

Fachgebiet 3.2
AAI/VO

Systementwurf und prototypische Implementierung von Kernfunk- tionen einer Management-Suite für D-Grid-VOs

Version 0.8, 22. August 2009

Autoren

Michael Schiffers, LMU

Wolfgang Kirchler, LMU

Das diesem Bericht zu Grunde liegende Vorhaben wurde mit Mitteln des Bundesministeriums für Bildung und Forschung unter dem Förderkennzeichen 01IG07014 gefördert. Die Verantwortung für den Inhalt dieser Veröffentlichung liegt bei den Autoren.

Inhalt

1	MANAGEMENT SUMMARY.....	3
2	ZIEL DES DOKUMENTS.....	3
3	EINFÜHRUNG	3
4	ANFORDERUNGEN DER MANAGEMENTANWENDUNGEN.....	5
4.1	Erstellen eines Benutzers	6
4.2	Erstellen einer VO	6
4.3	Hinzufügen eines Benutzers zu einer VO.....	6
4.4	Entfernen eines Benutzers aus einer VO.....	7
4.5	Nicht-Funktionale Anforderungen	7
5	FUNKTIONALER SYSTEMENTWURF	8
5.1	Datenverwaltung	8
5.2	Globaler Kontrollfluss	11
5.3	Softwarearchitektur	12
5.4	Subsysteme	13
5.5	Objektentwurf	14
6	PROTOTYPISCHE IMPLEMENTIERUNG	17
6.1	Hauptanwendung	17
6.2	Authentifizieren	17
6.3	Autorisieren	18
6.4	Management	20
7	STATUS UND AUSBLICK.....	23
8	ABKÜRZUNGEN	25
9	REFERENZEN.....	25

1 Management Summary

Um die Heterogenität von Ressourcen, Diensten, Sicherheitsniveaus und Policies im D-Grid-Betrieb zu verschatten, ist neben einer VO-Managementarchitektur, die von diesen Spezifika abstrahieren kann, ein „Werkzeugkasten“ (*tool suite*) erforderlich, der eine solche Abstraktion unterstützt und das Management virtueller Organisationen im D-Grid erleichtert. In einem früheren Dokument wurde ein entsprechendes Rahmenwerk beschrieben [suite]. Dieses Dokument baut auf den Vorarbeiten auf und liefert den dazu gehörenden Systementwurf auf der Basis der in [suite] referenzierten Feasibility-Studie [kirchler]. Der Systementwurf folgt dem objektorientierten Vorgehensmuster.

Zunächst werden die funktionalen und nicht-funktionalen Anforderungen an die Suite über entsprechende Anwendungsfälle definiert. Anschließend werden das System und die Objekte entworfen. Schließlich werden die Hinweise zur prototypischen Implementierung gegeben. Das System ist inzwischen in seinen Kernfunktionen vollständig implementiert, die grafische Benutzeroberfläche (GUI) wird zur Zeit entwickelt. Für eine mögliche Überführung in den Regelbetrieb sind allerdings noch einige Lücken zu schließen. Ebenso ist zu klären, inwieweit ein solches Management-Instrument überhaupt eingeführt werden soll. Die bisherigen Erfahrungen zeigen insbesondere vor dem Hintergrund nachhaltiger Grid-Infrastrukturen einen Benefit für den Einsatz.

2 Ziel des Dokuments

Ziel des Dokuments ist die Spezifikation des Funktionsumfangs einer VO-Management-Suite, deren Grundkonzept in [suite] dargestellt wurde.

3 Einführung

In [suite] wurden die Anforderungen an die Tool-Suite und deren Basisarchitektur beschrieben. Gleichzeitig wurde anhand einiger Feasibility-Studien dargestellt, dass eine Web Services Distributed Management (WSDM)-basierte Lösung nicht sinnvoll ist, dass allerdings das Ziel einer Vereinheitlichung einer VO-Managementschnittstelle als Kern der anzustrebenden VO-Management-Suite Erfolg versprechend ist. Die im Folgenden beschriebene Funktionalität der Suite basiert auf der Feasibility-Studie [kirchler].

In [kirchler] wurde ein neues VO-Managementkonzept in Form einer zusätzlichen Abstraktionsschicht vorgeschlagen, welche die Authentifizierung und Autorisierung von der Grid-Middleware entkoppelt und in eine darüber liegende Schicht, den so genannten VO-Layer, integriert. 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 überhaupt vereinheitlicht werden kann, ist eine generische VO-Struktur erforderlich. Diese wurde in [generischevo] vorgelegt.

Im Rahmen der Studie konnte gezeigt werden, dass die wesentlichen Authentifizierungs- und Autorisierungsanforderungen im D-Grid mit dem VO-Layer-Konzept erfüllbar sind. Der Benutzer gibt wie gewohnt sein Zertifikat an, anschließend übernimmt wird der VO-Layer die Zertifikatinformationen. Dort erfolgt der eigentliche Authentifizierungsvorgang und bei positiver Authentifizierung erhält der Benutzer ein Proxy-Zertifikat, welches ihn ermächtigt, die vom Resource Provider (RP) bereitgestellten Ressourcen zu nutzen. Die Authentifizierung ist somit trotz heterogener Middleware einheitlich möglich.

Auch zur Autorisierung fungiert der VO-Layer als Proxy zwischen Benutzer und der eigentlichen Grid-Middleware. Der Benutzer möchte auf einer Ressource des Grids einen Job ausfüh-

ren, 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 geprüft, ob sich beide in derselben VO befinden (eine Folge der zugrunde liegenden VO-Definition). Falls dies zutrifft, wird überprüft, ob der Benutzer die nötigen Zugriffsrechte anhand der spezifizierten Gruppen, Rollen und Fähigkeiten 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.

Das Ziel, alle im D-Grid verwendeten VO-Konzepte mit Hilfe einer generischen VO-Struktur zu unterstützen, wird durch die eingeführte VO-Struktur erreicht. Im Rahmen der Studie konnte gezeigt werden, dass VOs mit beliebig vielen Benutzern, Ressourcen und sub-VOs etabliert werden können. 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 (siehe auch [betriebskonzept]). Dadurch wird eine generische VO-Struktur geschaffen, welche von den D-Grid Communities – unabhängig von deren VO-Konzept - einheitlich verwendet werden kann [generischevo].

Neben der Authentifizierung und Autorisierung der Benutzer ist ein einheitliches und einfaches Management der VO-Struktur durch den VO-Layer möglich. Es hat sich gezeigt, dass die schon heute für den Betrieb verwendeten Datenbanken (VOMS, GRRS) [betriebskonzept] weiter verwendet werden können. Die Weiterleitung von Änderungen in der VO-Struktur an die darunter liegenden Grid Middleware-Technologien erfolgt periodisch.

Abbildung 1 zeigt das Prinzip des VO-Layerkonzeptes gemäß [kirchler].

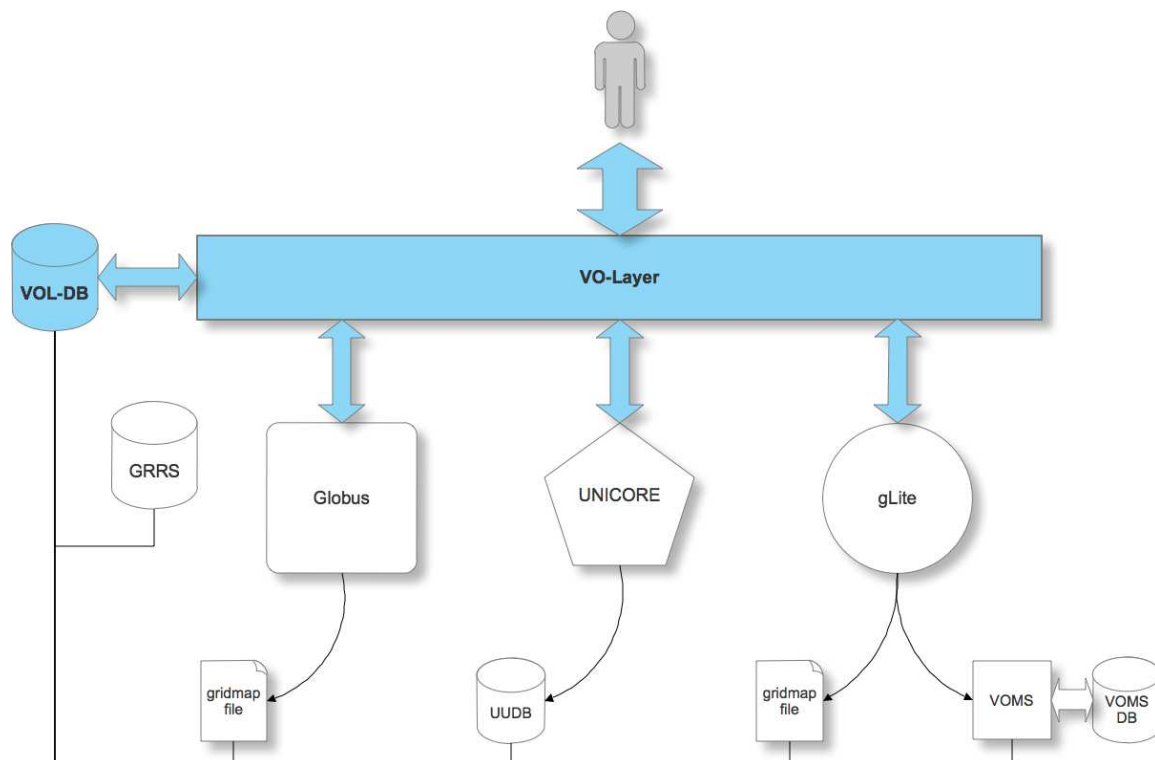


Abbildung 1: VO-Layerkonzept [kirchler]

4 Anforderungen der Managementanwendungen

In diesem Abschnitt werden die für das Management virtueller Organisationen erforderlichen Use Cases beschrieben. Sie stellen gleichzeitig die Anforderungen an die Managementsuite dar. Die für das Management erforderliche Datenstruktur soll dabei so dynamisch wie möglich sein und sich, mit Hilfe der Management Use Cases, beliebig verändern lassen. Die Use Cases in diesem Abschnitt beschreiben im Einklang mit [betriebskonzept] und [flowcharts] insbesondere das Aufnehmen und Löschen von Benutzern, Ressourcen, VOs, Gruppen und Rollen¹.

Die Management Use Cases können in vier Gruppen eingeteilt werden (siehe auch Abbildung 2 für ausgewählte Use Cases):

- Aufnehmen und löschen von Benutzern und Ressourcen
- Aufnehmen und löschen von VOs, Gruppen und Rollen
- Hinzufügen von Benutzern oder Ressourcen zu einer VO, Gruppe, Rolle
- Entfernen von Benutzern oder Ressourcen aus einer VO, Gruppe, Rolle

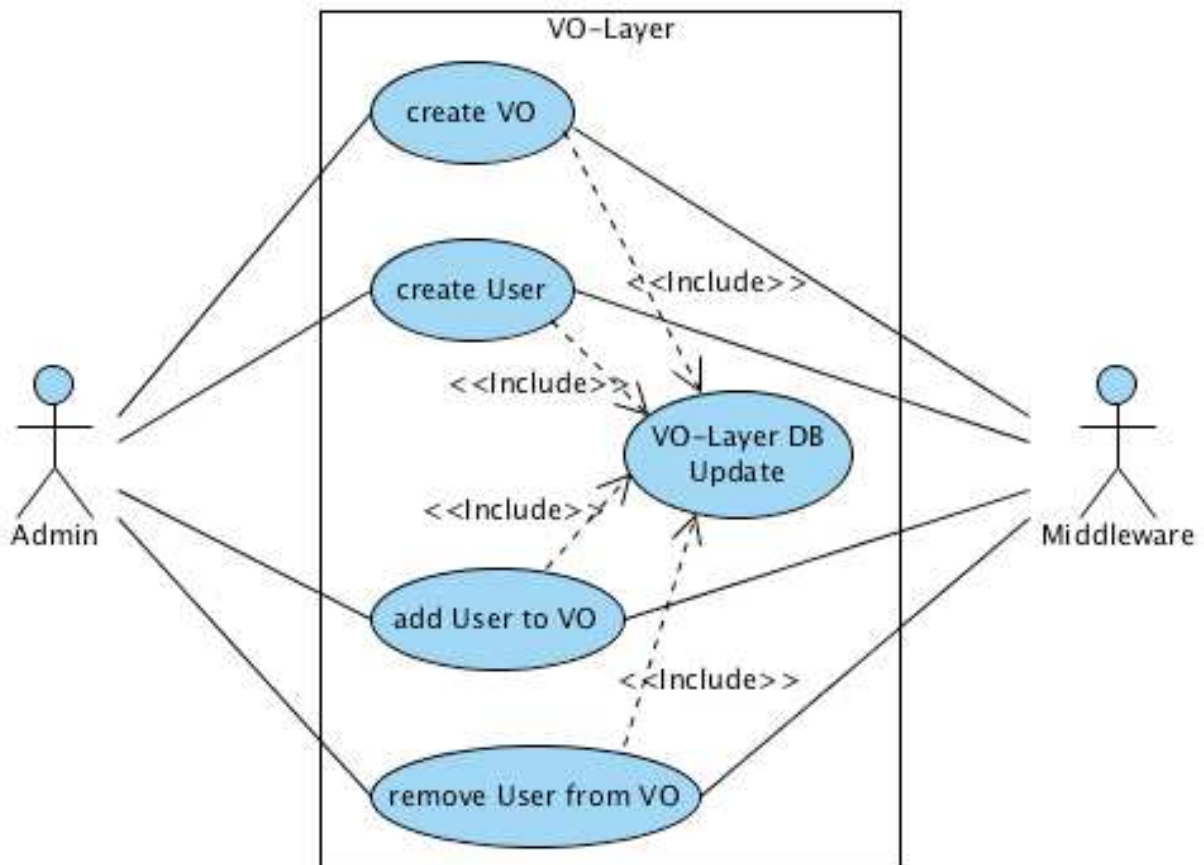


Abbildung 2: Ausgewählte Use Cases der Managementsuite [kirchler]

Zu beachten ist, dass jeder Use Case seinerseits den VO-Layer „DB Update“-Use Case aufruft, um die Änderungen des Gesamtsystems in der Datenbank festzuschreiben.

Die in Abbildung 2 aufgezeigten Use Cases sind beispielhaft in den folgenden Tabellen näher ausgeführt. Für weitere Use Cases sei auf [kirchler] verwiesen.

¹ Wir verwenden hier der Vollständigkeit halber auch Fähigkeiten (capabilities) obwohl diese im D-Grid explizit nicht unterstützt werden.

4.1 Erstellen eines Benutzers

Use Case	Create User
Akteure	Admin, Middleware
Ereignisfluss	<ol style="list-style-type: none"> 1. Admin möchte einen neuen Benutzer für das Grid erstellen und sendet den Befehl dafür an den VO-Layer. 2. VO-Layer überprüft die Parameter, erstellt den neuen Benutzer und meldet es den entsprechenden Middleware-Stacks. 3. Die Middleware-Stacks tragen ihrerseits den neuen Benutzer ein.
Anfangsbedingungen	Der Benutzer ist als Administrator angemeldet.
Abschlussbedingungen	Der Admin erhält eine Bestätigung oder eine Fehlermeldung.

Dieser Use Case gilt analog für das Erstellen von Ressourcen und das Löschen von Benutzern oder Ressourcen.

4.2 Erstellen einer VO

Use Case	Create VO
Akteure	Admin, Middleware
Ereignisfluss	<ol style="list-style-type: none"> 1. Admin möchte eine neue VO erstellen und sendet dazu einen Befehl an den VO-Layer. 2. VO-Layer überprüft die Parameter, erstellt die neue VO und meldet es den entsprechenden Middleware-Stacks. 3. Die Middleware-Stacks tragen ihrerseits die neuen Autorisierungsinformationen bei sich ein.
Anfangsbedingungen	Der Benutzer ist als Administrator angemeldet.
Abschlussbedingungen	Der Admin erhält eine Bestätigung oder eine Fehlermeldung.

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

4.3 Hinzufügen eines Benutzers zu einer VO

Use Case	Add User to VO
Akteure	Admin, Middleware
Ereignisfluss	<ol style="list-style-type: none"> 1. Admin möchte einen Benutzer zu einer weiteren VO hinzufügen. 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 Middleware-Stacks. 3. Die Middleware-Stacks fügen ihrerseits die neuen

	Autorisierungsinformationen bei sich ein.
Anfangsbedingungen	Der Benutzer ist als Administrator angemeldet.
Abschlussbedingungen	Der Admin erhält eine Bestätigung oder eine Fehlermeldung.

Dieser Use Case gilt analog für das Hinzufügen von Benutzern zu Gruppen und Rollen. Gleiches gilt für das Hinzufügen von Ressourcen.

4.4 Entfernen eines Benutzers aus einer VO

Use Case	Remove User from VO
Akteure	Admin, Middleware
Ereignisfluss	<ol style="list-style-type: none"> 1. Admin möchte einen Benutzer aus einer VO entfernen. 2. VO-Layer überprüft die Parameter, entfernt den Benutzer aus der VO in der VO-Layer DB und meldet es an alle betroffenen Grid Middleware-Stacks. 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.

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

Bei allen Use Cases wird zusätzlich verlangt, dass die Änderungen auch an die Grid Middleware-Stacks weitergegeben und dort gespeichert werden. Dabei werden die Änderungen, je nach verwendeter Grid-Middleware, in einem gridmap-File (Globus Toolkit und gLite) oder einer UUDB (UNICORE) gespeichert.

4.5 Nicht-Funktionale Anforderungen

Als nicht-funktionale Anforderungen gemäß ISO 9126 Standard wird gefordert:

- Benutzerfreundlichkeit: Um die Bedienbarkeit zu erleichtern, soll eine grafische 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.

5 Funktionaler Systementwurf

In diesem Kapitel wird der VO-Layer modelliert auf der Basis der in [generischevo] definierten generischen VO-Struktur, die es erlaubt, auf alle im D-Grid verwendeten Autorisierungskonzepte abgebildet zu werden. Die für das VO-Management erforderliche Datenverwaltung wird ebenso angegeben, wie der Entwurf der Systemstruktur und der assoziierte Objektentwurf.

5.1 Datenverwaltung

Für das VO-Management wird auf dem VO-Layer ein Datenmanagement erforderlich, das unabhängig von verwendeten Datenmodellen des D-Grids (VOMS, GRRS) ist. Das Datenmodell des VO-Layer basiert auf Standard-Software-Patterns und kann dadurch eine flexible Struktur aus VOs, mit dazugehörigen sub-VOs, Benutzern und Ressourcen annehmen. Zusätzlich werden auch Substrukturen wie Gruppen, Rollen und Fähigkeiten unterstützt, die zu Berechtigungsnachweisen verwendet werden. Abbildung 3 zeigt das Datenmodell des VO-Layers als Klassendiagramm mit den unterstützten Attributen und Operationen.

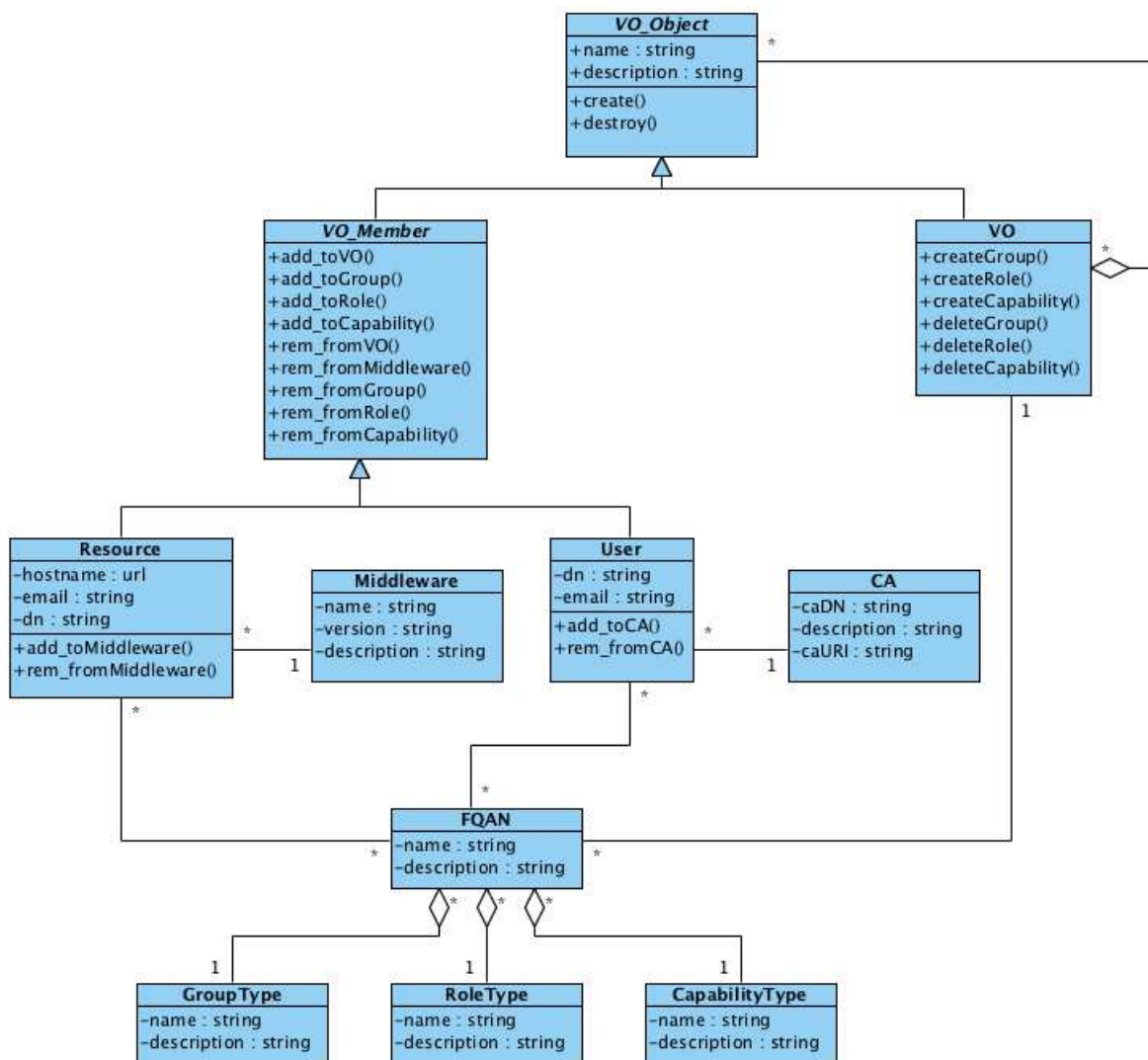


Abbildung 3: Datenmodell des VO-Layers nach [kirchler]

Jede Ressource ist mit genau einer Middleware-Klasse assoziiert, wobei über die Middleware auf beliebig viele Ressourcen zugegriffen werden kann. Das gleiche gilt für die Benutzer, denen jeweils genau ein CA-Objekt (Zertifikat) assoziiert ist.

Zur persistenten Speicherung des Datenmodells werden relationale Datenbanken vorgeschlagen. Abbildung 4 zeigt das entsprechende Datenbankschema.

Für die Gruppen, Rollen und Fähigkeiten existiert jeweils eine Tabelle, welche den Typ spezifiziert (`GroupType`, `RoleType` und `CapabilityType`). Aus diesen wird der in [generischevo] definierte FQAN erzeugt, welcher stets aus einer Kombination von VO, Gruppentyp und Rollentyp 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, wird der VO-Layer überprüfen, ob der Benutzer und die Ressource in derselben 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 gespeichert². 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

² Die hier beschriebene Struktur ist im Prototyp so implementiert. Für ein Deployment in den Produktivbetrieb ist an entsprechendes „Customizing“ erforderlich. Dazu wird ein Konfigurationsskript bereitgestellt.

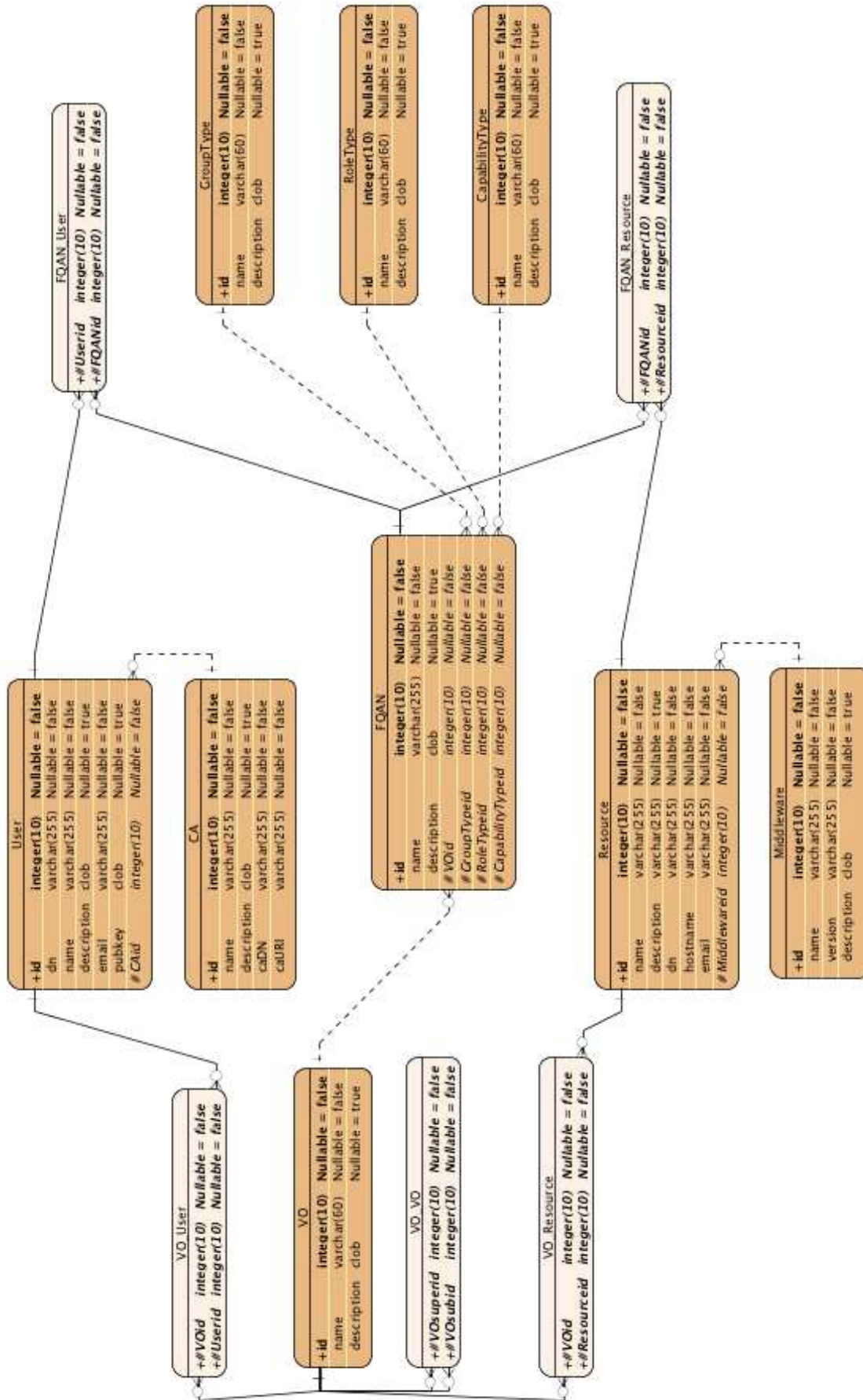


Abbildung 4: Ausschnitt des Datenbankschemas nach [Kirchler]

5.2 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, die die Datenstruktur stets in einem konsistenten Zustand halten. 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 (siehe Abbildung 5)

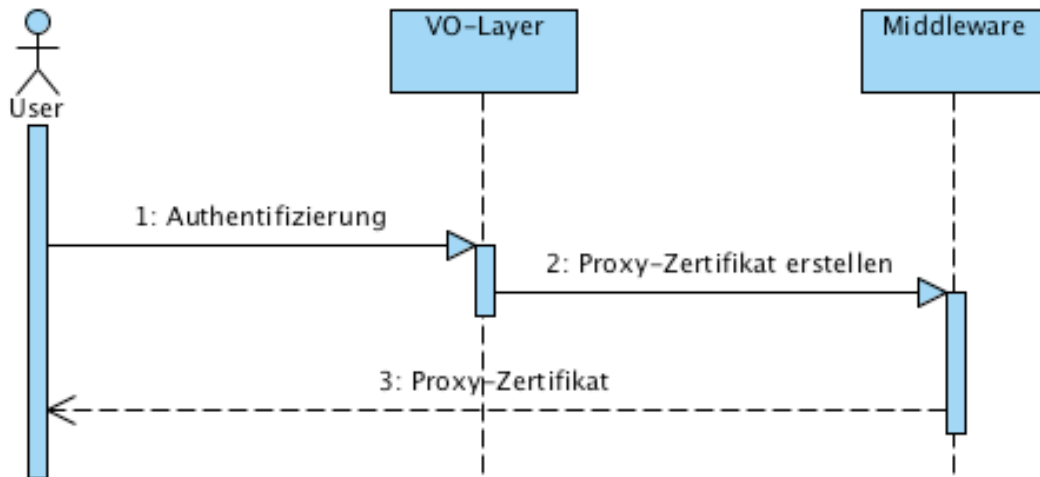


Abbildung 5: Kontrollfluss des Use Cases „Authentifizierung“

Die Autorisierung eines Benutzers für den Ressourcenzugriff läuft ebenfalls über den VO-Layer. Dieser überprüft zunächst die Berechtigungen des Benutzers und leitet im Erfolgsfall den Grid-Job an die jeweilige Middleware-Technologie weiter, wo die eigentliche Autorisierung geprüft und anschließend der Job gestartet wird. In Abbildung 6 ist dieser Kontrollfluss wiedergegeben.

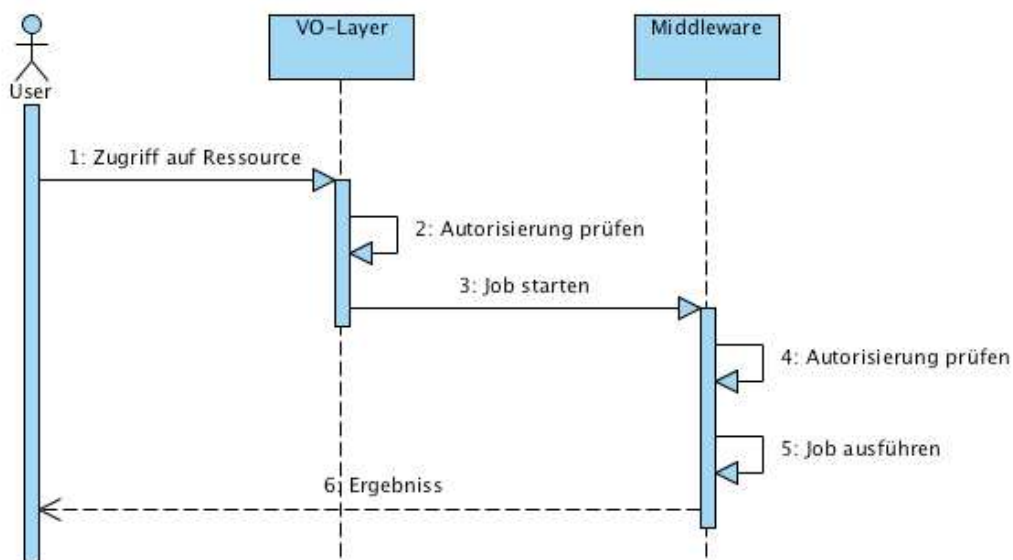


Abbildung 6: Kontrollfluss des Use Cases „Autorisierung“

Die Kooperation der verschiedenen Akteure für die im vorherigen Kapitel beschriebenen Use Cases lassen sich auf ähnliche Weise darstellen. Abbildung 7 stellt den Fall der Aufnahme eines neuen VO-Mitglieds dar.

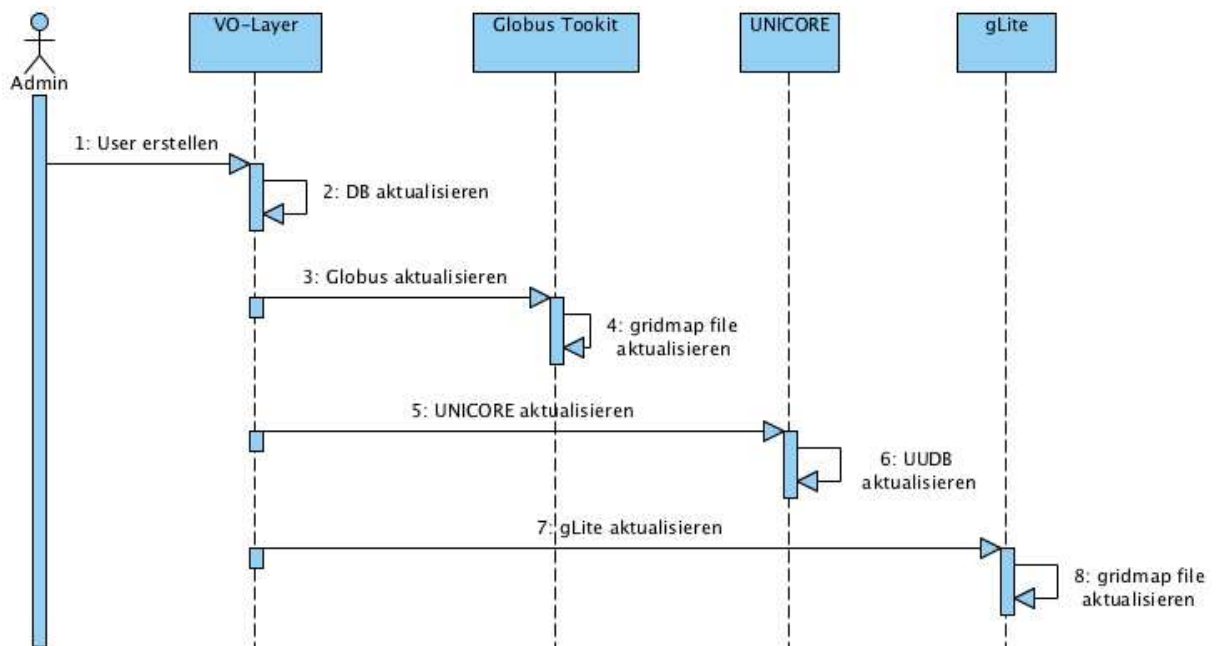


Abbildung 7: Kontrollfluss des Use Cases „Neues Mitglied“

5.3 Softwarearchitektur

Die Softwarearchitektur des VO-Layers basiert auf dem Model-View-Controller (MVC) Architekturstil (siehe Abbildung 8), der aus den drei Komponenten Model, View und Controller besteht.

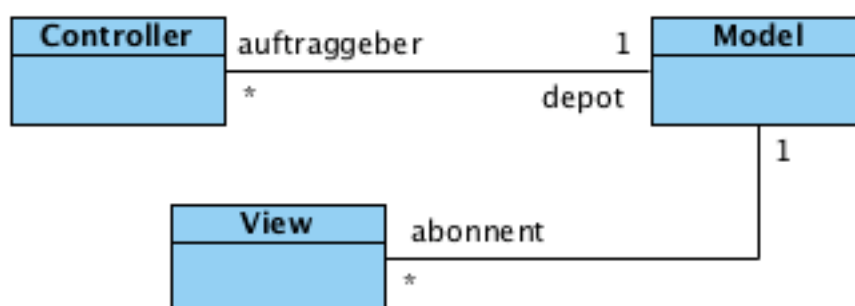


Abbildung 8: MVC Pattern

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 allerdings ständig ändern, wobei auch mehrere *Controller* und *Views* für das gleiche *Model* existieren können. Im nächsten Kapitel wird die prototypische Implementierung des *Models* und des *Controllers* beschrieben, die *View*-Komponente wurde bisher noch nicht implementiert.

5.4 Subsysteme

Nach der Definition der Anforderungen wird der VO-Layer in mehrere Subsysteme zerlegt. Dabei wird Wert darauf gelegt, dass die Kopplung zwischen den Subsystemen möglichst gering ist (siehe Abbildung 9).

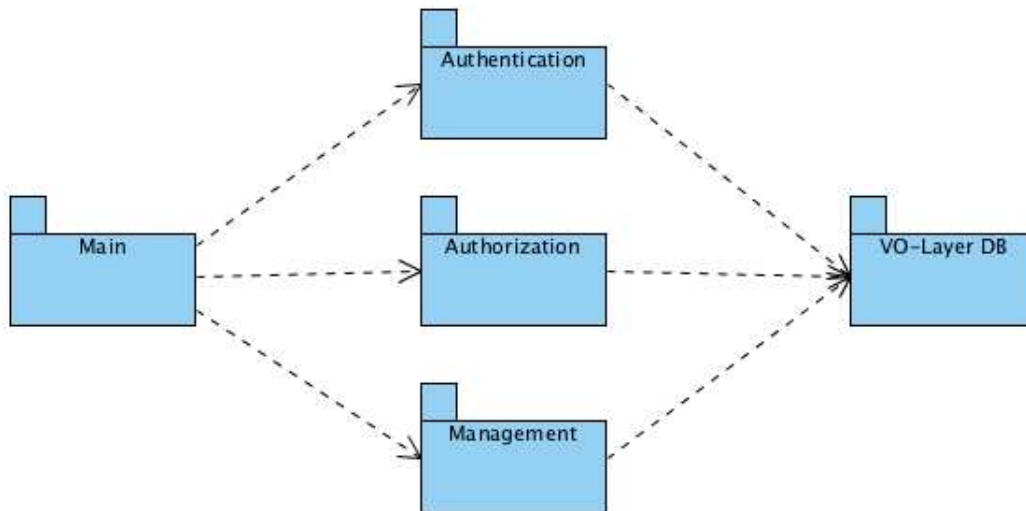


Abbildung 9: Subsysteme

- **Authentication**
Das Authentication-Subsystem leitet sich aus dem Use Case *Authentifizieren* 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* 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 Weiteren 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 Management-Use Cases. 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 gridmap 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 implementierte Objektmodell vorgestellt. Dazu wird auf das Datenmodell des VO-Layers aus Abbildung 3 zurückgegriffen und dieses an die Programmierumgebung angepasst, um die Daten des VO-Layers persistent zu speichern. In Abbildung 10 ist das Objektmodell des VO-Layer Datenmodells wiedergegeben. Dieses wird, mit Hilfe der im VO-Layer DB-Subsystem implementierten Abbildung von einem Objektmodell in ein relationales Datenbankschema übersetzt. Die Methoden zum Ändern der VO-Struktur, welche im ursprünglichen Datenmodell 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, die in den Datenbanktabellen gespeichert werden.

Für eine genauere Spezifikation weiterer Objektmodelle wird auf [kirchler] verwiesen.

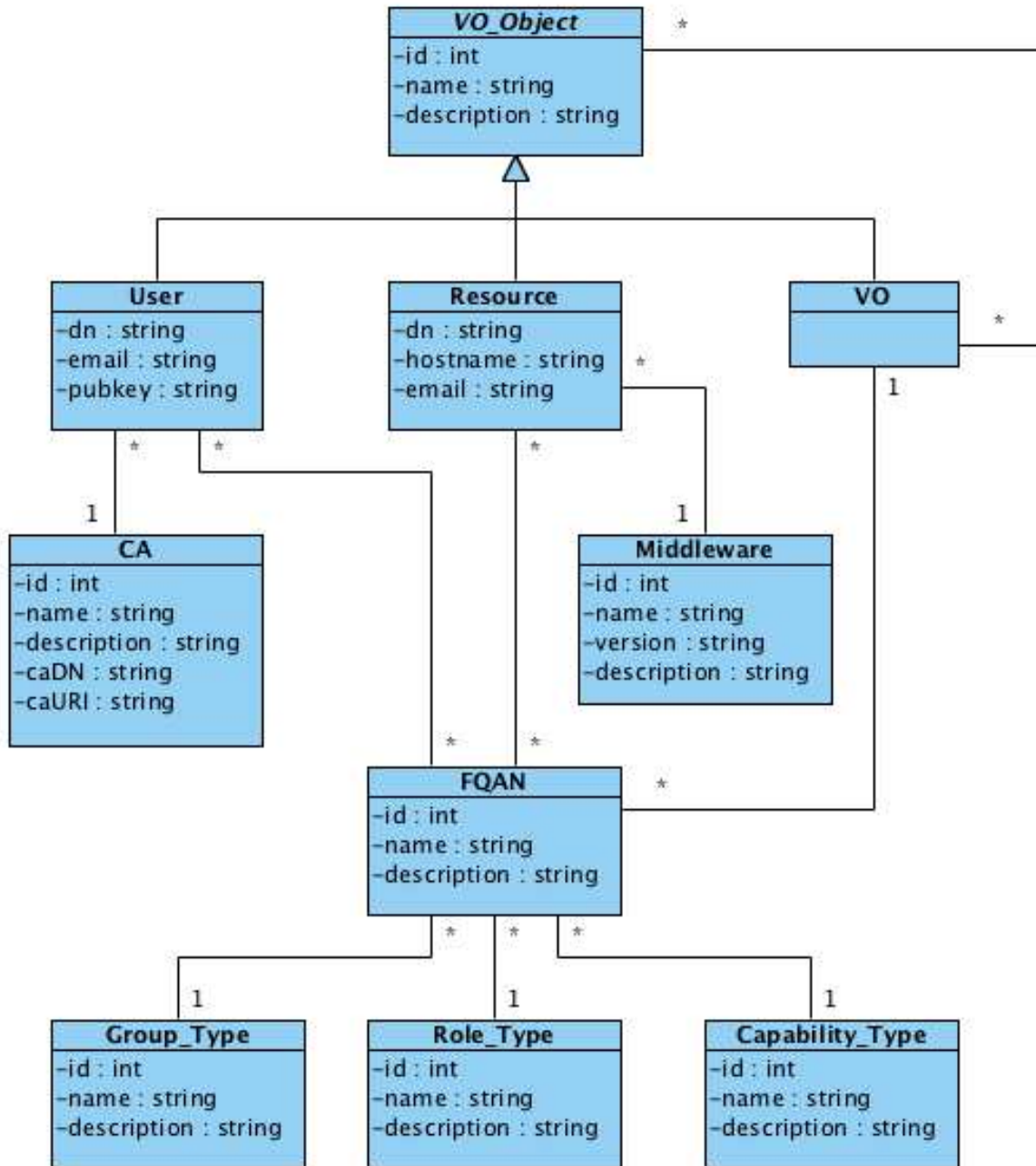


Abbildung 10: Objektmodell VO nach [kirchler]

Insbesondere wird ein Objektmodell für das VO-Layer DB-Subsystem benötigt. Dieses Subsystem ist vollständig von den anderen Subsystemen und der Hauptanwendung abgekoppelt. Initiiert wird es 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 Query-Typ - eine bestimmte Methode aus der VOldbCon-

nection-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 11 den Ablauf einer Datenbankabfrage als Sequenzdiagramm dar.

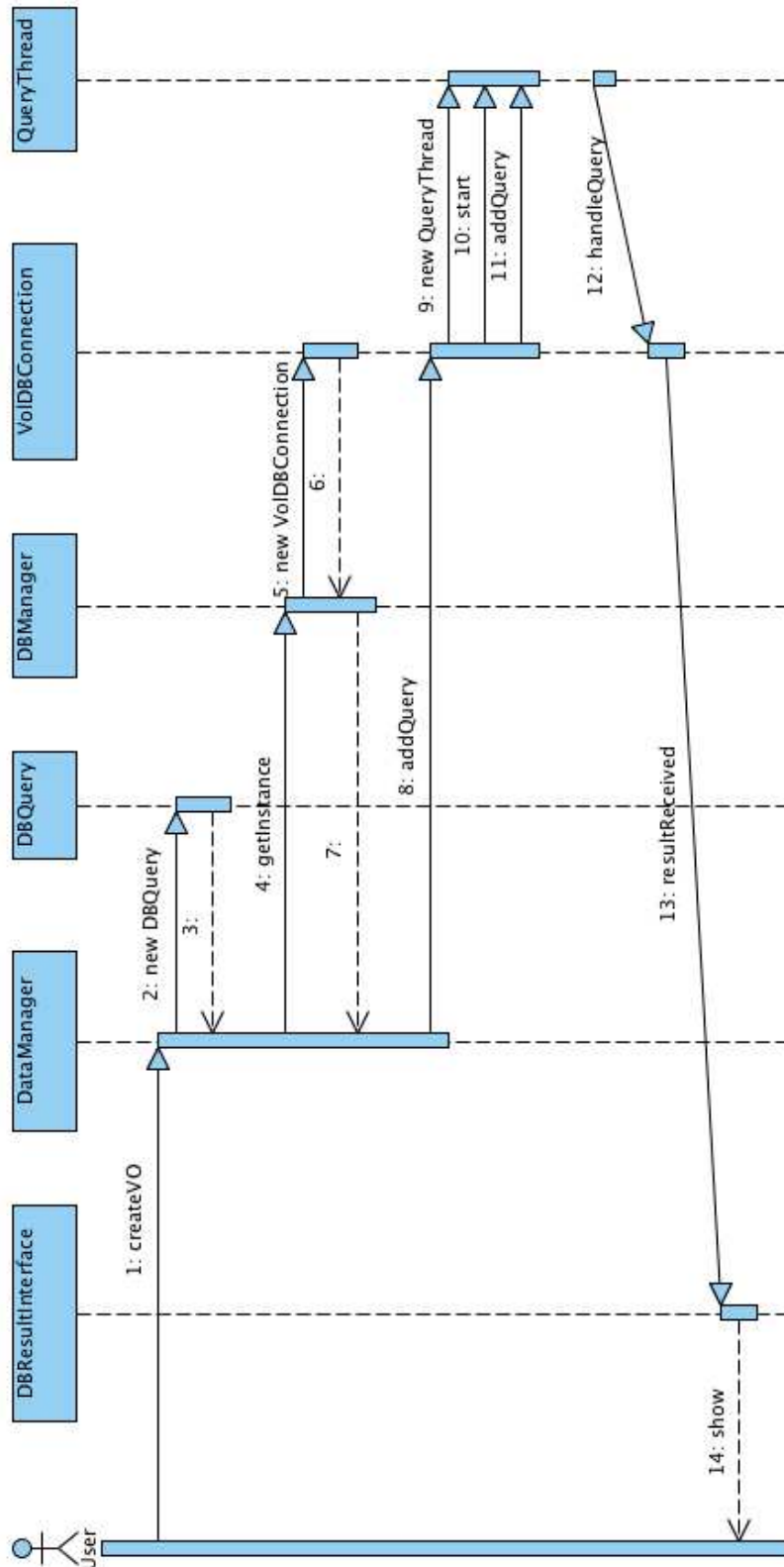


Abbildung 11: DB Access-Sequenz

Dabei wird der Schritt 5 nur durchgeführt falls noch keine `VolDBConnection` erstellt wurde. Die Schritte 9 und 10 werden nur durchgeführt falls noch kein `QueryThread` existiert.

6 Prototypische Implementierung

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.

6.1 Hauptanwendung

Der VO-Layer wird mit der Klasse `Start` ausgeführt. Diese Klasse enthält die statische Methode `main`, 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();
}
```

6.2 Authentifizieren

Um einen Benutzer im Grid zu authentifizieren, wird ein Benutzerzertifikat benötigt. Dieses wird der `authenticate` Methode der Klasse `Authentication` übergeben. 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. Dieser Befehl startet ein Shell-Skript, welches den eigentlichen Grid-Befehl zur Authentifizierung des Benutzers ausführt.

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

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

Es existiert für jede Grid-Middleware ein eigenes Shell-Skript zur Authentifizierung der Benutzer. Für Globus wird das Skript `globus_authenticate.sh` verwendet, für gLite das Skript `glite_authenticate.sh` und für UNICORE das Skript `unicore_authenticate.sh`. 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
```

```
#!/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
```

```
#!/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
```

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.

6.3 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. Sobald das Ergebnis der Datenbankabfrage vorliegt, wird die Methode `resultReceived` ausgeführt, die das Ergebnis der Autorisierung verarbeitet. 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.

Für jede Grid Middleware-Technologie existiert deshalb ein eigenes Skript, welches den angegebenen Job mit dem entsprechenden Grid-Befehl ausführt. Für Globus-Jobs existiert das Skript `globus_run.sh`, welches einen Web-Service Client mit der Web-Service URI des Servers aufruft. Dazu muss der Web-Service Client mit der Java Runtime gestartet werden, als Argument erhält der Client die URL des Servers.

```
#!/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
```

Damit Jobs auf Ressourcen ausgeführt werden können, welche von der gLite Middleware - Technologie bereitgestellt werden, existiert das Skript `glite_run.sh`. Dieses Skript führt ein Job-File auf der gLite Ressource mittels des Befehls `globus-job-run` aus.

```
#!/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
```

UNICORE-Jobs werden mit Hilfe des Skriptes `unicore_run.sh` 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 Benutzers 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 uncore 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
```

6.4 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 Benutzern und Ressourcen neue FQANs zugeteilt oder vorhandene entzogen werden. Die Aufgabe des VO-Layers an dieser Stelle ist es, die gewünschten Änderungen in der VO-Layer DB festzuschreiben und die geänderten Zugriffsberechtigungen an die Middleware-Technologien weiterzuleiten. Das folgende Listing zeigt zwei ausgewählte Methoden der Klasse `VOManagement`, welche dazu verwendet werden können, um die VO-Struktur des VO-Layers zu verändern. Beide Methoden setzen 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);
}
```

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.

- 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 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. Der Nachteil der Lösung 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 zu integrieren ist, wird dieser Ansatz hier weiterverfolgt. Für jede der drei Middleware-Technologien wird eine Methode erstellt, welche aus der VO-Struktur die entsprechenden ACLs erstellt. Im Listing unten sind die öffentlichen Methoden aufgelistet, die das Erzeugen 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 Middleware-spezifische Datenstruktur 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);
}
```

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` und `generate_uudb_file` erstellt. Das gridmap File für Globus und gLite erhält dabei als Dateinamen den Hostnamen des Zielsystems, auf dem es importiert werden muss. 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. 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();
    }
}
```

Für UNICORE muss eine Datei im Comma Separated Value (CSV)-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.

```
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();  
}  
}
```

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 dem 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
```

7 Status und Ausblick

Die Kernkomponenten der Suite sind mit dem VO-Layer komplett (prototypisch) implementiert. Erste Erfahrungen liegen damit vor.

1. Die Authentifizierung ist mittels des VO-Layer einheitlich möglich. Der VO-Layer bietet eindeutige Schnittstellen zur Authentifizierung und leitet die Authentifizierungsanfragen an die entsprechende Grid Middleware-Technologie weiter, die verwendete Grid-Middleware ist transparent.
2. Wie die Authentifizierung ist auch die Autorisierung mittels des VO-Layers über einheitliche Schnittstellen möglich. Nur die Kenntnis dieser Schnittstellen und ihrer Parameter sind erforderlich. Der VO-Layer kapselt die darunter liegende Grid Middleware-Technologie, indem jede Autorisierungsanfrage automatisch an die richtige Grid-Middleware weitergeleitet wird.
3. Das Ziel, alle im D-Grid verwendeten VO-Konzepte mit Hilfe einer generischen VO-Struktur zu unterstützen, wurde schon in [generischevo] erreicht. Mit der dort vorgeschlagenen Struktur ist es möglich, VOs mit beliebig vielen Benutzern, Ressourcen und internen Strukturen zu etablieren. Der hier entwickelte VO-Layer unterstützt die in [generischevo] definierten Strukturen und Prozesse vollständig. Das VO-Management über den VO-Layer kann dabei in zwei Bereiche aufgeteilt werden:
 - a. 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 übernommen.
 - b. 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 und Middleware-spezifisch.

In der prototypischen Implementierung wurden bisher die nicht-funktionalen Anforderungen (siehe Abschnitt 4.5) nur rudimentär berücksichtigt. Hier ist noch Zusatzaufwand nötig bevor das System produktiv gehen kann.

1. Zu einem benutzerfreundlichen System gehört eine ansprechende und leicht zu bedienende grafische Benutzeroberfläche (GUI) sowie eine einheitliche und gut dokumen-

tierte Entwicklerschnittstelle (API). Letztere liegt vor und kann bei der inzwischen begonnenen Entwicklung der noch offenen GUI eingesetzt werden.

2. 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 Prototyp Transaktionen, welche durch das Tabellenformat *InnoDB* Transaktionssicherheit und referenzielle Integrität gewährleisten.
3. In der prototypischen Implementierung wird noch nicht zwischen normalen Benutzern und VO-Administratoren unterschieden.
4. Die Leistung des Systems in Bezug auf Antwortzeit und Durchsatz hängt im Wesentlichen vom verwendeten Datenbanksystem und der verwendeten Hardware ab, genaue Messungen sind hier jedoch noch erforderlich.
5. 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.

Eine weitere offene Fragestellung betrifft den potenziellen Einsatz des VO-Layer-Konzeptes. Dazu ist mit FG2 zu klären, inwieweit das hier vorgestellte System in das D-Grid-Betriebskonzept eingebunden werden kann. Insbesondere ist zu klären, wie die Kooperation/Interaktion mit VOMS, VOMRS und GRRS umgesetzt werden kann.

8 Abkürzungen

ACL	Access Control List
GRRS	Grid Resource Registry Service
GUI	Graphical User Interface
VO	Virtuelle Organisation
VOMRS	Virtual Organization Membership Registration Service
VOMS	Virtual Organization Membership Service

9 Referenzen

- [betriebskonzept] Betriebskonzept für das D-Grid; http://www.d-grid.de/fileadmin/user_upload/documents/Kern-D-Grid/Betriebskonzept/D-Grid-Betriebskonzept.pdf
- [flowcharts] Grimm, Schiffers, Fieseler, Piger, Dohmen: Schematische Darstellung von VO-Management-Workflows im D-Grid; Juni 2008
- [generischevo] Grimm, Henne, Piger, Schiffers, Ziegler: Generische VO-Strukturen für das D-Grid; Bericht Fachgebiet 3.2 AAI/VO; Juli 2008; http://dgi.d-grid.de/fileadmin/user_upload/documents/DGI2-FG3/FG3-2/DGI-2_FG-3.2_Generische_VO-Strukturen_D-Grid.pdf
- [kirchler] Kirchler, W., *Entwicklung einer einheitlichen Autorisierungs- und Authentifizierungsschnittstelle für heterogene Grids am Beispiel D-Grid*, Diplomarbeit Technische Universität München, September 2008, <http://www.nm.ifi.lmu.de/pub/Diplomarbeiten/kirc08>
- [suite] Schiffers, Ziegler: Werkzeugunterstütztes Management virtueller Organisationen im D-Grid; Bericht Fachgebiet 3.2 AAI/VO; Februar 2009; http://dgi.d-grid.de/fileadmin/user_upload/documents/DGI2-FG3/FG3-2/DGI-2_FG-3.2_Werkzeugunterstuetztes_VO-Management.pdf