Xlogin;Role;Projects;CertInPEM
1;http://127.0.0.1:8080/DEMO-SITE;kirchler;Admin;;*Public-Key im PEM-Format*

```
public void generate_uudb_file() {
  File file = new File(Application.unicorePath+"/"+
      hostname.replace("http://", ""));
  try {
    BufferedWriter bw = new BufferedWriter(new FileWriter(file));
    bw.write("Nr.;GcID;Xlogin;Role;Projects;CertInPEM");
    bw.newLine();
    int i = 1;
    for (UUDB_Entry uudb_entry : uudb_entry_s) {
      bw.write(i+";"+uudb_entry.getGcid()+";"+
          uudb_entry.getXlogin()+";"+
          uudb_entry.getRole()+";"+uudb_entry.getProject()+
          ";"+uudb_entry.getCertInPEM());
      bw.newLine();
      i++;
    }
    bw.flush();
    bw.close();
  } catch (IOException e) {
    e.printStackTrace();
  }
}
```

Listing 6.17: Methode zur Erstellung einer CVS-Datei welche in UNICORE als UUDB importiert werden kann.

Die erstellten gridmap files und UUDBs werden im Dateisystem gespeichert und können von dort auf den jeweiligen Hostrechner kopiert und importiert werden. Bevor die neue UUDB importiert werden kann, muss die alte vom Zielsystem gelöscht werden. Dazu wird das Skript *admin.sh* verwendet, welches mit den UNICORE Server mitgeliefert wird. Zum Importieren einer UUDB auf einem Zielsystem müssen folgende Befehle ausgeführt werden:

```
# ./admin.sh remove ALL
# ./admin.sh import uudb.csv
```

Für jede der drei Grid-Middleware-Technologien gibt es einen eigenen Ordner in dem die gridmap files oder UUDBs abgelegt werden (diese Ordnerstruktur ist in Abbildung 5.3 abgebildet). Der Vorgang des Kopierens dieser Dateien auf den jeweiligen Hostrechner und das Importieren der Dateien in der Grid-Middleware wird in diesem Testsystem nicht implementiert.

## 6.3 Build und Deployment

In diesem Abschnitt werden alle nötigen Schritte erklärt, die erforderlich sind, um den VO-Layer auf einem System zu installieren und zu konfigurieren. Das VO-Layer System arbeitet dabei wie folgt: Der VO-Layer läuft als Benutzerprogramm auf allen Rechnern, die Zugriff zum Grid haben. Diese Rechner greifen auf die zentrale VO-Layer Datenbank (VOL-DB) zu und leiten die Benutzeranfragen an die entsprechende Grid Middleware-Technologie - und somit an die darunter liegenden Ressourcen - weiter. Abbildung 6.3 zeigt, wie der Zugriff auf Ressourcen in einem Grid, über verschiedene Client-Rechner und dem darauf installierten VO-Layer, abläuft.

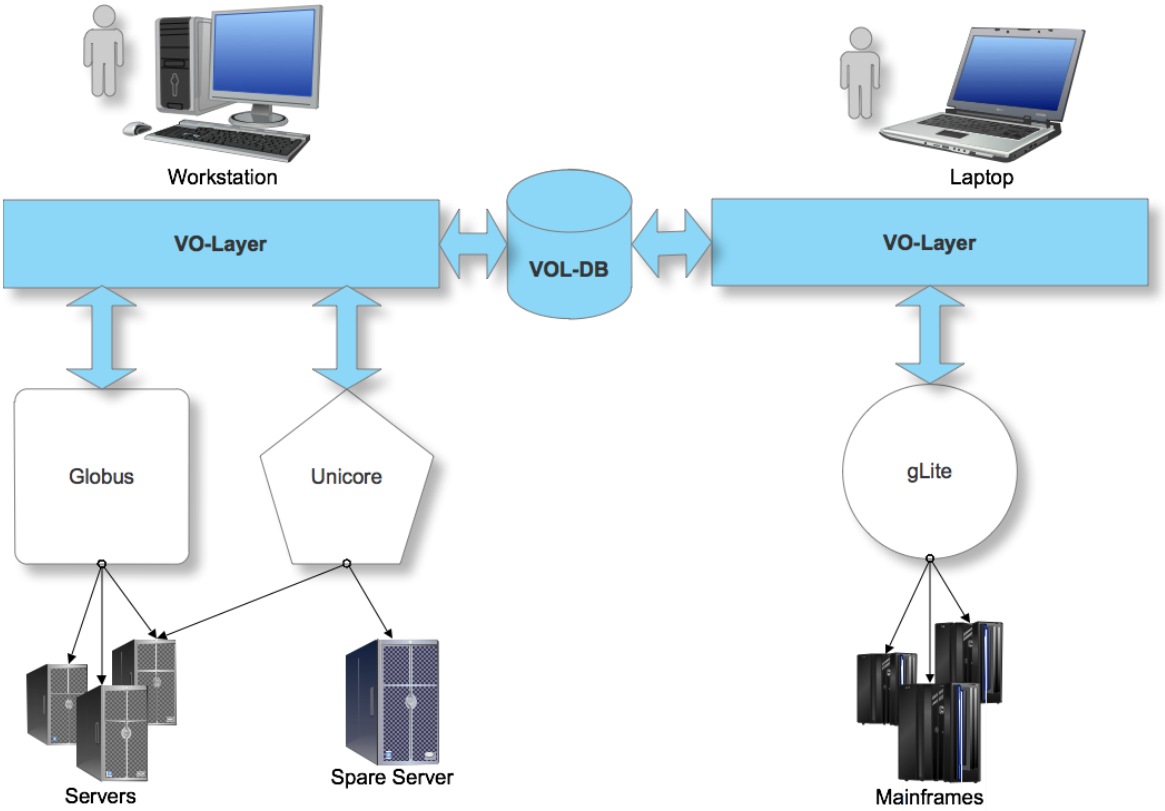



Abbildung 6.3: Beispiel einer Grid Umgebung mit einem Benutzer, der auf unterschiedlichen Rechnern, über den VO-Layer, auf verschiedene Ressourcen zugreift.

Der VO-Layer unterstützt ausschließlich das Linux Betriebssystem und benötigt eine Java Virtual Machine (JVM) der Version 1.5 oder höher. Der Server welcher die VO-Layer Datenbank beherbergt, muss ein Datenbanksystem vom Typ MySql bereitstellen. Die Datenbanktabellen müssen Fremdschlüssel und Transaktionen unterstützen. Aus diesem Grund muss MySql die Datenbanktabellen im Format *InnoDB*[4] ablegen. Nur dieses Tabellenformat bietet in MySql die nötige Unterstützung für Fremdschlüssel und Transaktionen.

[4]http://www.innodb.com/

Jedes System das den VO-Layer unterstützt, wird mindestens eine der drei Grid Middleware-Technologien einsetzen. Zur Installation der Grid Middleware-Technologien siehe Anhang A. Jeder Benutzer, der den VO-Layer benutzen möchte, benötigt einen eigenen Benutzer-Account auf dem entsprechenden System. Im Home-Verzeichnis des Benutzers muss die Ordnerstruktur aus Abbildung 5.3 angelegt werden. Innerhalb des *cert* Verzeichnisses muss für jede Grid-Middleware - die von diesem System aus erreichbar ist - ein eigener Ordner angelegt werden. In diesem Ordner wird das Benutzerzertifikat und der Private-Key gespeichert, der für den Zugang zur jeweiligen Grid-Middleware benötigt wird.

Die Skripte, welche zur Kommunikation zwischen VO-Layer und Grid-Middleware bei der Authentifizierung und Autorisierung verwendet werden, müssen auf jedem System gespeichert werden und im *PATH* enthalten sein. Des Weiteren müssen auch die Grid-Befehle welche von diesen Shell-Skripten benutzt werden im *PATH* enthalten sein.

Bevor der VO-Layer gestartet werden kann, muss die VO-Layer Datenbank erstellt werden. Dazu muss auf dem Datenbankserver, der die VO-Layer Datenbank beherbergt, eine neue Datenbank mit dem Namen *VOL-DB* erstellt werden. In dieser Datenbank müssen die Tabellen aus Abbildung 5.4 angelegt werden.

Da der VO-Layer in dieser PoC-Implementierung kein Benutzer-Interface besitzt muss für jede Aktion die der VO-Layer ausführen soll, die Methode `Test` angepasst werden. Nach den Anpassungen am Quellcode kann das Ant-Skript *build.xml* verwendet werden um aus den Quelldateien eine JAR-Datei zu erstellen. Die wichtigsten Teile des Ant-Skriptes - das Kompilieren und das Erstellen der JAR-Datei - sind in Listing 6.18 dargestellt. Sobald das System eingerichtet ist und die JAR-Datei erstellt wurde, kann der VO-Layer mit dem Befehl `java -jar vo-layer.jar` ausgeführt werden.

```
<!-- compile the java files -->
<target name="compile" depends="clean">
  <mkdir dir="${bin.dir}"/>
  <javac srcdir="${src.java.dir}" destdir="${bin.dir}"
      debug="yes" nowarn="true">
    <classpath refid="ExternalJar.classpath"/>
  </javac>
</target>

<!-- generate a jar-file -->
<target name="makejar" depends="compile">
  <jar destfile="vo-layer.jar" basedir="${bin.dir}">
    <manifest>
      <attribute name="Main-Class" value="vo_layer/Start"/>
      <attribute name="Class-Path" value="${manifest.jars}"/>
    </manifest>
  </jar>
</target>
```

Listing 6.18: Ant-Skript zum Kompilieren der Java Dateien und Erstellen einer JAR-Datei (build.xml).

Sobald der VO-Layer in einem Grid verwendet wird, können jene Ressourcen, welche über eine bestimmte Grid Middleware-Technologie bereitgestellt werden, nicht mehr auf den üblichen Weg angefordert werden. Von diesem Zeitpunkt an, werden nämlich die gridmap files oder UUDBs in den Middleware-Technologien überschrieben und durch jene ersetzt, die vom VO-Layer generiert werden. Aus diesem Grund gibt es nach der Entscheidung für den VO-Layer und dessen erstmaligem Einsatz, kein Zurück mehr zu den alten Autorisierungsverfahren.

## 6.4 Zusammenfassung

Dieses Kapitel hat die wichtigsten Details zur Implementierung des VO-Layers behandelt. Zunächst wurde die Testumgebung vorgestellt, in der der VO-Layer entwickelt wurde. Zu dieser Testumgebung gehören, neben den Grid Middleware-Technologien Globus Toolkit, UNCORE und gLite auch noch diverse Softwarepakte welche für die Entwicklung des VO-Layer benötigt wurden.

Das VO-Layer System wurde anschließend in den wichtigsten Bereichen vorgestellt. Abbildung 6.2 zeigt das Objektdiagramm des VO-Layer mit dem verwendeten Testtreiber. Anschließend wurden die einzelnen Subsysteme für die Authentifizierung, die Autorisierung und das Management betrachtet. Bei der Authentifizierung wird ein externes Shell-Skript aufgerufen, welches mit dem angegebenen Benutzerzertifikat einen Grid-Befehl ausführt, der wiederum das Zertifikat überprüft und gegebenenfalls ein Proxy-Zertifikat erstellt.

Die Autorisierung der Benutzer findet komplett im VO-Layer statt. Dieser überprüft die Zugriffsberechtigung des Benutzers auf die angegebene Ressource und startet - im Erfolgsfall - ein externes Shell-Skript, welches den angegebenen Grid-Job an die entsprechende Middleware weiterleitet und dort zur Ausführung bringt.

Das Management der VO-Struktur läuft ausschließlich über Methoden des VO-Layer, welche die VO-Struktur in gewünschter Weise verändern. Damit die Änderungen auch wirksam werden, werden aus der VO-Struktur regelmäßig die Zugriffskontrolllisten für die Hostrechner, auf denen sich die Ressourcen befinden, erstellt. Diese ACLs in Form von gridmap files und UUDBs müssen anschließend auf den Hostrechnern importiert werden.

Der letzte Teil dieses Kapitels beschäftigte sich mit dem Deployment des VO-Layers in einer neuen Umgebung. Dazu müssen einige Konfigurationen vorgenommen werden. Des Weiteren wurde beschrieben, wie der VO-Layer kompiliert und daraus eine JAR-Datei erstellt werden kann.

Im nächsten Kapitel werden die Ergebnisse dieser Arbeit, anhand der in Kapitel 3 identifizierten Anforderungen bewertet. Dabei werden zunächst die funktionalen und anschließend die nicht-funktionalen Anforderungen untersucht.

# 7 Bewertung

Dieses Kapitel untersucht die in dieser Diplomarbeit erstellte Lösung einer einheitlichen Authentifizierungs- und Autorisierungs-Schnittstelle für heterogene Grids. In Kapitel 3 sind sämtliche funktionalen und nicht-funktionalen Anforderungen an eine einheitliche AAI festgeschrieben. Das in Kapitel 5 entworfene und in Kapitel 6 prototypisch implementierte System, wird nun anhand dieser Anforderungen bewertet.

## 7.1 Bewertung der funktionalen Anforderungen

Ziel dieser Arbeit ist die Entwicklung einer einheitlichen Authentifizierungs- und Autorisierungsschnittstelle für heterogen Grids, wobei die in Kapitel 3 identifizierten funktionalen Anforderungen die wichtigsten Ziele beschreiben. In den folgenden Abschnitten wird untersucht, inwieweit diese Anforderungen durch den VO-Layer erfüllt werden.

### 7.1.1 Authentifizierung

Der VO-Layer fungiert beim Use Case *Authentifizieren* als Proxy zwischen dem Benutzer und der eigentlichen Grid-Middleware. Der Benutzer muss für die Authentifizierung sein Zertifikat angeben, anschließend wird der VO-Layer aktiv und übermittelt die Zertifikatinformationen mittels eines Grid-Befehls an die entsprechende Grid-Middleware. Dort erfolgt der eigentliche Authentifizierungsvorgang und bei positiver Authentifizierung erhält der Benutzer ein Proxy-Zertifikat, welches ihn ermächtigt, die von der Grid-Middleware bereitgestellten Ressourcen zu benutzen.

Die Authentifizierung ist somit mittels des VO-Layer einheitlich möglich. Der VO-Layer bietet dem Benutzer eindeutige Schnittstellen zur Authentifizierung und leitet die Authentifizierungsanfragen der Benutzer an die entsprechende Grid Middleware-Technologie weiter. Der Benutzer muss deshalb nichts über die darunter liegende Grid-Middleware wissen.

### 7.1.2 Autorisierung

Auch beim Use Case *Autorisieren* fungiert der VO-Layer als Proxy zwischen Benutzer und der eigentlichen Grid-Middleware. Der Benutzer möchte auf einer Ressource des Grids einen Job ausführen, benötigt dazu aber die Zugriffsrechte auf die Ressource. Zunächst überprüft der VO-Layer, ob der Benutzer überhaupt berechtigt ist auf die gewünschte Ressource zuzugreifen. Dazu wird zunächst geprüft, ob sich beide in der selben VO befinden. Falls dies

zutrifft, wird überprüft, ob der Benutzer die nötigen Zugriffsrechte anhand der spezifizierten Gruppen, Rollen und Fähigkeiten besitzt. Dazu prüft der VO-Layer, ob der Benutzer mindestens einen gleichen FQAN wie die angeforderte Ressource besitzt. Ist dies der Fall, darf der Benutzer den Job auf der Ressource ausführen, ansonsten wird der Zugriff auf die Ressource bereits durch den VO-Layer unterbunden. Wird der Zugriff genehmigt, startet der VO-Layer den eigentlichen Job auf der entsprechenden Grid-Middleware. Dazu wird - durch einen Grid-Befehl - der Job an das Zielsystem übergeben. Die Grid-Middleware autorisiert den Benutzer erneut - wobei diese Autorisierung stets erfolgreich verlaufen muss. Daraufhin wird der Job zur Ausführung gebracht und der Benutzer kann im Anschluss das Ergebnis des Jobs abrufen.

Wie die Authentifizierung ist auch die Autorisierung mittels des VO-Layers über einheitliche Schnittstellen möglich. Der Benutzer muss nur diese Schnittstellen kennen und die richtigen Parameter für die Aufrufe verwenden. Der VO-Layer kapselt ihn von der darunter liegenden Grid Middleware-Technologie ab, indem er jede Autorisierungsanfrage automatisch an die richtige Grid-Middleware weiterleitet.

Das Ziel, alle im D-Grid verwendeten VO-Konzept mit Hilfe einer generischen VO-Struktur zu unterstützen wird durch die in Kapitel 5 eingeführte VO-Struktur erreicht. Mit dieser Struktur ist es möglich, VOs mit beliebig vielen Benutzern, Ressourcen und sub-VOs zu etablieren. Benutzer und Ressourcen können ihrerseits in beliebig vielen VOs vertreten sein. Da viele D-Grid Communities das Konzept der Gruppen, Rollen und Fähigkeiten in verschiedenen Ausführungen umsetzen (nur Gruppen, nur Rollen oder Kombinationen aus den dreien), werden zusätzlich zur Entität *VO* auch noch die Entitäten *Gruppe*, *Rolle* und *Fähigkeit* eingeführt. Aus diesen drei Entitäten können beliebige Tripel (Gruppe, Rolle, Fähigkeit) generiert werden. Diese so genannten FQANs geben für einen Benutzer an, welche Zugriffsberechtigungen er in einer VO hat. Für eine Ressource legen sie fest, welche Zugriffsbedingungen für den Zugang nötig sind. Da nicht alle D-Grid Communities alle drei Entitäten unterstützen, kann eines oder mehrere der drei den Wert *NULL* annehmen. Dadurch wird eine generische VO-Struktur geschaffen, welche von den D-Grid Communities - unabhängig von deren VO-Konzept - einheitlich verwendet werden kann.

### 7.1.3 Management

Neben der Authentifizierung und Autorisierung der Benutzer ist ein einheitliches und einfaches Management der VO-Struktur eine wichtige Anforderung an den VO-Layer. Das Management der VO-Struktur teilt sich dabei in zwei Bereiche auf. Zum einen muss die VO-Layer Datenbank aktualisiert werden. Dazu werden die Daten, welche der VO-Administrator hinzufügt, ändert oder löscht, in die VO-Layer Datenbank eingetragen. Zum anderen müssen die Änderungen an der VO-Struktur an die darunter liegenden Grid Middleware-Technologien weitergeleitet werden, damit auch diese die neuen Zugriffsbedingungen kennen. Diese Weiterleitung erfolgt periodisch, wobei aus der VO-Struktur für jeden Hostrechner - auf dem sich eine Ressource befindet - eine neue ACL in Form eines gridmap file (für Globus und gLite Ressourcen) oder einer UUDB (für UNICORE Ressourcen) generiert wird. Diese Dateien werden anschließend auf die jeweiligen Hostrechner übertragen.

## 7.2 Bewertung der nicht-funktionalen Anforderungen

Neben den funktionalen Anforderungen welche alle im Prototypen, der in Kapitel 6 vorgestellt wurde, implementiert werden, sind in Kapitel 3 auch eine Reihe von nicht-funktionalen Anforderungen identifiziert worden. Diese wurden bei der Erstellung des Prototypen nicht explizit berücksichtigt, allerdings wurden Überlegungen angestrengt, wie diese nicht-funktionalen Anforderungen bei der Überführung des Prototypen in ein System für den produktiven Einsatz erfüllt werden können.

**Benutzerfreundlichkeit** Zu einem benutzerfreundlichen System gehört eine ansprechende und leicht zu bedienende GUI, sowie einheitliche und gut dokumentierte Schnittstellen für die Entwickler. Letzteres wird im Rahme dieser Diplomarbeit umgesetzt und kann bei der Entwicklung einer GUI, welche im Rahmen dieser Arbeit nicht entwickelt wurde, eingesetzt werden.

**Zuverlässigkeit** Der VO-Layer liefert stetige Informationen zum Erfolg oder Misserfolg von bestimmten Aktionen. Zum einen werden die Fehler- beziehungsweise Erfolgsmeldungen der Grid-Befehle angezeigt und zum anderen werden die Änderungen in der Datenbank erst festgeschrieben, wenn alle voneinander abhängigen Aktionen ausgeführt wurden. Die VO-Layer Datenbank implementiert bereits im Prototypen Transaktionen, welche durch das Tabellenformat *InnoDB* zur Verfügung gestellt werden.

**Sicherheit** In der Konzeptimplementierung wird die Sicherheit des Systems nicht berücksichtigt. Des Weiteren wird beim Management-Subsystem nicht zwischen normalen Benutzern und VO-Administratoren unterschieden. Bei einem späteren Einsatz muss auf die Sicherheit großer Wert gelegt werden. Die VO-Administratoren können durch bestimmte Rollen identifiziert werden und benötigen zum Zugang auf das Management-Subsystem ein passendes Zertifikat.

**Leistung** Die Leistung des Systems in Bezug auf Antwortzeit und Durchsatz hängt im Wesentlichen vom verwendeten Datenbanksystem und der Hardware auf dem dieses läuft ab. Neben einer MySql Datenbank, welche in der Konzeptimplementierung verwendet wird, bietet sich für einen späteren Einsatz mitunter ein leistungsstärkeres, kommerzielles Datenbanksystem wie DB2 oder Oracle an.

**Skalierbarkeit** Wie die Leistung, hängt auch die Skalierbarkeit im Wesentlichen vom verwendeten Datenbanksystem ab. Neben der Datenbank muss auch die Management-GUI für eine große Anzahl von Benutzern und Ressourcen ausgelegt sein, damit stets eine einfache Konfiguration und Verwaltung, auch von vielen Benutzern und Ressourcen, möglich ist.

**Portabilität** Da der VO-Layer mit Hilfe von Java entwickelt wird, ist dieser auf einer Vielzahl unterschiedlicher Betriebssystem und Hardware einsetzbar. Bei den verwendbaren Betriebssystemen gibt es allerdings die Einschränkung, dass der VO-Layer nur auf Linux Betriebssysteme laufen kann, welche eine Java-VM der Version 1.5 unterstützen. Die Einschränkung auf Linux erfolgt deshalb, da der VO-Layer mit Hilfe von Shell-Skripten Grid Befehle ausführt und diese Skripte nur in einer Linux Umgebung verfügbar sind.

# 8 Zusammenfassung und Ausblick

## 8.1 Zusammenfassung

Grids sind eine immer wichtiger werdende IT-Infrastruktur, deren Ziel die gemeinsame, institutionsübergreifende Nutzung von Ressourcen ist. Zur Zeit sind Grids vor allem Gegenstand zahlreicher Projekte innerhalb der Wissenschaft, deren Ziel es ist, eines Tages ein weltumspannendes einheitliche Grid zu etablieren, mit dem so einfach auf Ressourcen zugegriffen werden kann, wie man Strom aus der Steckdose bezieht. Damit diese Vision Wirklichkeit werden kann, gilt es eine Reihe von Problemen zu lösen. Eines davon betrifft die Authentifizierung und Autorisierung der Benutzer in einem Grid, welches mit verschiedenen Grid Middleware-Technologien aufgebaut wird. Ein ebensolches, heterogenes Grid ist auch das D-Grid, in dem drei verschiedenen Grid Middleware-Technologien in verschiedenen Grid-Communities zum Einsatz kommen. Diese Diplomarbeit beschäftigt sich mit der Erstellung einer einheitlichen Authentifizierung- und Autorisierungsschnittstelle für solche heterogene Grids.

Zunächst wurden in Kapitel 2 die Grundlagen im Bereich Grids eingeführt. Dazu gehört der Begriff *Grid-Computing*, die *virtuelle Organisation*, sowie die Authentifizierung und Autorisierung. Des Weiteren wurde das D-Grid und alle darin verwendeten Grid Middleware-Technologien vorgestellt. In Kapitel 3 wurden anhand der Aufgabenstellung die Anforderungen an ein einheitliches AAI identifiziert. Diese wurden in Form der drei Use Cases *Authentifizieren*, *Autorisieren* und *Management* genau spezifiziert. Zu diesen funktionalen Anforderungen wurden auch noch eine Reihe von nicht-funktionalen Anforderungen ermittelt. Bereits existierende Techniken zur Authentifizierung und Autorisierung in Grids wurden in Kapitel 4 auf ihre Tauglichkeit, in Bezug auf die Erfüllung der soeben ermittelten Anforderungen, hin untersucht. Dabei hat sich gezeigt, dass alle untersuchten Technologien mindestens eine Anforderung nicht erfüllen und somit ein neues System, der so genannte *VO-Layer*, entwickelt wurde.

Das System des VO-Layers wurde in Kapitel 5 aus den definierten Anforderungen entwickelt. Der wichtigste Schritt dabei war die Entwicklung einer generischen VO-Struktur, welche in allen D-Grid Communities eingesetzt werden kann. Diese Struktur unterstützt VOs und sub-VOs. Benutzer und Ressourcen können Mitglieder von VOs sein und zur Autorisierung können Gruppen, Rollen und Fähigkeiten verwendet werden. Das so modellierte System wurde anschließend als Prototyp implementiert. Kapitel 6 beschreibt dazu das Testbett in dem der Prototyp entwickelt wurde und anschließend wurden die wichtigsten Teile des Quellcodes genau vorgestellt. Kaptitel 7 bewertet den in dieser Arbeit entwickelten VO-Layer anhand der in Kapitel 3 identifizierten Anforderungen. Dabei wurde festgestellt, dass die gestellten Anforderungen vom VO-Layer erfüllt werden.

## 8.2 Ausblick

Das Ziel dieser Arbeit war die Entwicklung einer einheitlichen Authentifizierungs- und Autorisierungs-Schnittstelle für heterogene Grids, unter der Voraussetzung, dass kein bereits vorhandenes Konzept die gestellten Anforderungen erfüllt. Das in dieser Diplomarbeit entwickelte System bietet seinen Benutzern die Möglichkeit, transparent auf die Ressourcen eines heterogenen Grids zuzugreifen. Durch die neu entwickelte, generische VO-Struktur kann die Autorisierung sehr flexibel gehandhabt werden und jede D-Grid Community kann ihr eigenes Autorisierungskonzept beibehalten.

Neben diesen Kernfunktionalitäten wurden im Laufe dieser Diplomarbeit einige Erweiterungen zu dem hier entwickelten System identifiziert. Diese Erweiterungen bilden dabei einen echten Mehrwert und könnten dazu beitragen, dass der VO-Layer seine Chance in einem professionellen Grid Umfeld erhält. Folgende Ergänzungen sind dabei sinnvoll:

- Entwicklung einer GUI mit einer einfachen Benutzerschnittstelle zur Authentifizierung des Benutzers im Grid und zur Ausführung von Jobs auf Grid-Ressourcen.
- Entwicklung einer GUI zur Visualisierung und einfachen Änderung der VO-Struktur über eine Management Anwendung.
- Implementierung von Methoden zur Zugriffskontrolle und Security, damit nur berechtigte Benutzer die VO-Struktur verändern können.
- Implementierung von Methoden zur sofortigen Aktualisierung der Autorisierungsinformationen in den Grid Middleware-Technologien.
- Bereitstellung von Methoden zur Ausführung weiterer Jobs um die Akzeptanz des VO-Layers in den Communities zu erhöhen.
- Erweiterung des VO-Layer um eine Zertifikatsverwaltung, welche es dem Benutzer auf einfache Weise gestattet für verschiedene Situationen verschiedene Zertifikate zu verwenden.
- Portierung des VO-Layers auf weitere Zielsysteme.

# A Installation

Dieser Anhang beschäftigt sich mir der Installation und Konfiguration der drei, in dieser Diplomarbeit, verwendeten Grid Middleware-Technologien. In Kapitel 6.1 wird die Testumgebung vorgestellt, in der der VO-Layer entwickelt wurde. Im Folgenden wird auf die Installation und Konfiguration der drei Grid Middleware-Technologien eingegangen.

## A.1 Globus Toolkit 4

Die Installation von Globus erfolgt anhand der in [SoCh 05] beschriebenen Installationsanleitung. Diese beschreibt das Einrichten eines Systems zum Testen und Entwickeln von Grid-Services mit Hilfe von Globus Toolkit. Als weitere Installationshilfe sei zudem auf die Globus Webseite[1] verwiesen.

Die Konfiguration von Globus erfolgt nach den Standardvorgaben, allerdings muss beim gemeinsamen Betrieb von Globus und gLite beziehungsweise UNICORE einiges beachtet werden:

**Globus und gLite** Da diese beiden Grid Middleware-Technologien sehr ähnliche Konzepte verwenden, muss bei einer Installation der beiden auf dem selben Betriebssystem beachtet werden, dass die gridmap files der beiden nicht verwechselt werden. Globus speichert sein gridmap file standardmäßig unter `$GLOBUS_LOCATION/etc/gridmapfile` wobei `$GLOBUS_LOCATION` das Installationsverzeichnis von Globus ist (standardmäßig `/usr/local/gt4`). Das gridmap file von Globus kann aber auch an jedem anderen Ort im Dateisystem gespeichert werden, dazu muss im Security Descriptor des Web-Services der Wert `⟨gridmap value='etc/gridmapfile' /⟩` angepasst werden. Das bedeutet auch, dass in Globus für jeden Web-Service ein anderes gridmap file zur Autorisierung verwendet werden kann.

**Globus und UNICORE** Bei der Verwendung dieser beiden Grid Middleware-Technologien, muss darauf geachtet werden, dass die beiden nicht den selben Port verwenden. Dazu kann beim Start von Globus mit dem Befehl `globus-start-container` noch das Argument `-p xxxx` mit der gewünschten Portnummer angegeben werden.

[1] http://globus.org/toolkit/

## A.2 UNICORE

Die Installation von UNICORE ist, verglichen mit der von Globus oder gLite, relativ einfach. Anhand einer graphischen Step-by-Step Installation werden die wichtigsten Parameter spezifiziert. Eine Anleitung hierzu gibt [UNICORE 08] oder [Breu 07]. Für UNICORE 5 empfiehlt sich die Anleitung [Ramb ]. Für die Installation, Konfiguration und Bedienung des UNICORE Commandline Clients sei auf [UCC] verwiesen.

Bei einem gleichzeitigen Betrieb von Globus Toolkit und UNICORE muss beachtet werden, dass die beiden Grid Middleware-Technologien nicht den selben Port für ihre Web-Services verwenden. Dies muss bereits bei der Installation des UNICORE Servers berücksichtigt werden.

## A.3 gLite

Für gLite muss auf dem Testsystem eine virtuelle Umgebung mit Hilfe von *VirtualBox* eingerichtet werden (Hilfestellung hierzu gibt [VBox 08]). Nachdem die virtuelle Umgebung eingerichtet ist, kann Scientific Linux Cern [SLC4] darauf installiert werden. Anschließend wird auf diesem System der gLite VOMS-Knoten installiert. Informationen dazu finden sich in den Installationsanweisungen [GIG3.1] und [gLite 07].

Bei der Konfiguration von gLite muss darauf geachtet werden, dass, bei gemeinsamen Betrieb mit Globus auf einem System, das gridmap file von gLite nicht mit jenem von Globus verwechselt wird. Standardmäßig wird das gridmap file in gLite unter `/etc/grid-security` abgelegt, allerdings kann durch Ändern der Zeile `host.gridmapfile` in der Konfigurationsdatei `$GLITE_LOCATION/etc/config/glite-global.cfg.xml` das gridmap file an einem beliebigen Ort im Dateisystem abgelegt werden.

# Abbildungsverzeichnis

# Tabellenverzeichnis

# Listings

# Literaturverzeichnis


[ACC+ 03] Alfieri, R., R. Cecchini, V. Ciaschini, L. dell'Agnello, A. Frohner, A. Gianoli, K. Lorentey und F. Spataro: *VOMS, an Authorization System for Virtual Organizations*. http://grid-auth.infn.it/docs/VOMS-Santiago.pdf, Februar 2003.

[ACC+ 04] Alfieri, R., R. Cecchini, V. Ciaschini, F. Spataro, L. dell'Agnello, A. Frohner und K. Lorentey: *From gridmap-file to VOMS: managing Authorization in a Grid environment*. http://grid-auth.infn.it/docs/voms-FGCS.pdf, April 2004.

[BDE+ 07] Büchner, Otto, Christa Dohmen, Thomas Eifert, Harry Enke, Thomas Fieseler, Anton Frank, Stefan Freitag, Ariel Garcia, Christian Grimm, Wolfgang Gürich, Helmut Heller, Thomas Jejkal, Holger Nitsche, Olaf Schneider und Wolfgang Ziegler: *Betriebskonzept für die D-Grid Infrastruktur*. http://www.d-grid.de/uploads/media/D-Grid-Betriebskonzept.pdf, Oktober 2007.

[BrDu 04] Brügge, Bernd und Allen H. Dutoit: *Objektorientierte Softwaretechnik*. Pearson Studium, ISBN 3-8273-7082-5, 2004.

[Breu 07] Breu, Rebecca: *UNICORE 6 in 30 minutes*. http://www.unicore.eu/documentation/tutorials/unicore6/files/unicore6_30min.pdf, Dezember 2007.

[Cant 05] Cantor, Scott: *Shibboleth Architecture: Protocols and Profiles*. http://shibboleth.internet2.edu/docs/internet2-mace-shibboleth-arch-protocols-200509.pdf, September 2005.

[CFF+ 04] Czajkowski, Karl, Donald F Ferguson, Ian Foster, Jeffrey Frey, Steve Graham, Igor Sedukhin, David Snelling, Steve Tuecke und William Vambenepe: *The WS-Resource Framework*. http://www.globus.org/wsrf/specs/ws-wsrf.pdf, Mai 2004.

[Cias 06] Ciaschini, Vincenzo: *EGEE User's Guide: VOMS Core Services*. https://edms.cern.ch/document/571991/1, Oktober 2006.

[DEF+ 06a] Dussa, Tobias, Ursula Epting, Bartol Filipovic, Gerti Foest, Jürgen Glowka, Joachim Götze, Christian Grimm, Markus Hillenbrand, Christian Kohlschütter, Rudolf Lohner, Siegfried Makedanz, Paul Müller, Marcus Pattloch, Stefan Piger, Tobias Straub und Jan Wiebelitz: *Analyse von AA-Infrastrukturen in Grid-Middleware*. http://www.d-grid.de/fileadmin/user_upload/documents/DGI-FG3-4/Analyse-AAI_v1_1.pdf, March 2006.

[DEF+ 06b] Dussa, Tobias, Ursula Epting, Bartol Filipovic, Gerti Foest, Joachim Götze, Christian Grimm, Markus Hillenbrand, Christian Kohlschütter, Rudolf Lohner, Paul Müller, Marcus Pattloch, Stefan Piger, Tobias Straub und Jan Wiebelitz: *Use Cases for Authorization in Grid-Middleware*. http://www.d-grid.de/fileadmin/user_upload/documents/DGI-FG3-4/Use_Cases_V13.pdf, September 2006.

[Ecke 05] Eckert, Claudia: *IT-Sicherheit: Konzepte - Verfahren - Protokolle*. Oldenbourg, ISBN 978-3486252989, 2005.

[EpPa 05] Epting, Ursula und Marcus Pattloch: *Authentifizierung im D-Grid*. http://www.d-grid.de/fileadmin/user_upload/documents/DGI-FG3-4/vorschlagspapier-authz_v2.pdf, Dezember 2005.

[Erwi 03] Erwin, Dietmar: *UNICORE Plus Final Report*. http://www.unicore.eu/documentation/files/erwin-2003-UPF.pdf, 2003.

[FaZi a] Faroughi, Arash und Wolfgang Ziegler: *Integration UNICORE - VOMS*. http://www.d-grid.de/fileadmin/user_upload/documents/DGI-FG1-IVOM/IVOM-VOMS-1.1.pdf.

[FaZi b] Faroughi, Roozbeh und Wolfgang Ziegler: *Integration UNICORE - Shibboleth*. http://www.d-grid.de/fileadmin/user_upload/documents/DGI-FG1-IVOM/IVOM-Shibboleth-1.1_01.pdf.

[FKNT 02] Foster, Ian, Carl Kesselman, Jeffrey M. Nick und Steven Tuecke: *The Physiology of the Grid - An Open Grid Services Architecture for Distributed Systems Integration*. http://www.globus.org/alliance/publications/papers/ogsa.pdf, Juni 2002.

[FKS+ 06] Foster, I., H. Kishimoto, A. Savva, D. Berry, A. Djaoui, A. Grimshaw, B. Horn, F. Maciel, F. Siebenlist, R. Subramaniam, J. Treadwell und J. Von Reich: *The Open Grid Services Architecture, Version 1.5*. http://www.ogf.org/documents/GFD.80.pdf, Juli 2006.

[FKT 01] Foster, Ian, Carl Kesselman und Steven Tuecke: *The Anatomy of the Grid. Enabling Scalable Virtual Organisations*. International Journal of High Performance Computing Applications, 3:200–222, 2001.

[FKTT 98] Foster, Ian, Carl Kesselman, Gene Tsudik und Steven Tuecke: *A Security Architecture for Computational Grids*. ftp://ftp.globus.org/pub/globus/papers/security.pdf, 1998.

[FoKe 98] Foster, Ian und Carl Kesselman: *The Grid: Blueprint for a New Computing Infrastructure*. Morgan Kaufmann, ISBN 1-55860-475-8, November 1998.

[Fost 02] Foster, Ian: *What is the Grid? A Three Point Checklist*. http://www-fp.mcs.anl.gov/~foster/Articles/WhatIsTheGrid.pdf, Juli 2002.

[Fost 05] Foster, Ian: *Globus Toolkit Version 4: Software for Service-Oriented Systems*. http://www.globus.org/alliance/events/sc05/GT4.pdf, 2005.

[GGG+ 08] Gietz, Peter, Christian Grimm, Ralf Gröper, Martin Haase, Siegfried Makedanz, Jörg Matthes, Hans Pfeiffenberger, Michael Schiffers, Thomas Weuffel und Wolfgang Ziegler: *Integra-*

*tion and Deployment*. http://www.d-grid.de/fileadmin/user_upload/documents/DGI-FG1-IVOM/Report_AP44_5_v1_0_final.pdf, Mai 2008.

[GHPS 07] Gietz, Peter, Martin Haase, Hans Pfeiffenberger und Michael Schiffers: *VO-Management Requirements from a Community Perspective*. http://www.d-grid.de/fileadmin/user_upload/documents/DGI-FG1-IVOM/requirements-v1.0.pdf, 2007.

[GIG3.1] *Generic Installation and Configuration Guide for gLite 3.1*. https://twiki.cern.ch/twiki/bin/view/LCG/GenericInstallGuide310.

[gLite 07] *gLite 3.1 VOMS server Installation & configuration guide*. https://edms.cern.ch/file/818502/3.1/gLite_3.1_VOMS_Installation_Configuration_guide.pdf, Dezember 2007.

[GRSW 06] Gemmill, Jill, John-Paul Robinson, Tom Scavo und Von Welch: *myVocs and GridShib: Integrated VO Management*. http://grid.ncsa.uiuc.edu/presentations/i2mm-myvocs-gridshib-april06.ppt, April 2006.

[Henn 06] Hennig, Patrick: *Analyse der Verfahren zur Authentifizierung und Autorisierung in Grid-Middlewares am Beispiel gLite*. http://www.rrzn.uni-hannover.de/fileadmin/ful/kolloquium/Bachelor_Hennig.pdf, March 2006.

[ISO9126] *ISO/IEC 9126-1:2001*. http://www.iso.org/iso/iso_catalogue/catalogue_tc/catalogue_detail.htm?csnumber=22749.

[JDT 05] JRA1-Design-Team: *EGEE Middleware Architecture*. https://edms.cern.ch/document/594698/, July 2005.

[JH 04] John Hughes, Eve Maler: *Technical Overview of the OASIS Security Assertion Markup Language (SAML) V1.1*. http://www.oasis-open.org/committees/download.php/6837/sstc-saml-tech-overview-1.1-cd.pdf, Mai 2004.

[Lato 06] Latour, André: *Ein Web-Service-basierter Autorisierungsdienst für Grid-Umgebungen am Beispiel von UNICORE*. http://www.fz-juelich.de/jsc/docs/autoren2006/latour, February 2006.

[Lore 05] Lorentey, Károly: *EGEE User's Guide: VOMS Admin Services*. https://edms.cern.ch/document/572406/, März 2005.

[Mari 06] Marienfeld, Stefan: *Aufbau und Analyse einer Shibboleth/GridShib-Infrastruktur*. Diplomarbeit, Leibniz Universtität Hannover, Juli 2006.

[MSZ 08] Milke, Jens-Michael, Michael Schiffers und Wolfgang Ziegler: *Rahmenkonzept für das Management Virtueller Organisationen im D-Grid*. http://www.d-grid.de/fileadmin/user_upload/documents/DGI-FG1-10/VO_Rahmenkonzept-final.pdf, Februar 2008.

[NKG 07] Neuroth, Heike, Martina Kerzel und Wolfgang Gentzsch: *Die D-Grid Initiative*. Universitätsverlag Göttingen, ISBN 978-3-940344-01-4, September 2007. http://www.univerlag.uni-goettingen.de/content/list.php?notback=1&details=isbn-978-3-940344-01-4.

[PGG+ 07] Piger, Stefan, Christian Grimm, Joachim Götze, Markus Hillenbrand, Jürgen Glowka, Gerti Foest und Tobias Dussa: *AAI-Prototyp für die D-Grid Infrastruktur*. http://www.d-grid.de/fileadmin/user_upload/documents/DGI-FG3-4/AAI_Prototyp.pdf, Juni 2007.

[PGGH 07] Piger, Stefan, Christian Grimm, Joachim Götze und Markus Hillenbrand: *Ergebnisse der Interviews mit den D-Grid-Communities*. http://www.d-grid.de/fileadmin/user_upload/documents/DGI-FG3-4/Auswertung_Interviews_11.pdf, Januar 2007.

[Ramb ] Rambadt, Michael: *Server Installation Tutorial*. http://www.unicore.eu/documentation/tutorials/unicore5/files/UNICORE-Server-Installation.pdf.

[RBJS 01] Roure, David De, Mark Baker, Nicholas Jennings und Nigel Shadbolt: *The Evolution of the Grid*. http://www.semanticgrid.org/documents/evolution/evolution.pdf, 2001.

[Schi 07] Schiffers, Michael: *Management dynamischer Virtueller Organisationen in Grids*. Dissertation, Ludwig-Maximilians-Universität München, Juli 2007. http://edoc.ub.uni-muenchen.de/7352/.

[Schi 08] Schiffers, Michael: *VO Management*. http://dgi.d-grid.de/fileadmin/user_upload/documents/DGI-FG1-IVOM/Schiffers_IVOM_Workshop_Februar_2008.pdf, February 2008.

[Schu 07] Schuller, Bernt: *UNICORE Version 6.0*. http://www.unicore.eu/documentation/tutorials/unicore6/files/2007-07-25-DGrid-Unicore6.pdf, Juli 2007.

[ScNi 02] Schopf, Jennifer M. und Bill Nitzberg: *Grids: The Top Ten Questions*. http://www.globus.org/alliance/publications/papers/topten.final.pdf, 2002.

[SLC4] *Scientific Linux CERN 4*. http://linux.web.cern.ch/linux/scientific4/.

[SoCh 05] Sotomayor, Borja und Lisa Childers: *Globus Tookit 4: Programming Java Services*. Morgan Kaufmann, ISBN 0-12-369404-3, Dezember 2005.

[TWE+ 04] Tuecke, S., V. Welch, D. Engert, L. Pearlman und M. Thompson: *Internet X.509 Public Key Infrastructure (PKI): Proxy Certificate Profile*. http://www.ietf.org/rfc/rfc3820.txt, Juni 2004.

[UCC] *UNICORE UCC*, 2008, http://www.unicore.eu/documentation/manuals/unicore6/ucc/index.html .

[UNICORE 08] *Installation and Configuration of UNICORE 6*. http://www.unicore.eu/documentation/tutorials/unicore6/files/Installation_UNICORE6.pdf, July 2008.

[VBox 08] *Sun xVM VirtualBox - Version 1.6.4*. http://www.virtualbox.org/download/1.6.4/UserManual.pdf, July 2008.

[VOM 06] *Thesenpapier zum VO Management in D-Grid*. http://www.d-grid.de/fileadmin/dgrid_document/Dokumente/VOMS-Thesenpapier.pdf, Januar 2006.

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

[WBKS 05] Welch, Von, Tom Barton, Kate Keahey und Frank Siebenlist: *Attributes, Anonymity, and Access: Shibboleth and Globus Integration to Facilitate Grid Collaboration*. http://grid.ncsa.uiuc.edu/papers/gridshib-pki05-final.pdf, April 2005.

[Witz 08] Witzig, Christoph: *Interoperability Shibboleth - gLite*. http://www.d-grid.de/fileadmin/user_upload/documents/DGI-FG1-IVOM/Witzig_080219_ivom.pdf, Februar 2008.

[X.509] *ITU-T Recommendation X.509*. http://www.itu.int/rec/T-REC-X.509-200508-I, August 2005.

[Zieg 08] Ziegler, Wolfgang: *Autorisierung in Grid Middleware*. http://www.d-grid.de/fileadmin/user_upload/documents/DGI-FG1-IVOM/IVOM_WS_Ziegler.pdf, Februar 2008.

[ZiGr 06] Ziegler, Wolfgang und Christian Grimm: *Interoperabilität und Integration der VO-Management Technologien im D-Grid*. http://www.d-grid.de/fileadmin/user_upload/documents/DGI-FG1-IVOM/Ueberblick_IVOM.pdf, Dezember 2006.

# Abkürzungsverzeichnis

| | |
|---|---|
| **AA** | Attribute Authority. |
| **AAI** | Authentifizierungs- und Autorisierungs-Infrastruktur. |
| **ABAC** | Attribute-Based Access Control. |
| **ACL** | Access Control List. |
| **ACS** | Assertion Consumer Service. |
| **AJO** | Abstract Job Object. |
| **API** | Application Programming Interface. |
| **AR** | Attribute Requester. |
| **ARP** | Attribute Release Policy. |
| **C** | Country. |
| **CA** | Certificate Authority. |
| **CERN** | Conseil Européen pour la Recherche Nucléaire. |
| **CN** | Common Name. |
| **CVS** | Comma Separated Value. |
| **DAC** | Discretionary Access Control. |
| **DB** | Database. |
| **DGI** | D-Grid Integrationsprojekt. |
| **DN** | Distinguished Name. |
| **EGEE** | Enabling Grids for E-sciencE. |
| **FQAN** | Fully Qualified Attribut Name. |
| **FTP** | File Transfer Protocol. |
| **GRRS** | Grid Resource Registry Service. |
| **GSI** | Grid Security Infrastructure. |
| **GT4** | Globus Toolkit 4. |
| **GUI** | Graphical User Interface. |
| **HEPCG** | Hochenergiephysik Computing Grid. |
| **HS** | Handle Service. |
| **IdP** | Identity Provider. |
| **IVOM** | Interoperabilität und Integration der VO-Management Technologien im D-Grid. |

**JVM** Java Virtual Machine.
**LCG** LHC Computing Grid.
**LHC** Large Hadron Collider.
**MAC** Mandatory Access Control.
**MDS** Monitoring and Discovery System.
**MVC** Model View Controller.
**NJS** Network Job Supervisor.
**O** Organisation.
**OGSA** Open Grid Service Architecture.
**ORM** Object Relational Mapping.
**OU** Organisational Unit.
**PIN** Persönliche Identifikationsnummer.
**PKI** Public Key Infrastruktur.
**PoC** Proof of Concept.
**RBAC** Role-Based Access Control.
**RFC** Requests for Comments.
**RP** Resource Provider.
**SAML** Security Assertion Markup Language.
**SL** Scientific Linux.
**SLA** Service Level Agreements.
**SLC** Scientific Linux Cern.
**SOA** Serviceorientierte Architektur.
**SP** Service Provider.
**SSO** Single Sign On.
**TAN** Transaktionsnummer.
**TSI** Target System Interface.
**ttp** TTP.
**UCC** UNICORE Commandline Client.
**USite** UNICORE Site.
**UUDB** UNICORE User Database.
**VO** Virtuelle Organisation.
**VOL-DB** VO-Layer Datenbank.
**VOMRS** Virtual Organization Membership Registration Service.
**VOMS** Virtual Organisation Membership Service.
**VOX** VOMS eXtension.

**VSite** Virtual Site.

**WAYF** Where Are You From.

**WLHCG** Welt LHC Grid.

**WS** Web Service.

**WSRF** Web Services Resource Framework.