Fachhochschule Köln
Cologne University of Applied Sciences

Masterarbeit

Integration von VO-Management-Technologien in UNICORE

Erstgutachter: Prof. Dr. Lutz Köhler

Zweitgutachter: Prof. Dr. Erich Ehses

Betreuer: Wolfgang Ziegler

Gummersbach, 06.12.2007

Fraunhofer Institute
Algorithms and
Scientific Computing

von:

Arash Faroughi | Medieninformatik Master | 11034593
Roozbeh Faroughi | Medieninformatik Master | 11034594

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

Foster et al. [FOS01]

Danke….

Ein Dank gebührt den Personen, die uns
in der schwierigen Zeit der Erstellung dieser
Masterarbeit unterstützt haben.

Ganz besonders:
Unseren Gutachtern Prof. Dr. Lutz Köhler
und Prof. Dr. Erich Ehses,

Ein besonderer Dank gebührt unseren
Freunden Robert Sebastian Reith, Marco Hülsmann und
unserem Betreuer im
Fraunhofer Institut SCAI Wolfgang Ziegler

Abstract

Reliable authentication and authorisation are crucial for both service providers and their customers, where the former want to protect their resources from unauthorised access and fraudulent use while their customers want to be sure that unauthorised access to their data is prevented. In Grid environments Virtual Organisations (VO) have been adopted as a means to organise and control access to resources and data based on roles that are assigned to users. Moreover, attribute based authorisation has emerged providing a decentralised approach with better scalability. Up to now UNICORE authentication and authorisation is based on X.509-certificates only. In this master thesis we will present two implementational solutions to integrate both role or attribute based authorisation using VOMS and attribute based authorisation using Shibboleth into UNICORE.

Angewandte Konventionen

In dieser Masterarbeit wurden folgende Konventionen verwendet:

- *Kursiv* wurde verwendet, um Zitate zu kennzeichnen, Eigennamen hervorzuheben und neue Begriffe einzuführen.
- **Fett** kam zum Einsatz, um den Beginn bestimmter, den Hauptkapiteln untergeordneter Teilabschnitte kenntlich zu machen, sowie Abbildungen, Diagramme und Auflistungen zu beschriften.
- `Nichtproportional`-Schrift diente der Kennzeichnung von Programm-Codes und Kommando-Zeilen.

Abkürzungsverzeichnis:

AA	Attribute Authority
AAI	Authentifizierungs- und Autorisierungs- Infrastruktur
ABAC	Attribute-Based Access Control
AJO	Abstract Job Object
AR	Attribute Requester
ARP	Attribute Release Policy
AZ	Attributzertifikat
BZ	Benutzerzertifikat
CA	Certificate Authority
CN	Common Name
CRL	Certificate Revocation List
CS	Credential-Service
DFN	Deutsches Forschungs-Netz
DN	Distinguished Name
EGA	Enterprise Grid Alliance
FIM	Federated Identity Management
FQAN	Full-Qualified Attribute Name
GGF	Global Grid Forum
GRAM	Grid Resource Allocation & Management System
GSH	Grid Service Handle
GSI	Grid Security Infrastructure
GT	Globus Toolkit
HS	Handle Service
IBAC	Identity-based-Access Control
IdM	Identity Management
IdP	Identity Provider
IVOM	Interoperabilität und Integration der VO-Management-Technologien im D-Grid
JPA	Job Preparation Area
MyVocs	Virtual organization collaboration system
NJS	Network Job Supervisor
OGF	Open Grid Forum
OGSA	Open Grid Service Architecture
PDP	Policy Decision Point
PKI	Public Key Infrastructure
RA	Registration Authority

RBAC	Role-based-Access Control
SAML	Security Assertion Markup Language
SDE	Service Data Element
SLC	Short Lived Credential
SLCS	Short Lived Credential Service
SOAP	Simple Object Access Protocol
SP	Service Provider
SSO	Single-Sign-On
TSI	Target System Interface
UNICORE	Uniform Interface to Computing Resources
Usites	UNICORE Sites
UUDB	UNICORE User Database
UUDBVO	UNICORE User Database for Virtual Organisation
VO	Virtuelle Organisation
VOMRS	Virtual Organisation Membership Registration Service
VOMS	Virtual Organization Membership Service
Vsites	Virtual Sites
W3C	World Wide Web Consortium
WAYF	Where Are You From
WSDL	Web Service Description Language
WSRF	Web Service Resource Framework

Inhaltsverzeichnis:

ABKÜRZUNGSVERZEICHNIS: VI

1. EINLEITUNG 1

1.1 IVOM-PROJEKT 1
1.3 GLIEDERUNG 3

2. GRUNDLAGEN 5

2.1 GRID COMPUTING 5
2.1.1 Charakterisierung von Grids 6
2.1.2 Grid-Architektur 9
2.1.3 OGSA und Grid Services 11
2.2 VIRTUELLE ORGANISATION 14
2.2.1 Charakterisierung von Virtuellen Organisationen 15
2.2.2 VO-Struktur 19
2.2.3 Defizite der heutigen VO-Ansätze 19
2.3 PUBLIC KEY INFRASTRUCTURE 21
2.3.1 X.509-Zertifikate 25
2.3.2 Mehrseitige Authentifizierung 27
2.3.3 Proxy-Zertifikate 28
2.3.4 Attributzertifikate 28
2.3.5 Short-lived-Credentials 29
2.4 AUTHENTIFIZIERUNGS- UND AUTORISIERUNGS-INFRASTRUKTUREN 30
2.4.1 Authentifizierung 30
2.4.2 Autorisierung 32
2.4.3 Zugriffskontrolle 33
2.4.4 Mögliche Probleme bei der Verwendung von AAI 35
2.5 FEDERATED IDENTITY MANAGEMENT 36
2.5.1 SAML 38
2.6 GRID - MIDDLEWARE 41
2.6.1 UNICORE 41
2.6.2 Globus Toolkit 44
2.6.3 gLite 46

3. UNICORE AAI UND VO MANAGEMENT 47

3.1 DAS UNICORE-SICHERHEITSMODELL 47

3.1.1 Authentifizierung *47*
3.1.2 Autorisierung *47*
3.1.3 Integrität der Jobs *48*
3.1.4 Single-Sign-On in UNICORE *48*
3.2 Probleme von identitätsbasierter Zugangsentscheidung 49
3.3 VO-Management 51
3.3.1 Anforderungen an ein VO-Management-System *51*
3.4 VO-Management-Systeme 54
3.4.1 Shibboleth *54*
3.4.2 Shibbolisierung des Grids *58*
3.4.3 VO-Management per Shibboleth *58*
3.4.4 MyVocs *60*
3.4.5 GridShib *61*
3.4.6 VOMS *63*
3.4.7 VOMRS *66*
3.5 Motivation für die UNICORE-Erweiterung 67

4. UNICORE-ERWEITERUNG DURCH VO-TECHNOLOGIEN 68

4.1 Anforderungen für die UNICORE-Integration durch VO-Technologien ... 68
4.1.1 Allgemeine Anforderungen an die Integration: *68*
4.1.2 Anforderungen aus Benutzer-Sicht *70*
4.1.3 Anforderungen aus Administrations-Sicht *70*
4.2 Kreativitätsraum der UNICORE-Integration 72

5. KONZEPTION 73

5.1 Allgemeine Integrationsarchitektur 73
5.2 Konzeption der VO-Authentifizierung 75
5.2.1 Begrifflichkeiten der Authentisierung *75*
5.2.2 Benötigte Benutzer-Schritte für den UNICORE-Zugang *79*
5.2.3 Integrationskonzepte für die Authentisierung *80*
5.3 Konzeption der VO-Autorisierung 84
5.3.1 Parteien und Objekte einer VO-Autorisierung *84*
5.3.2 Autorisierung durch VO-Credentials *85*
5.3.3 Kommunikationsstrategien für die verteilte Autorisierung *87*
5.3.4 Pull- oder Push-Ansatz der VO-Credentials *88*
5.3.5 Richtlinien für die verteilte Autorisierung *89*
5.3.6 Die identitätsbasierte UUDB *90*
5.3.7 Konzeption der UUDBVO *91*

5.4 Integration von Shibboleth in UNICORE 95
5.4.1 Einbehaltung der Shibboleth-Kommunikation...95
5.4.2 Integration durch ein externes Modul 98
5.4.3 Integration durch GridShib-CA und UNICORE-Plugin..............................100
5.5 Integration von VOMS in UNICORE..103
5.5.1 Extrahieren der VOMS-Attribute..103
5.5.2 Allgemeine Integrationsarchitektur von VOMS und UNICORE 105
5.5.3 Architekturvorschlag durch AJO-Integration...106
5.5.4 Architekturvorschlag durch einen SLC-Service 108
5.5.5 Architekturvorschlag durch eine Keystore-Erweiterung............................110
5.6 Integration der Shibboleth- und VOMS-Architektur 112
5.7 Interoperabilität der UNICORE-Konzeption...114
5.7.1 Kombination von VO- und Campusattributen 116

6. IMPLEMENTIERUNG 121

6.1 Implementierung der Plugins...121
6.1.1 Verarbeiten der Credential-Dateien..121
6.1.2 Erstellung sowie Erweiterung einer UNICORE-Benutzeridentität..............123
6.2 Implementierung der UUDBVO...127
6.2.1 Bereitstellung einer Autorisierung durch Attribute 127
6.2.2 Administration der Attributliste..131
6.2.3 Verifizieren und Validieren der Credentials 134
6.2.4 Extrahieren der Attribut-Credentials 136
6.2.5 UNICORE-PDP...146

7. BEWERTUNG DER UNICORE-ERWEITERUNG..149

7.1 Bewertung der allgemeinen Anforderungen 149
7.2 Bewertung der Anforderungen der Benutzer...150
7.3 Bewertung der Anforderungen des Administrators 150

8. ZUSAMMENFASSUNG UND AUSBLICK...151

LITERATURVERZEICHNIS XI

INTERNETLINKS.. XIII

ABBILDUNGSVERZEICHNIS XVI

DIAGRAMME XVII

LISTINGS XVIII

1. Einleitung

Seit den 90er Jahren ist das Konzept der *Virtuellen Organisation* (VOs) von fundamentaler Bedeutung für das Grid-Computing und die organisationsübergreifende Bereitstellung komplexer IT-Lösungen. Unter Virtueller Organisation wird ein Zusammenschluss von Firmen, Abteilungen und Personen verstanden, die zeitweise für einen bestimmten Zweck zusammenarbeiten. Diese VOs treffen globale Vereinbarungen mit Ressourcenanbietern, die wiederum Mitgliedern dieser VOs die Nutzung ihrer Ressourcen gestatten (sie dazu autorisieren), basierend auf Attributen, die in der VO für die Mitglieder definiert werden. Durch dieses Konzept entstehen für das Grid-Computing Anforderungen an eine flexible und dynamische Grid-Infrastruktur. Einige Grid-Systeme erfüllen diese Anforderungen jedoch nicht. So zum Beispiel verfolgt *UNICORE* für die Authentisierung und Autorisierung einen rein identitätsbasierten Ansatz. Dieser führt jedoch zu einigen Problemen: Zum einen erfordert dieser Ansatz einen hohen administrativen Aufwand beim Verwalten von Identitätszertifikaten und zum anderen erweist sich eine solche Autorisierung als nicht skalierbar, da für die Abbildung von Zertifikaten zu einem lokalen Account die Identitäten aller Benutzer bekannt sein müssen. Aufgrund dieser Probleme wird in UNICORE eine Autorisierung angestrebt, die auf Rollen bzw. Attributen basiert.

In den meisten *Communities* wird einer von zwei verschiedene VO-Ansätzen verfolgt: Einige wie z.B. die Hochenergiephysik bevorzugen ein zentrales Management von VOs (*VOMS*), andere Communities wie z.B. die Klimaforschung streben einen dezentralen *shibbolisierten*[1] VO-Ansatz an. UNICORE ist von den gängigen Middleware-Systemen das einzige, welches noch keine Integration mit Shibboleth oder VOMS anbietet. Deswegen soll UNICORE im Rahmen des D-Grid IVOM-Projektes um VO-Autorisierungsmechanismen erweitert werden, die speziell die Integrationen mit VOMS und Shibboleth gewährleisten. Die vorliegende Arbeit hat das Ziel, diese Integration zu realisieren und ist Bestandteil des IVOM-Projektes, das im Folgenden vorgestellt wird.

1.1 IVOM-Projekt

IVOM steht für die *„**I**nteroperabilität und Integration der **VO**-**M**anagement-Technologien im D-Grid“* und ist ein Projekt, das im Rahmen der *D-Grid-Initiative* [1] vom Bundesministerium für Bildung und Forschung gefördert wird. Die D-Grid-Initiative hat das Ziel, eine nachhaltige Grid-Infrastruktur für *e-Science* in Deutschland zu etablieren. D-Grid umfasst momentan *Community-Projekte* aus verschiedenen Wissenschafts-

[1] shibbolisiert: abgeleitet von Shibboleth

bereichen, wie zum Beispiel Astrophysik, Klimaforschung, Ingenieurwissenschaften, Medizin, Hochenergiephysik und Textwissenschaften. Die Community-Grids dieser Projekte verwenden unterschiedliche heterogene VO-Management- und Grid-Technologien. Deswegen verfolgt IVOM das Ziel, eine Interoperabilität zwischen den verschiedenen Management-Systemen zu gewährleisten. Außerdem sollen Voraussetzungen für eine Zusammenarbeit mit vergleichbaren Communities im internationalen Rahmen geschaffen werden [NEU07].
Zwei Schwerpunkte lassen sich beim IVOM-Projekt kategorisieren:

- Entwurf einer allgemeinen VO-Management-Infrastruktur für das D-Grid. Dafür werden die Lösungen von internationalen Projekten bewertet und mit den Anforderungen der deutschen Grid-Communities verglichen [NEU07].
- UNICORE wird durch VO-Autorisierungsmechanismen erweitert. Es soll dadurch die Autorisierung anhand von Shibboleth- und VO-Attributen durchführen können.

Im IVOM-Projekt sind folgende Partner-Organisationen beteiligt:

- Alfred-Wegener-Institut für Polar- und Meeresforschung in der Helmholtz-Gemeinschaft
- DFN Verein (assoziiert)
- Forschungszentrum Jülich (assoziiert)
- Fraunhofer Institut Algorithmen und Wissenschaftliches Rechnen SCAI
- Leibniz Rechenzentrum der Bayerischen Akademie der Wissenschaften (LRZ)
- Regionales Rechenzentrum für Niedersachsen (RRZN) und Forschungs-zentrum L3S
- Universität Göttingen (assoziiert)
- Gewerblich:
    - DAASI International GmbH
    - Sun Microsystems GmbH (assoziiert)

Die Projektlaufzeit von IVOM ist von Oktober 2006 bis März 2008 befristet.

1.2 Aufgabenstellung und Zielsetzung

Ziel der Masterarbeit ist es, UNICORE der Version 5 um VO-Autorisierungsmechanismen zu erweitern, um eine Integration von VOMS und Shibboleth in UNICORE zu realisieren. Die identitätsbasierte Autorisierung von UNICORE soll durch eine attributbasierte ersetzt werden. Dadurch erfolgt eine Autorisierungsentscheidung nicht mehr auf der Basis der Identität, sondern anhand von VOMS- und Shibboleth-Attributen erfolgen.

Im Gegensatz zu anderen Grid-Middleware-Systemen, die bisher nur eine attributbasierte Autorisierung durch Campus- oder VO-Attribute erteilen, soll UNICORE im Rahmen der Masterarbeit das erste Grid-Middleware-System sein, dass Autorisierungsentscheidungen durch die Kombination *beider* Attributformen trifft.

In der Grid-Umgebung ist für den Benutzer eine Kombination von VOMS und Shibboleth denkbar, was durch diese Arbeit ermöglicht werden soll.

Darüber hinaus soll sich die Integration nicht nur auf die Technologien VOMS und Shibboleth beschränken, sondern soll eine Interoperabilität zu anderen VO-Management-Systemen gewährleisten. Somit könnte eine Erweiterung von UNICORE um VO-Management-Systeme wie MyVocs oder VOMRS ohne großen Aufwand möglich sein.

Durch diese Ziele, welche Gegenstand der Arbeit sind, könnte UNICORE verglichen mit den anderen gängigen Grid-Middleware-Systemen wie Globus Toolkit 4 oder gLite die flexibelste attributbasierte Autorisierung zur Verfügung stellen.

Desweiteren wäre mit dieser UNICORE-Implementierung eine Möglichkeit geschaffen, ein übergreifendes VO-Management anzubieten, das auch unabhängig vom jeweilig verwendeten Grid-Middleware benutzt werden kann.

Die Integration in UNICORE soll durch Plugins und veränderbare Komponenten erweitert werden. Es soll eine möglichst problemlose Einfügung in bereits bestehende UNICORE-Systeme möglich sein. Ende des Jahres 2007 soll die Implementierung für die D-Grid-Communities im Webportal SourceForge des UNICORE Projekts [2] [3] zur Verfügung gestellt werden.

1.3 Gliederung

Zunächst beschäftigt sich *Kapitel 2* mit den theoretischen und technologischen Grundlagen, die im weiteren Verlauf der Arbeit benötigt werden. Dazu werden die Begriffe Grid-Computing und Virtuelle Organisation definiert und die Grundlagen verschiedener Authentifizierungs- und Autorisierungs-Infrastrukturen beschrieben.

Kapitel 3 erläutert das UNICORE-Sicherheitsmodell und geht dann auf die Probleme des identitätsbasierten Ansatzes ein, die UNICORE für die Autorisierung von Benutzern verwendet. Zum Schluss werden verschiedene VO-Technologien beschrieben.

Im *Kapitel 4* werden Anforderungen für die UNICORE-Erweiterung durch VO-Technologien vorgestellt, die bei der späteren Konzeption sowie Implementierung berücksichtigt wurden.

Kapitel 5 beschreibt die Konzeption der UNICORE-Integrationen. Dabei werden die Bereiche der Authentifizierung und Autorisierung getrennt behandelt. Zu dem jeweiligen Bereich werden die zunächst relevanten VO-Konzepte vorgestellt bzw. entwickelt. Anschließend wird beschrieben, wie man diese Konzepte in eine UNICORE-Umgebung integrieren kann. Zum Schluss geht das Kapitel auf die konkreten Integrations-Konzepte von UNICORE und VOMS sowie UNICORE und Shibboleth ein.

Die Implementierung der Integrationskonzepte wird in *Kapitel 6* beschrieben. Auch hier werden die Bereiche der Authentisierung und Autorisierung getrennt behandelt. Zum Schluss verdeutlicht das Kapitel, welche administrativen Schritte durchgeführt werden müssen, um ein bestehendes UNICORE-System mit den Integrations-Implementierungen zu erweitern.

Im abschließenden *Kapitel 7* erfolgt zunächst eine kurze Zusammenfassung der Arbeit. Im zweiten Teil des Kapitels werden die weiterführenden Arbeiten in einem Ausblick beschrieben.

2. Grundlagen

Dieses Kapitel beschreibt Grundlagen und Technologien, die in dieser Masterarbeit verwendet werden und für den Verlauf der weiteren Kapitel notwendig sind. Zunächst wird auf die Begrifflichkeiten und Charakteristiken des Grid-Computings sowie Virtuelle Organisationen eingegangen. Anschließend werden die Grundlagen zu den Infrastrukturen Public-Key-Infrastruktur sowie Authentifizierungs- und Autorisierungs-Infrastrukturen beschrieben. Zum Schluss werden das Federated Identity Management und die gängigen Grid-Middleware-Systeme näher erläutert.

2.1 Grid Computing

Im Hinblick auf die stagnierende Wirtschaft verfolgen Unternehmen immer mehr das Ziel, Investitionen zu reduzieren, aber trotzdem den Umsatz zu steigern. Die Globalisierung ist der „Trendsetter" der Wirtschaft. Durch Auslandsinvestitionen sowie die Bildung von strategischen Partnerschaften wird versucht dieses Ziel zu erreichen. Folgendes belegt diesen Trend: Die Zahl der getätigten ausländischen Direktinvestitionen hat sich von 13 Milliarden US-Dollar im Jahr 1970 über 208 Milliarden US-Dollar 1990 auf 648 Milliarden US-Dollar im Jahr 2004 erhöht [BPB04] . Auch in der Forschung ist die Globalisierung ein wichtiger Begriff. Anstatt zu konkurrieren, versucht man, immer mehr Wissen über Länder- und Institutsgrenzen hinaus zu teilen sowie gemeinsam zu erlangen. Man spricht hierbei von der Bildung von *scientific communities*. Vor allem in Bereichen wie Medizin, Astronomie, Geophysik, Chemie und Biologie entsteht zunehmend die Notwendigkeit größere Datenmengen in kürzester Zeit zu verarbeiten, was mit heutiger Technik nur durch gemeinsame Anstrengungen auf gebündelten, gemeinsam genutzten Ressourcen realisiert werden kann. Diesen Trend, ob in Wirtschaft oder Forschung, kann man mit drei Worten zusammenfassen: *„Bündelung verteilter Ressourcen"*. Informatik wird hier benötigt, um geeignete IT-Infrastrukturen zu entwickeln. In modernen Unternehmen ist die IT-gestützte Informationsbereitstellung von höchster Geschäftsrelevanz. In der Forschung benötigt man hingegen IT-Infrastrukturen, die eine sichere Kommunikation und effizientes Rechnen ermöglichen. Diesen fließenden Übergang von Wirtschaft und Forschung zur Informatik kann man mit folgenden Schlagwörtern beschreiben: Von *Outsourcing* zu *IT-Outsourcing*; von *Communities* zu *Virtuellen Organisationen*; von *zentral organisierten Infrastrukturen* zu *verteilt organisierten IT-Infrastrukturen*.

Der Begriff *„Grid"* wurde erstmals im Buch *„The Grid: Blueprint for a New Computing Infrastructure"* [FOS98] verwendet und als Analogon zum Stromnetz (engl. *Power Grid*) eingeführt. Ein Grid ist eine netzwerkbasierte IT-Infrastruktur, die das Ziel hat, nicht genutzte Ressourcen anderer Rechner zu verwenden [PIU05], da die Auslastung eines Prozessors im Schnitt nur 10 - 30% beträgt [POL03]. Die Analogie zum Stromnetz verdeutlicht, dass die Verteilung von Ressourcen einfach und transparent sein soll. Der Grid-Nutzer benötigt idealerweise keine technischen Details oder Kenntnisse über Herkunft und Erzeugung, um Ressourcen verwenden zu können. Unter *Grid-Ressourcen* werden meistens Rechner-Ressourcen jeglicher Form verstanden. So können zum Beispiel Soft- und Hardware, Supercomputer, Vektor-, Parallelrechner und Cluster [TAN07] als Grid-Ressourcen eingesetzt werden. Grid Computing kommt vor allem bei Wissenschaftlern zum Einsatz, die an Aufgaben arbeiten, welche hohe Rechenkapazitäten benötigen [POL03].

2.1.1 Charakterisierung von Grids

Grids können charakterisiert werden anhand der Komplexität sowie des Typs. Deswegen werden zunächst die drei Komplexitäts-Stufen [TAN07] des Grid-Computings vorgestellt und anschließend werden die verschiedenen Grid-Typen näher beschrieben.

Komplexitätsstufen des Grids :

- **Stufe 1: Cluster**
  Die einfachste Form eines Grids ist der Aufbau eines *Clusters*. Die Rechner, die miteinander lokal verbunden werden, sind typischerweise homogen. Sie verwenden also die gleiche Hardware sowie das gleiche Betriebssystem. Solche Grids werden meistens für das Testen von Grid-Anwendungen eingesetzt. In Forschungs-Instituten kommen Cluster-Grids oft zum Einsatz, um größere Datenmengen durch paralleles Rechnen in Form von Batch-Jobs abzuarbeiten. Anwendungen, wie zum Beispiel *PBS* (Portable Batch System) sowie *Maui Scheduler*, werden verwendet, um Jobs und Ressourcen zu verwalten.

- **Stufe 2: Intragrid**
  Intragrids sind Vernetzungen von mehreren Cluster-Grids oder heterogenen Rechnern. Die Anforderungen an die Ressource-Verwaltung sowie die Komplexität des Grids sind dabei höher als bei der Cluster-Grid-Lösung. Ein

Beispiel hierbei stellt die Vernetzung von Rechnern mehrerer Abteilungen innerhalb eines Unternehmens dar.

- **Stufe 3: Intergrid**
  Intergrids stellen die komplexeste Form von Grids dar. Die Grid-Ressourcen können bei dieser Komplexitätsstufe geographisch weit auseinander und in unterschiedlichen administrativen Domänen liegen. Dabei können Ressourcen selbst Grid-Systeme einfacher Art sein, wie zum Beispiel Cluster-Grids. Die Anforderungen an Ressource-Scheduling und Sicherheit nehmen verglichen mit den vorher genannten Stufen weiter zu.

Verschiedene Grid-Typen:

Ursprünglich beschrieb man Grid Computing als eine Erweiterung des *Parallel Computing-Paradigmas*: Von *fest gekoppelten Cluster-Systemen* zu *geographisch verteilten Systemen* [LIN03]. In der Praxis hat sich jedoch gezeigt, dass sich die ursprüngliche Bedeutung des Grid-Computings ausgeweitet hat. Grid Computing wird heute als Lösung verschiedener Problemen in verteilten Systemen angesehen. Folgende Grid-Typen haben sich entwickelt [LIN03] [PIO05]:

- **Computational Grid**
  Als *Computational Grid* bezeichnet man den Verbund von Computer-Ressourcen, um die Gesamtrechenleistung zu erhöhen.

- **Datagrid**
  Datagrid verwaltet den Zugriff auf verteilte Datenbanken. Durch den Einsatz von Datagrids können zum Beispiel neue Informationen aus den bereits bestehenden Daten bestimmt werden.

- **Service Grid**
  In diesem Modell werden Leistungen des Grids in Form von Services angeboten. Gerade wenn Unternehmen oder einzelne Personen nicht über die benötigten finanziellen Mittel verfügen, um eigene Grids zu bilden, können sie durch *Service-Grids*, die zum Beispiel von einer Firma angeboten werden, Grid Computing nutzen.

- **Distributed Supercomputing**
  Bei *Distributed Supercomputing* (auch *Meta-Computing* genannt) werden Supercomputer zu einem Grid verbunden. Diese Lösung ist für Anwendungen sinnvoll, bei denen ein großer Bedarf an Speicher und CPU besteht.

- **High-Throughput-Computing:**
  High-Throughput-Computing kommt zum Einsatz bei Berechnungen mit hohen Anforderungen an Durchsatz und insgesamt hohem Zeitaufwand. Beispiele hierfür sind Vorgänge, die Rechenzeiten von Monaten in Anspruch nehmen. SETI@home[2] [4] ist zum Beispiel ein Projekt, in dem High Throughput Computing angewendet wird.

- **Desktop-Supercomputing**
  Desktop Supercomputing versucht, dem Benutzer den Zugang zu Super-Computern zu erleichtern. So kann er von einem Desktop-Rechner aus Jobs auf einem entfernten Supercomputer ausführen. Hierbei braucht der Benutzer keine weitreichenden Kenntnisse über System-Konfiguration.

- **Data intensive Computing**
  Data intensive Computing bedeutet die Integration von Daten aus multiplen, verteilten Datei-Systemen. Dabei entstehen hohe Ansprüche an die Rechenleistung sowie an die Kommunikationsleistung.

- **Collaborative Environments**
  Collaborative Environments verbinden Computer- sowie *Human Resources*, um das kollaborative Arbeiten zu gewährleisten. Diese Form von Grid-Computing ist sehr eng mit dem Konzept von Virtuellen Organisationen verbunden (siehe Unterabschnitt 2.2). Es wird hierbei eine Echtzeit-Interaktion zwischen Nutzern und Applikationen auf virtuellem Arbeitsfeld ermöglicht.

- **On-Demand Computing:**
  Wenn spezielle Berechnungen nur für eine kurze Zeit durchgeführt werden müssen, ist es nicht kosteneffektiv, die Ressourcen lokal zu installieren. *On-Demand Computing* bedeutet, dass man nur die benötigten Ressourcen zur

[2] SETI@home ist die Abkürzung für "Search for extraterrestrial intelligence at home" und wurde von der Universität Berkeley entwickelt, um nach außerirdischem Leben zu suchen. Das wissenschaftliche Experiment verfolgt das Ziel, unbenutzte Ressourcen von privaten Rechnern, die über das Internet verbunden sind, einzusetzen.

Verfügung stellt. Hierdurch erhoffen sich Unternehmen eine optimale Ressourcenverwaltung. Ressourcen wie CPU-Leistung oder Speicherplatz sollen also nur dann verwendet werden, wenn sie wirklich gebraucht werden.

2.1.2 Grid-Architektur

Seit den 90er Jahren ist das Konzept der Virtuellen Organisation von zentraler Bedeutung für das Grid-Computing. Foster und Kesselmann beschreiben das Grid als das Zusammenwirken vieler Ressourcen in einem organisatorisch übergreifenden Netz [FOS04]. Grid-Computing habe die zentrale Aufgabe, die verteilte Benutzung von Ressourcen innerhalb einer Teilnehmergruppe zu verwalten [TAN07]. Ihre Mitglieder können Benutzer oder Provider sein. Durch den Ansatz des Grid-Computings entstehen Anforderungen an eine flexible und dynamische Grid-Infrastruktur. Foster und Kesselmann präsentieren in ihrem Buch *„The Grid 2: Blueprint for a New Computer Infrastructure"* [FOS04] eine Grid-Architektur, welche an die neuen Anforderungen des Grid-Computings angepasst ist. Abbildung 1 veranschaulicht diese Architekturvorstellung:

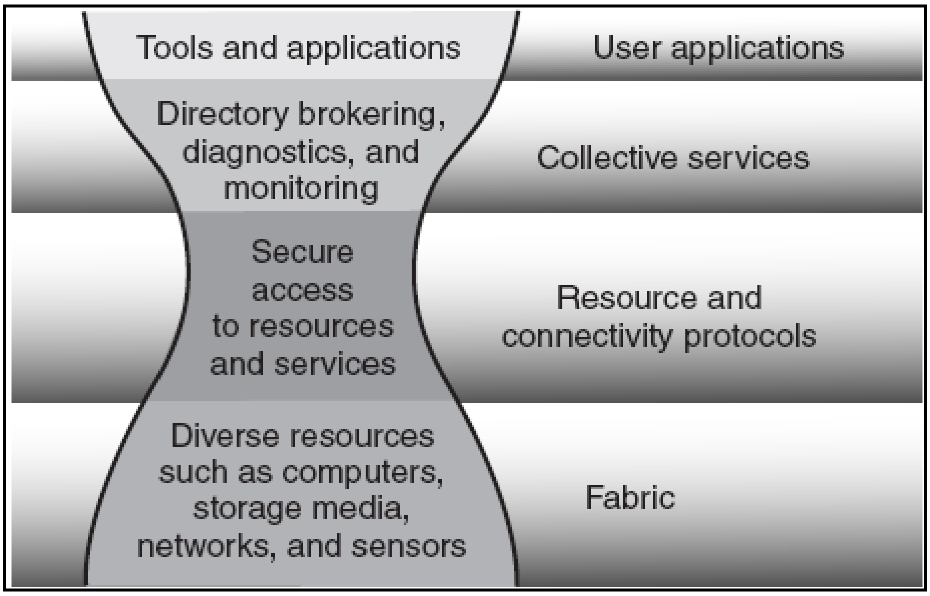

Abb.1: Das Schichtenmodell nach Foster et al [FOS04]

Wie man aus der obigen Abbildung erkennen kann, basiert die Architektur auf dem Sanduhr-Prinzip. Der Sanduhr-Hals wird als *Core Service* definiert und beinhaltet eine Menge von *„core abstractions"* und Protokollen. Er besteht aus Verbindungs- und Ressource-Protokollen. Die oberen Schichten entsprechen den *High-Level Services*, die von globaler Natur sind und die untere Schicht wird durch *Local Services* dargestellt.

Die Architektur ist aufgeteilt in vier verschiedene Schichten:

- Die **Fabrik-Schicht** ist im Allgemeinen für die Verwaltung der lokalen Ressourcen zuständig. Die Ressourcen können logisch oder physikalisch sein. Beispiele für Ressourcen sind: Rechner-, Speicher-, Netzwerkressourcen, Kataloge und Sensoren. Die Schicht stellt Abfrage- und Ressource-Management-Mechanismen bereit. Es sollen unter anderem folgende Operationen möglich sein: Starten und Ausführung von Programmen, Monitoring und Controlling der Rechner-Ressourcen, Steuerung des Netzwerks sowie Anfrage und Aktualisierung der Kataloge [TAN07]. Höhere Schichten werden direkt von den Funktionen der Fabrikschicht beeinflusst.

- Die **Verbindungsschicht** definiert Kommunikations- und Authentifizierungs-protokolle, die für Grid-spezifische Netzwerk-Transaktionen benötigt werden. Die Kommunikationsprotokolle ermöglichen den Austausch von Daten mit den Ressourcen der Fabrik-Schicht. Zum Beispiel sollen Aufgaben wie Routing, Transport und Namensauflösung mit den Kommunikationsprotokollen durchgeführt werden. Die Authentifizierungsprotokolle werden für den Einsatz von kryptografischen Sicherheitsmechanismen verwendet, um die Identität der jeweiligen Benutzer und/oder Ressourcen zu verifizieren. Diese Mechanismen sollen unter anderem *Single-Sign-On* sowie *Delegation* (siehe Unterabschnitt 3.4.1 und 2.3.3) unterstützen.

- Die **Ressource-Schicht** hat die Aufgabe, den Grid-Benutzern Interaktions-Möglichkeiten auf entfernten Ressourcen und Services anzubieten. Sie baut auf den Protokollen der Verbindungsschicht auf und definiert sichere Protokolle für Initialisierung, Monitoring, Controlling, Buchhaltung sowie Zahlung für die jeweiligen Ressourcen. Die Protokolle der Ressource-Schicht können in zwei verschiedene Klassen unterteilt werden:

  - *Informationsprotokolle* werden verwendet, um Informationen über Struktur und Zustand der Ressourcen zu liefern.

- *Management-Protokolle* werden eingesetzt, um den Zugang zu den Ressourcen zu verwalten.

Diese Protokolle rufen die lokalen Funktionen der Fabrikschicht auf.

- Die **Kollektivschicht** ist für die Koordinierung von multiplen Ressourcen zuständig. Im Gegensatz zur Ressourceschicht beziehen sich die Protokolle der Kollektivschicht nicht auf spezifische Ressourcen, sondern bieten Interaktionsmöglichkeiten mit einer Menge von Ressourcen. Die Methoden dieser Schicht können an die spezifischen Anforderungen einer Virtuellen Organisation angepasst sein, sich aber auch auf allgemeine Grid-spezifische Anforderungen beziehen, wie zum Beispiel auf Verzeichnisservices, Koallokation, Scheduling, Brokering, Monitoring, Daten-Replikation, Management des Arbeitsumfangs, Auffinden der Software oder Autorisierung der Gemeinschaften und Kontenverwaltung [TAN07].

- Zuletzt folgt die **Anwendungsschicht**. Sie fasst die Tools und Anwendungen innerhalb der Umgebung einer Virtuellen Organisation zusammen. Diese sind oft hoch entwickelte Systeme wie *Frameworks* und Bibliotheken [TAN07].

2.1.3 OGSA und Grid Services

Seit den 90er Jahren wird in den Organisationen und Konsortien versucht, eine Grid-Architektur zu konzipieren, die auf offenen Standards beruht. Bereits im Jahr 2002 wurde vom *Global Grid Forum* (GGF)[3] die service-orientierte Architektur *Open Grid Service Architecture* (OGSA) als Standardisierungsvorschlag vorgelegt. Als Basis für zukünftige Standardisierungen wurde vom OGF das *Web Service Resource Framework* (WSRF) gewählt. Dadurch wurden *Grid-Services* in das allgemeine Web-Service-Framework des *World Wide Web Consortium* (W3C) eingebettet [SCH07].

Unter Grid-Services werden alle Services verstanden, die im Rahmen von Grid-Computing eingesetzt werden. Durch die Entscheidung zugunsten von WSRF kann man Grid-Services als Web Services betrachten. Trotzdem gibt es Anforderungen, die klassische Web Services nicht erfüllen. Diese sind unter anderem [COJ07]:

[3] Seit 2006 haben sich GGF und *Enterprise Grid Alliance* (EGA) zum Open Grid Forum zusammengeschlossen.

- **Kurzlebige Grid-Services:**
  Grid Services werden über Instanzen verwaltet und sind im Gegensatz zu klassischen Web Services nicht persistent. Wenn der Dienst nicht mehr benötigt wird, wird die Instanz gelöscht.

- **Services mit Zustandsspeicherung:**
  Web Services sind in der Regel zustandslos. Da beim Grid-Computing das Speichern von Zwischenergebnissen für das spätere Rechnen wichtig ist, sollen Grid-Services zustandsbehaftet sein.

- **Eindeutige Adressierung:**
  Aufgrund der Tatsache, dass Grid-Services durch Instanzen erzeugt werden, besteht die Anforderung, dass jeder Grid-Service eindeutig adressiert werden muss.

- **Lebenszyklusmanagement:**
  Der Lebenszyklus von Grid-Services muss verwaltet werden können.

In OGSA werden die Grid-Funktionalitäten als Web Services angeboten. Da Grids im klassischen Sinne wegen fehlender Serviceorientierung nicht so flexibel sind wie Web Services, aber dafür im Gegensatz zu diesen sichere und zuverlässige Übertragung gewährleisten, werden die Vorteile von Grid und Web Services in OGSA kombiniert [TAN07]. Folgende Abbildung illustriert das OGSA-Konzept:

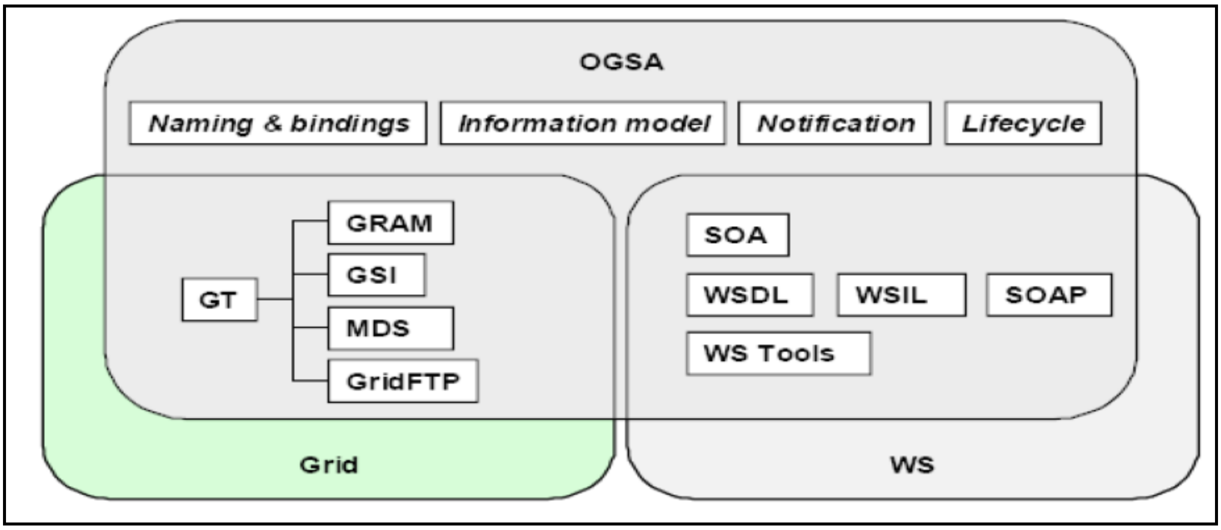

Abb.2: OGSA-Konzept [TAN07]

Für Grid-Computing bieten Web Services geeignete Standards, wie zum Beispiel *SOAP* (Simple Object Access Protocol) und *WSDL* (Web Service Description Language), da sie vor allem in einer heterogenen Umgebung eingesetzt werden können. Die Benutzung von SOAP verspricht Interoperabilität und für die Service-Beschreibungen wird WSDL eingesetzt. Die obige Abbildung zeigt auf der Seite des Grids das Globus Toolkit 4 (GT4), die erste Implementierung der OGSA. GT4 verwendet Protokolle für Ressourcen-Allokation, Informationsmanagement und Sicherheit und wird in diesem Konzept als Basis für Grid-Services dargestellt. OGSA kann als eine Vereinigung von Grid- mit Web-Services verstanden werden. OGSA hat folgende Zusatzkomponenten:

- **Naming and Bindings:**
  Wie schon bei den Anforderungen für Grid-Services beschrieben, sind diese Services kurzlebig. OGSA bietet Konzepte für dynamische Erzeugung von Grid-Service-Instanzen sowie deren Referenzierung. Dafür wird das *Grid Service Handle* (GSH) eingesetzt, das die Instanzen eindeutig kennzeichnet.

- **Information model:**
  Durch das OGSA-Konzept *ServiceData* können die Eigenschaften und Zustandsdaten der Service-Instanzen in so genannten *ServiceData Elements* (SDEs) abgespeichert werden. Hiermit wird die Anforderung „*Services mit Zustandsspeicherung*“ erreicht.

- **Notification:**
  Das *Notification*-Konzept sorgt dafür, dass zwei Services asynchron Nachrichten austauschen können.

- **Lifecycle:**
  Für OGSA wurde ein Lifecycle-Management entwickelt, das die Grid-Ressourcen, die nicht verwendet werden, wieder frei gibt.

2.2 Virtuelle Organisation

Unter „Virtuelle Organisation“ (VO) wird ein Zusammenschluss von Firmen, Abteilungen und Personen verstanden, die zeitweise für einen bestimmten Zweck zusammenarbeiten. Der Begriff erhielt durch das im Jahre 1992 erschienene Buch „The Virtual Corporation“ von Davidow und Malone seinen Durchbruch [MUE97]. Durch die Virtualität spielt der physikalische Standort der Organisation keine Rolle. Die entscheidenden Vorteile gegenüber traditionellen Organisationen sind die Flexibilität sowie der Wegfall von Raum-, Organisations- bzw. Reisekosten. Ein Beispiel für eine VO ist das vom Alfred-Wegener-Institut für Polar- und Meeresforschung koordinierte Projekt *„Plate Tectonics and Polar Gateways in Earth History“*. Hierbei handelt es sich um eine VO, in der ca. 50 Institutionen mit über 80 Personen aus 14 Ländern zusammenarbeiten [MAK06].

Das Konzept der Virtuellen Organisation ist nach Schiffers [SCH07] von zentraler Bedeutung für Grids und bildet ein fundamentales Konzept, um organisationsübergreifend komplexe IT-Lösungen bereitzustellen. Deswegen beschreiben Foster et al [FOS04] das Grid-Problem allgemein als:

„coordinated resource sharing and problem solving in dynamic, multi-institutional virtual organizations.“
(Ian Foster)

Virtuelle Organisationen ermöglichen *legal entities* (reale Organisationen) und/oder einzelnen Personen das Zusammenarbeiten und die Verwendung von Ressourcen und/oder Diensten.

Durch den Einsatz von Virtuellen Organisationen in Grids entstehen für das Grid-Management nicht nur Anforderungen, Grid-Ressourcen und -Dienste VO-geeignet bereitzustellen, sondern auch Anforderungen an das Management der VOs selbst. Daraus hat sich der Begriff *VO-Management* gebildet, der mit folgenden Worten von der deutschen Grid Initiative „D-Grid“ beschrieben wurde [5]:

„Um umfangreiche e-Science Aufgaben zu koordinieren, bedarf es nicht nur einer geeigneten IT-Plattform, sondern auch einer übergeordneten Instanz, die die Abwicklung und Steuerung der essentiellen Geschäftsfälle für die Kunden des Grid bewerkstelligt. Hierzu muss den Kunden eine virtuelle Plattform im Netz bereitgestellt werden, über die diese Geschäftsfälle initiiert und elektronisch abgewickelt werden.“

2.2.1 Charakterisierung von Virtuellen Organisationen

Zunächst beschreibt dieser Unterabschnitt verschiedene Eigenschaften und Darstellungskonzepte von VOs. Anschließend werden die Unterschiede zu realen Organisationen erläutert.

Eigenschaften von Virtuellen Organisationen:

Die folgende Liste beschreibt VO-Eigenschaften, die vor allem für D-Grid von Bedeutung sind [WP2-07]:

- eine VO besteht aus mindestens einem Mitglied
- eine VO kann aus einer kompletten realen Organisation bestehen
- eine VO kann Sub-VO einer anderen VO sein, die dann selbst als Super-VO bezeichnet wird
- eine VO kann komplett aus anderen VOs bestehen (Hierarchien von VOs)
- eine VO „besitzt“ keine realen Ressourcen, allenfalls virtuelle Ressourcen
- eine VO verwaltet Ressourcen realer Organisationen
- eine VO ist *keine* gesetzlich definierte Organisation (legal entity)

Systemtheoretische Sicht:

Der Organisationsbegriff bei Virtuellen Organisationen ist aus der Sichtweise der Informatik systemtheoretisch motiviert [SCH07]. Demnach kann man Organisationen als „Systeme“ interpretieren. Deswegen können folgende Begriffe für die Beschreibung von VOs verwendet werden:

- *Boundaries:* Abgrenzung von System und Umwelt.
- *VO-Entitäten* entsprechen den Systemelementen.
- *VO-Prozesse* beschreiben das Zusammenwirken der Entitäten.

Alle Elemente, die außerhalb des Systems bzw. der VO liegen, sind Bestandteile der Umwelt.

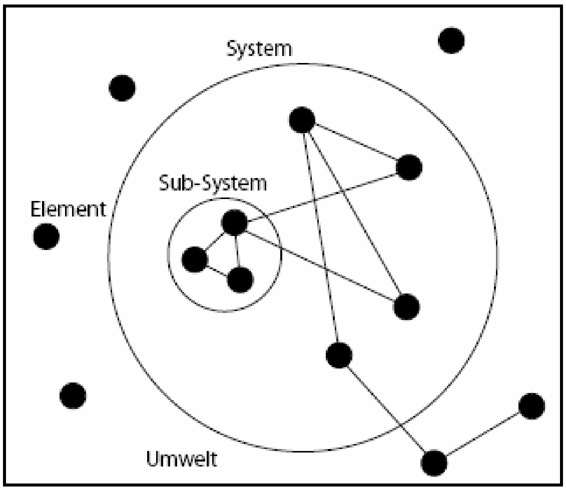

Abb.3: Darstellung eines Systems [SCH07]

Offene und geschlossene VOs:

Virtuelle Organisationen lassen sich in zwei Organisationsformen unterteilen:

Virtuelle Organisationen sind ***offen***, wenn mindestens eine VO-Entität in Wechselwirkung zu einer anderen Organisation (Umwelt) steht.

Virtuelle Organisationen sind ***geschlossen***, wenn keine Wechselwirkung existiert.

VO-Gestaltungskonzepte:

Allgemein kann man bei einer VO zwei verschiedene Gestaltungskonzepte unterscheiden:

- **Intraorganisatorisches Gestaltungskonzept:**
  Ziel dieses Konzepts ist es, die räumlichen und zeitlichen Begrenzungen zu überwinden und die Vorteile des verteilten Operierens zu nutzen. Der Fokus liegt auf der Kollaboration der Mitarbeiter.

- **Interorganisatorisches Gestaltungskonzept:**
  Das Konzept stellt die VO als ein kooperatives, flexibles Netzwerk rechtlich selbständiger Organisationen dar, die Teile ihrer Ressourcen gemeinsam nutzen [SCH07].

Betrachtungsebenen anhand von Komplexitätsstufen:

Eine VO kann wie in Abbildung 4 zu erkennen ist, in drei Betrachtungsebenen beschrieben werden:

1. **Mikroebene:**
   Die Mikroebene entspricht der intraorganisationalen Kooperation.

2. **Mesoebene:**
   Die Mesoebene beruht auf dem interorganisatorischen Konzept und beschreibt die Kooperation zwischen Organisationen.

3. **Makroebene:**
   Bei der Makroebene liegt der Schwerpunkt auf der Kooperation, die zwischen VOs stattfindet.

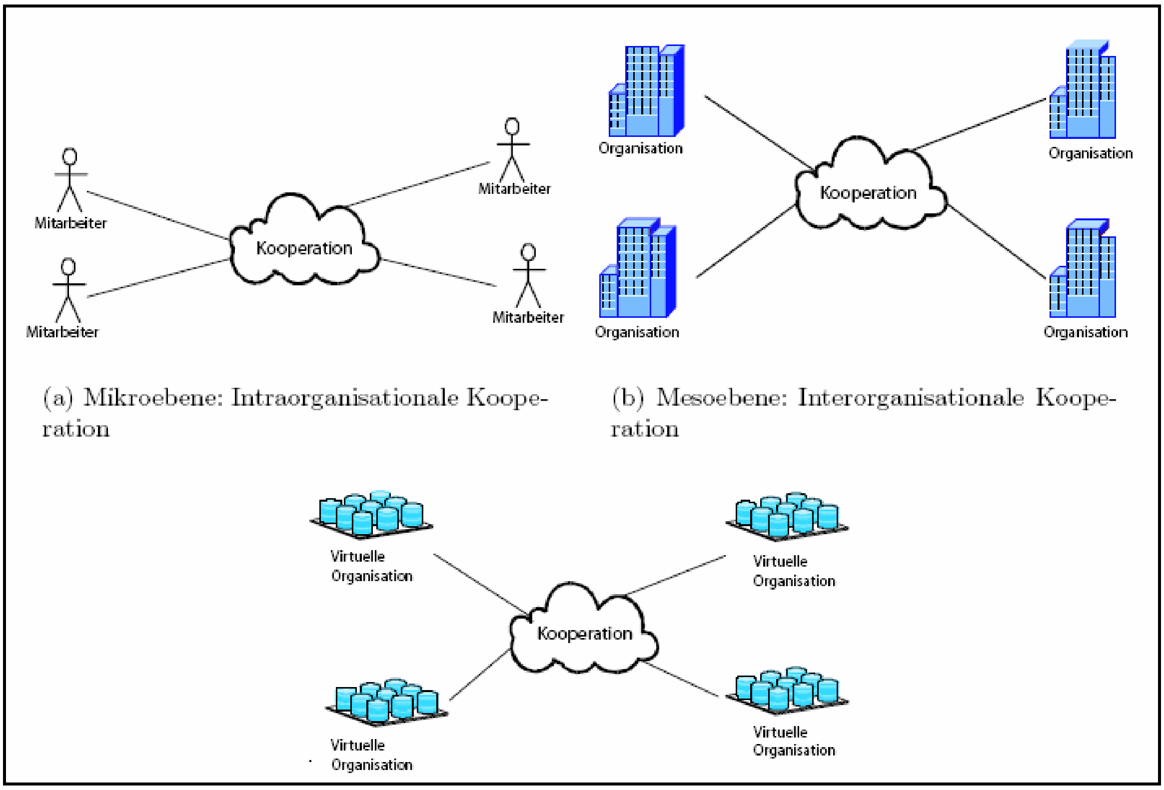

Abb.4: Betrachtungsebenen von Virtuellen Organisationen [SCH07]

Unterschiede zu realen Organisationen:

Die folgenden Punkte veranschaulichen die wesentlichen Unterschiede zu realen Organisationen [COJ07]:

- **Physikalischer Standort:**
  Anders als bei realen Organisationen ist der physikalische Standort irrelevant. Die eingesetzten Ressourcen sowie Mitglieder können unterschiedlichen Standorten sowie unterschiedlichen Organisationen angehören.

- **Keine direkte Kommunikation:**
  Die Kommunikation verläuft durch das Netzwerk. Deswegen gibt es keine direkte Kommunikation zwischen den Mitgliedern einer VO.

- **Zielsetzung:**
  Mitglieder einer VO verfolgen *ein* gemeinsames Ziel. Dies ist der Hauptgrund für die Gründung einer VO.

- **Gründung:**
  Die Gründung ist verglichen mit realen Organisationen viel *flexibler*. Aufgaben, wie Standortssuche und Personaleinstellung entfallen.

- **Dynamik:**
  Virtuelle Organisationen haben eine wesentlich dynamischere Struktur verglichen mit realen Organisationen. VOs befinden sich in ständiger Umstrukturierung. Bei Bedarf können Mitglieder und Ressourcen zu jeder Zeit die VO verlassen bzw. sich anderen VOs anschließen.

- **Zeitlich befristet:**
  VOs sind zeitlich befristet, da sie nach Erreichen des Ziels aufgelöst werden.

- **Kosten:**
  Zwar verursacht die VO aufgrund des zusätzlichen administrativen Aufwandes Verwaltungskosten, doch Transport-, Organisations- sowie Raumkosten entfallen völlig.

- **Autorisierung:**
  Die Autorisierung ist ein wichtiger Bestandteil von VOs und muss flexibel und skalierbar gestaltet sein. Im Gegensatz zu realen Organisationen können sich die Rechte von VO-Mitgliedern kurzfristig ändern.

2.2.2 VO-Struktur

Eine VO kann eine komplexe, hierarchische Struktur haben, die Gruppen und Unter-Gruppen besitzt. Diese Struktur kann mit einem *DAG*[4]-Graph dargestellt werden:

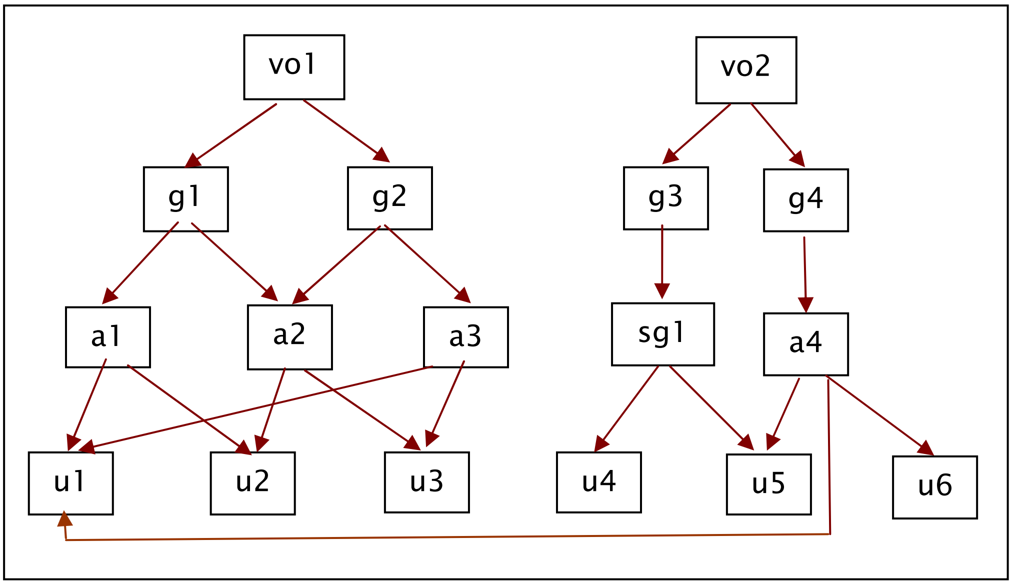

Abb.5: VO-Struktur

Wie auf der obigen Abbildung zu erkennen ist, hat der Benutzer `u1` in der VO `vo1` unterschiedliche Attribute (`a1`,`a3`) und ist gleichzeitig Mitglied einer anderen VO mit dem Attribut `a4`.

2.2.3 Defizite der heutigen VO-Ansätze

Das „Management von Lebenszyklen Virtueller Organisationen über eine Service-orientierten Grid-Infrastruktur“ ist nach Schiffers für ein effektives und effizientes Grid-Management von fundamentaler Bedeutung und bis heute nur marginal behandelt worden. In heutigen Ansätzen sieht Schiffers noch Defizite, die unter anderem aus den folgenden impliziten oder expliziten Annahmen resultieren:

Annahme langlebiger und geschlossener VOs:

LHC-Grid, DEISA-Supercomputing-Grid, NASA Information Power Grid oder GriPhiN-Grid sind Beispiele für Grid-Projekte, die langjährig angelegt sind und ein statisches VO-Modell verfolgen [SCH07]. Diese Annahme entspricht jedoch nicht den bereits beschriebenen Eigenschaften von VOs, die eine kurze Lebensdauer und eine hohe Dynamik aufweisen.

[4] DAG ist die Abkürzung für Directed Acyclic Graph

Annahme manueller Administration

Zwar beschreibt Schiffers, dass eine vollständige Automatisierung des VO-Managements visionär ist, jedoch der administrative Aufwand durch weitgehende Automatisierung des Formationsprozesses reduziert werden kann. Diese Automatisierung kann nach Schiffers durch organisationsübergreifende Workflows realisiert werden. Die Annahme, das VO-Management nur manuell zu betreiben, wie es bei den gängigen VO-Technologien der Fall ist, ist für spontane sowie kurzlebige VOs ungeeignet. Folgender administrativer Aufwand wird für das Bilden einer VO benötigt:

- Erstellen einer VO
- Definieren von VO-spezifischen Policies
- Setzen von individuellen Rechten und
- Integrieren der Grid-Entitäten

2.3 Public Key Infrastructure

Eine *Public-Key-Infrastructure* (PKI) ist ein Sicherheitsframework, das für die Verwaltung von Public-Key-Zertifikaten zuständig ist. Public-Key-Zertifikate werden in VOs verwendet, um den Benutzer zu authentifizieren und seine Autorisierungsinformation zu transportieren. Public-Key-Zertifikate basieren auf asymmetrischer Verschlüsselung, die im Verlauf des Unteranschnitts beschrieben wird. Unter Verwaltung wird konkret das Erstellen, Speichern, Verteilen sowie das Zurückrufen von Zertifikaten verstanden.

Eine PKI ist im Allgemeinen in vier Bereiche unterteilt [6]:

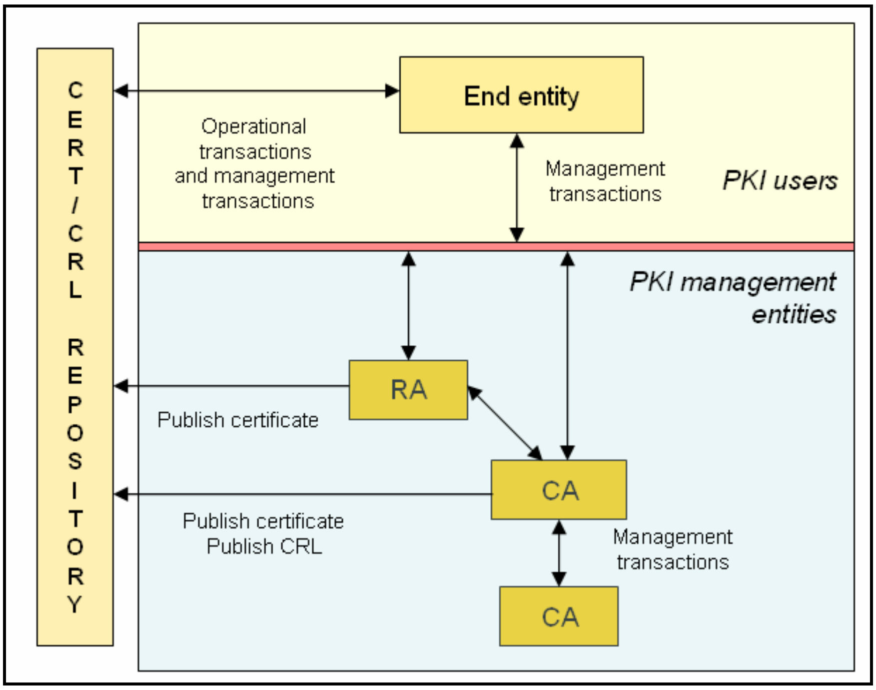

Abb.6: PKI Entities

- *End Entities* sind die Besitzer der Zertifikate. End-Entity-Zertifikate können ausgestellt werden sowohl für Benutzer als auch für Systeme bzw. Systemkomponeneten. Sie beschreiben die Identität der Besitzer. Diese Zertifikate sind nötig, um die Besitzer von den anderen Entitäten in einem digitalen Sicherheitssystem zu unterscheiden.

- *Registration Authorities* (RAs) garantieren die Bindung zwischen den Public Keys und den Identitäten der Zertifikatsbesitzer. So muss sich eine Person erst an eine RA wenden, um ein Zertifikat zu beantragen. Die RA überprüft seine Identität sowie die Angaben in seinem Antrag. Falls diese korrekt vorliegen, wird der Antrag an eine *Certificate Authority* weitergereicht.

- *Certificate Authorities* (CAs) sind Zertifizierungsinstanzen, die Public-Key-Zertifikate ausstellen, verteilen, archivieren und widerrufen. Eine CA signiert die Schlüsselpaare, nachdem der Anforderer von einer vertrauenswürdigen RA überprüft wurde. Für eine CA können mehrere RAs zum Einsatz kommen. CAs können im Allgemeinen in Sub-CAs eingeteilt werden. So kann zum Beispiel eine CA speziell für die Verwaltung von Server-Zertifikaten zuständig sein, eine andere für Benutzerzertifikate.

- *Repository* ist ein Verzeichnisdienst, in dem alle gültigen sowie gesperrten Zertifikate gespeichert werden. Alle gesperrten Zertifikate werden von der CA in einer so genannten Zertifikatssperrliste (Certificate Revocation List CRL) eingetragen.

Im Folgenden werden die Prinzipien der asymmetrischen Verschlüsselung, des Hashings sowie der digitalen Signatur beschrieben:

Asymmetrische Verschlüsselung:
Das erste public-key-basierte Sicherheitsverfahren geht auf Whitfield Diffie und Martin Hellman zurück, die im Jahr 1976 ein Paper mit dem Titel „New Directions in Cryptography“ (*IEEE Transactions on Information Theory)* veröffentlichten [VAC05]. Dieses Verfahren revolutionierte die Welt der kryptographischen Forschung. Sie stellten für die damalige Zeit ein Paradoxon auf: die Möglichkeit, Nachrichten *sicher* im *unsicheren* Netz zu verschicken. Mit anderen Worten wird mit diesem Mechanismus kein separater Sicherheitskanal benötigt, um den geheimen Schlüssel vom Sender A zum Empfänger B zu übermitteln, was bei symmetrischer Verschlüsselung notwendig war.
Hellmann et al. [HEL76] beschrieben diese Voraussetzung der symmetrischen Verschlüsselung gerade im Hinblick auf E-Business als Problem und fügten folgende Wörter hinzu:

“The cost and delay imposed by this key distribution problem is a major barrier to the transfer of business communications to large teleprocessing networks.”
(Martin Hellmann, Whitfield Diffie)

Der Mechanismus von Hellmann und Diffie basiert auf der asymmetrischen Verschlüsselung.

Dieser Mechanismus wird „Public Key Cryptography“ genannt. Folgende Bedingungen sollen im Allgemeinen bei diesem Verfahren erfüllt werden:

- Der Empfänger soll ohne Schwierigkeiten, d.h. ohne große Vorbereitung, alle Nachrichten entschlüsseln oder verifizieren können.
- Der Empfänger soll aber auch nur die Nachrichten, die für ihn bestimmt sind, entschlüsseln können.
- Die Nachricht darf von keiner dritten Partei gelesen oder verändert werden können.

Im Gegensatz zur symmetrischen Verschlüsselung werden bei dieser Verschlüsselungsform für die Ver- und Entschlüsselung zwei korrespondierende Schlüssel verwendet. Jeder Kommunikationspartner besitzt einen privaten und einen öffentlichen Schlüssel, wobei der private Schlüssel nicht mit realistischem Aufwand aus dem öffentlichen errechnet werden kann. Wie man es dem Namen entnehmen kann, muss der private Schlüssel streng geheim gehalten werden, denn wenn der private Schlüssel nicht nur dem Besitzer bekannt ist, ist das Schlüsselpaar kompromittiert. Der öffentliche Schlüssel dagegen kann bekannt gemacht werden.
Die untere Abbildung veranschaulicht dieses Prinzip der Ver- und Entschlüsselung [KAR06]:

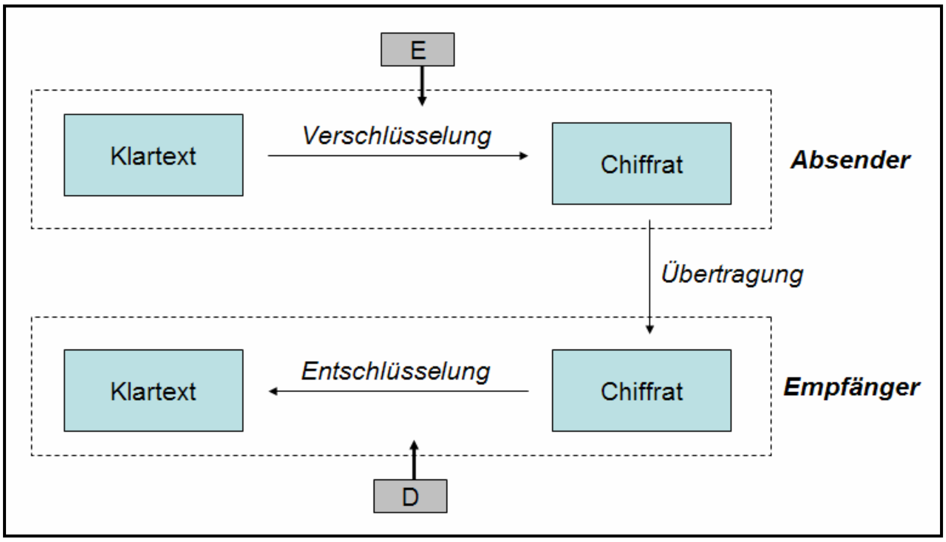

Abb.7: asymmetrische Verschlüsselung

Folgende Eigenschaften gelten bei dem Schlüsselpaar [LEH02]:

- Zu einem öffentlichen Schlüssel gehört immer auch ein privater Schlüssel. Diese sind mathematisch voneinander abhängig.
- Was mit dem öffentlichen Schlüssel verschlüsselt wurde, lässt sich nur mit dem korrespondierenden privaten Schlüsseln entschlüsseln. Dies gilt auch umgekehrt.

Hashing:
Hashing ist eine Methode, die einen „digitalen Fingerabdruck“ (Hash) für eine Nachricht erzeugt. Aus einer beliebig langen Nachricht wird ein Hash mit einer fixen Größe gewonnen (normalerweise 128 oder 180 Bits). Die Besonderheit des Hash-Codes ist, dass er immer eindeutig ist. (Eine andere Nachricht würde dementsprechend ein anderes Hash produzieren).
Der Hash-Algorithmus wird auch „Einweg-Hashfunktion“ (one-way hash function) genannt, da mit der Funktion nur der Hashwert aus der Nachricht berechnet werden kann und eine Rekonstruktion der Nachricht aus Sicherheitsgründen nicht möglich ist. Für diesen Algorithmus wurden aber auch andere Namen benutzt, wie *Message Digest*, *kryptographische Prüfsumme*, *Message Integrity Check* (MIC) sowie *Manipulation Detection Code* (MDC).

Mit Hilfe des Hash-Codes kann *Integrität* gewährleistet werden, falls der Angreifer nicht die Möglichkeit hat, den Hashwert während des Transports zu manipulieren. *Authentizität* kann gewährleistet werden, wenn in die Berechnung des Hashwertes ein „Geheimnis“ einfließt. Umso höher die Länge des Hashwertes ist (normalerweise wird heute ein Hash von mind. 160 Bit gefordert), desto höher ist die Wahrscheinlichkeit, dass folgende Sicherheitsbedingungen erfüllt werden:

- Der Hashwert muss sich schnell berechnen lassen.
- Es muss sich um eine Einwegfunktion handeln, d.h. es muss praktisch unmöglich sein, zu einem gegebenen Hashwert die ursprünglichen Daten zu berechnen.
- Die Funktion muss kollisionsresistent sein, d.h. es muss praktisch unmöglich sein, zwei Daten zu finden, die denselben Hashwert haben.

- Es muss sich um eine Kompressionsfunktion handeln, d.h. dass es müssen beliebig lange Daten auf einen Hashwert fester Länge abgebildet werden.

Digitale Signatur:

Die *digitale Signatur* kann man als ein Analogon zur menschlichen Unterschrift ansehen. Sie ist eine Kombination aus der beschriebenen asymmetrischen Verschlüsslung und Hashing. Die menschliche Unterschrift wird meistens verwendet, um mehr oder weniger eindeutig überprüfen zu können, ob eine Nachricht von einer bestimmten Person geschrieben worden ist. Eine digitale Signatur dagegen erhebt den Anspruch wesentlich fälschungssicherer zu sein.

Bei der digitalen Signatur wird der private Schlüssel nicht auf die Nachricht, sondern auf den Hashwert angewendet. Das bedeutet, dass die Nachricht selbst unverschlüsselt bleibt. Die digitale Signatur dient als Nachweis der Authentizität und Integrität.

Der Sender A berechnet also zunächst mittels einer Hash-Funktion den Hashwert einer Nachricht und verschlüsselt diesen mit seinem privaten Schlüssel. Dadurch entsteht die digitale Signatur. Die Nachricht und die verschlüsselte Prüfsumme (Hash) verschickt er an Empfänger B. Der Empfänger B, der den öffentlichen Schlüssel von A besitzt, entschlüsselt den übermittelten Wert und berechnet zur Kontrolle den Hash der Nachricht, die er von A erhalten hat. Wenn beide Werte übereinstimmen, kann sich Empfänger B sicher sein, dass der richtige Text übermittelt worden ist.

2.3.1 X.509-Zertifikate

Bei der obigen Beschreibung von digitalen Signaturen wurde vorausgesetzt, dass die öffentlichen Schlüssel auf einem vertrauenswürdigen Weg ausgetauscht wurden, so dass z.B. der Sender A sicher sein kann, dass der ihm gelieferte öffentliche Schlüssel auch dem Empfänger B gehört. Wenn dies nicht der Fall ist, können Sender und Empfänger ihre öffentlichen Schlüssel durch eine Person C signieren lassen, dem sie vertrauen oder die Signatur findet durch eine vertrauenswürdige Instanz statt. Der zweite Fall entspricht dem Verfahren des Zertifikats.

Ein Zertifikat ist laut Signatur-Gesetz:

„eine mit einer digitalen Signatur versehene digitale Bescheinigung über die Zuordnung eines öffentlichen Signaturschlüssels zu einer natürlichen Person“ [7]

Der öffentliche Schlüssel wird von einer vertrauenswürdigen Instanz (CA) signiert und den jeweils zugehörigen Personen zugeschickt. Damit ermöglichen digitale Zertifikate prinzipiell den Schutz der Vertraulichkeit, Authentizität und Integrität von Daten durch kryptographische Verfahren. Das Zertifikat ist durchaus vergleichbar mit dem Personalausweis. Die vertrauenswürdige Instanz bei Personalausweisen ist das "Meldeamt", das sicherstellt, dass die Unterschrift, die sich auf dem Ausweis befindet, auch tatsächlich zu der Person gehört, deren Stammdaten und Passbild sich auf dem Ausweis befinden.

Ein *X.509-Zertifikat* ist durch *ITU-T (International Telecommunication Union)* ein standardisiertes Zertifikatsformat, das wie folgt aufgebaut wird:

- **Version:** Das Zertifikat-Format wird hier angegeben.
- **Seriennummer:** Die Seriennummer entspricht einem ganzzahligen Wert. Diese wird einmalig von der CA erstellt.
- **Algorithmenidentifikation:** Der eingesetzte Verschlüsselungsalgorithmus wird hier gekennzeichnet.
- **Aussteller:** Das DN-Subjekt (siehe untere Tabelle) des CA-Zertifikats wird hier eingetragen.
- **Gültigkeitsdauer:** Die Gültigkeitsdauer entspricht der Lebensdauer des Zertifikats und beschreibt, wie lange es noch gültig ist.
- **Subject:** Hier wird das DN-Subjekt des Benutzers eingetragen.
- **Public-Key-Informationen:** Der öffentliche Schlüssel und der verwendete Algorithmus werden hier eingetragen
- **Signatur:** Hier steht die digitale Signatur des CA-Zertifikats

Der *Distinguished Name* (DN) hat folgende Struktur:

Bezeichnung	Enthält	Beispiel
[C] Country	ISO-Code des Staats	DE
[ST] State	Land, Provinz	NRW
[O] Organisation	Organisation	Fraunhofer Institut
[OU] Organisation-Unit	Abteilung	SCAI
[CN] Common Name	Identitätsname	Max Mustermann
[L] , City, Location	Stadt	Sankt Augustin
Etc.	...	...

2.3.2 Mehrseitige Authentifizierung

Eine Ein-Weg-Authentifizierung ist bei verteilten Systemen nicht sicher genug, da der klassische Ansatz von Client/Server-Kommunikation hier nicht gegeben ist. In den meisten Fällen müssen die Hosts in der Lage sein, sich gegenseitig zu authentisieren, d.h. dass zwei Kommunikationspartner in der Lage sein müssen, die Identität des anderen zu überprüfen. Wie in Unterabschnitt 2.1.3 beschrieben, ist die allgemeine Grid-Architektur komponentenbasiert. Daher müssen sich die Komponenten gegenseitig authentisieren können.

Die *Zwei-Wege-Authentifizierung* beschreibt, wie zwei verschiedene Parteien sich mittels Zertifikaten gegenseitig authentisieren können. Voraussetzung hierfür ist, dass beide Parteien ein Zertifikat besitzen und den jeweiligen Zertifizierungsstellen vertrauen. Folgende Abbildung verdeutlicht dieses Prinzip:

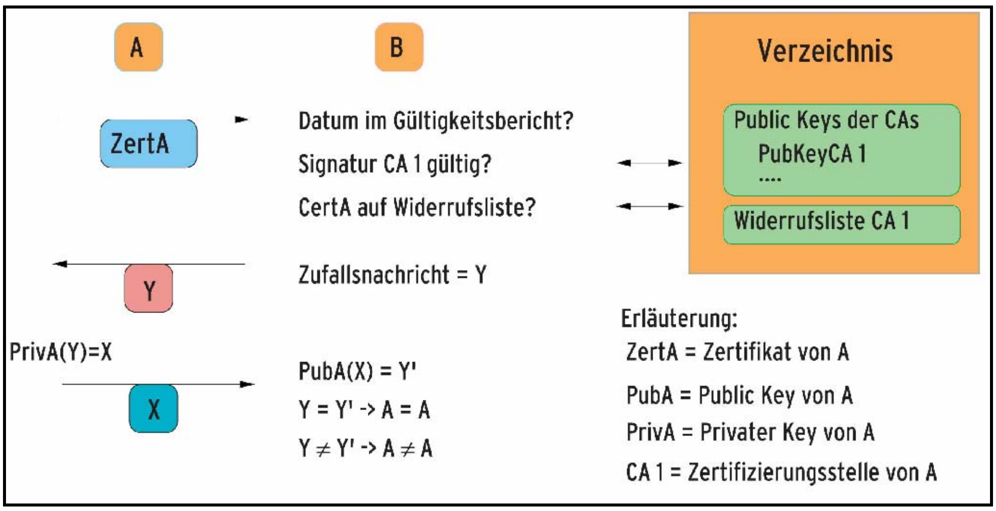

Abb.8: Zwei-Wege-Authentifizierung [8]

Zunächst sendet Partei A ihr eigenes Zertifikat *„CertA“* an Partei B. Partei B überprüft *„CertA“* nach Gültigkeit und Richtigkeit der Signatur mithilfe des öffentlichen Schlüssels der Zertifizierungsstelle, der Partei A vertraut. Die öffentlichen Schlüssel der Zertifizierungsstellen haben beide Parteien lokal in einem bestimmten Verzeichnis gespeichert. Beim Globus Toolkit (siehe Unterabschnitte 2.6.2) werden die Public Keys der CAs standardmäßig im Verzeichnis `/etc/grid-security/certificates` abgelegt. Wenn das Zertifikat nicht in der Widerrufsliste (Certificate Revocation List) eingetragen ist, sendet Partei B eine Nachricht zufälligen Inhalts an Partei A. Partei A verschlüsselt die Nachricht mit ihrem privaten Schlüssel und sendet sie zurück an Partei B. Nachdem Partei B diese Nachricht mit dem öffentlichen Schlüssel von A entschlüsselt hat und das Ergebnis der Nachricht mit der zuvor gesendeten Nachricht übereinstimmt, ist gewährleistet, dass Partei A im Besitz des privaten Schlüssels ist. Nun authentisiert sich Partei B bei Partei A in derselben Form.

2.3.3 Proxy-Zertifikate

Da das X.509-Zertifikat die Möglichkeit bietet, von ihm abgeleitete, neue Schlüsselpaare zu generieren, ist ein Proxy-Mechanismus möglich. Normalerweise wird der geheime Schlüssel eines Benutzers zusätzlich durch ein Passwort geschützt, das bei Verwendung des Schlüssels stets eingegeben werden muss. Bei Proxy-Zertifikaten ist dies nicht der Fall. Dafür haben Proxy-Zertifikate eine wesentlich kürzere Lebensdauer als gewöhnliche End-Entity-Zertifikate, in der Regel nämlich nur 8-12 Stunden. Der Sinn eines Proxy-Zertifikates ist, dass es stellvertretend für ein End-Entity-Zertifikat genutzt werden kann und hierbei keine weitere Interaktion mit dem Besitzer erforderlich ist, da die Eingabe eines Passwortes entfällt. Der Benutzer generiert vereinfacht ausgedrückt ein Proxy-Zertifikat über folgende Schritte: Zunächst generiert er zwei Schlüssel, wobei der geheime Schlüssel nicht mittels eines Passworts geschützt ist. Anschließend wird der öffentliche Schlüssel mit dem geheimen Schlüssel seines Benutzerzertifikates signiert. Dadurch ist die Zertifikatskette bis zum Wurzelzertifikat überprüfbar. Ein Proxy-Zertifikat erkennt man typischerweise daran, dass sein DN-Subjekt mit CN=proxy erweitert wird. Da das Proxy-Zertifikat auf X.509-Zertifikaten basiert, können diese für die gegenseitige Authentifizierung von Hosts sowie die System-Komponenten verwendet werden.

2.3.4 Attributzertifikate

Ein *Attributzertifikat* ist ein Eigenschaftszertifikat, das durch das Signieren von einer vertrauenswürdigen Zertifizierungsstelle an ein Benutzerzertifikat gebunden wird.

Der Einsatz eines X.509-Attributzertifikates ist durch sein wohldefiniertes, standardisiertes Format sowie durch seine einfache Erweiterungsmöglichkeit gerade in Umgebungen, die ihre Authentifizierung über X.509-Zertifikate erreichen [CIA04], eine sinnvolle Lösung. Durch ihren Einsatz kann eine höhere Flexibität als bei gängigen PKI-Lösungen erreicht werden.
Ein Vorteil für die Bildung von Attributzertifikaten ist, dass sie von jeder vertrauenswürdigen Zertifizierungsstelle erzeugt werden können. Es ist nicht zwingend erforderlich, dass sie von der Zertifizierungsstelle des Benutzerzertifikats signiert wird.

Typische Inhalte eines Attributzertifikates sind:

- Benutzerrollen, mit der sich der Benutzer am System autorisieren kann
- Kennzeichen für Nutzungseinschränkungen
- Kennzeichen für berufliche und fachliche Qualifikationen und den damit verbundenen Berechtigungen

2.3.5 Short-lived-Credentials

Die Grid-Infrastruktur basiert auf der Verwendung von Zertifikaten. Nun gibt es viele Identitätsmanagement-Systeme, die keine Authentifizierung und Autorisierung (siehe Unterabschnitt 2.4) durch Zertifikate unterstützen. Um eine lokale Identität auf eine gridfähige Identität abzubilden, wurden kurzlebige Zertifikate entwickelt (engl. ***S**hort **L**ived **C**redential*).
Die Unterschiede zwischen langlebigen Zertifikaten und SLCs liegen in der Lebensdauer und im Erhalt der Zertifikate. SLCs haben eine Lebensdauer von maximal einer Million Sekunden[5], während langlebige Zertifikate eine Lebensdauer von über einem Jahr besitzen können.
Ein Benutzer erhält ein SLC nach einer erfolgreichen Authentifizierung an seinem Identitätsmanagement-System. Langlebige Zertifikate erhält man durch die Verwendung von RAs, die die Identität des Benutzers z.B. mittels des Personalausweises an das Zertifikat binden. Eine wichtige Bedingung der kurzlebigen Zertifikate ist, dass die DN eindeutig sein müssen, damit sie einer einzigen End-Entity der Organisation zugeordnet werden können.

[5] Eine Million Sekunden entsprechen ca. 11 Tage.

2.4 Authentifizierungs- und Autorisierungs-Infrastrukturen

Eine *Authentifizierungs- und Autorisierungs- Infrastruktur* (AAI) ist die Grundlage für Verfahren, welches Mitgliedern unterschiedlicher Institutionen/Organisationen einen kontrollierten Zugriff auf geschützte Ressourcen ermöglicht, die verteilt auf unterschiedlichen Servern liegen können [9].

Das ***D**eutsche **F**orschungs-**N**etz* (DFN) betreibt eine AAI, die sowohl von Wissenschaftseinrichtungen als auch von (kommerziellen) Anbietern genutzt werden kann. Durch die *DFN-AAI* wird ein notwendiges Vertrauensverhältnis zwischen Anwendern und Anbietern sowie ein organisatorischer Rahmen für den Austausch von Nutzerinformationen geschaffen.

Der Abschnitt beschreibt zunächst die Motivation einer AAI und geht später auf die verschiedenen Ansätze zur Zugriffskontrolle ein.

Motivation für die Verwendung einer AAI

Die Benutzung des Internets, welches mehr und mehr zu einem sozialen Gebilde avanciert, benötigt einen sicheren Zugang zu geschützten Ressourcen wie Dokumenten, e-Commerce, e-Banking, Communities, Blogs oder Skripten.

Unternehmen, aber auch Institute, Universitäten und Schulen wollen jedoch den *sicheren* Zugriff auf Informationen bzw. Ressourcen auf befugte Benutzergruppen (wie z.B. Mitarbeiter, Studenten und Professoren) beschränken. In diesem Kontext ist mit dem Begriff „sicher“ das Schutzziel Vertraulichkeit gemeint, dass nur Berechtigten einen Zugriff auf Ressourcen erlaubt. Dementsprechend muss der Schutz gegen unberechtigte Kenntnisnahme und Nutzung von Ressourcen gewährleistet werden. Für den sicheren Zugriff zu einer Ressource bedarf es einer erfolgreichen Authentifizierung und Autorisierung des Benutzers.

2.4.1 Authentifizierung

Die Authentifizierung ist der Vorgang der Überprüfung der Identität einer Person. Der Vorgang beinhaltet zwei Schritte: die Identifikation und die Authentisierung. Bei der Identifikation präsentiert der Benutzer seine Identität (z.B. bei der Eingabe seiner „User-ID“). Im nächsten Schritt überprüft das System bei der Authentisierung die Identität des Benutzers. Die Authentifizierung kann über Wissen (z.B. Passwortabfrage), Besitz (z.B. Zertifikate, Tokens) oder über ein persönliches Merkmal (z.B. Fingerabdruck, Retinamuster) des Benutzers erfolgen.

Bei verteilten Systemen ist die Authentifizierungs-Prozedur um einiges komplexer, da ein verteiltes System mehr als eine Komponente auf mehr als einem Host besitzt. Die Komponenten bzw. die Hosts müssen für den Benutzer transparent und sicher miteinander interagieren können.
Im Allgemeinen sind folgende Authentifizierungs-Prozeduren bei verteilten Systemen vorstellbar:

- **Multiple Authentifizierung:** Unter multipler Authentifizierung wird verstanden, dass der Benutzer sich für jedes Teilsystem oder jede Anwendung in Form von beispielsweise Benutzername und Passwort authentisieren muss.

- **Replizierung:** Der Benutzer braucht sich bei der Prozedur „Replizierung" nur einmal zu authentisieren. Seine *Credentials*[6] werden auf die anderen Teilsysteme und Hosts kopiert.

- **Single-Sign-On (SSO):** Ähnlich wie bei der Replizierung braucht der Benutzer sich nur einmal anzumelden. Seine Credentials werden nicht kopiert, sondern ein SSO-Mechanismus führt den Identifikationsnachweis bei den Teilsystemen und Hosts durch.

Die multiple Authentifizierung hat den Vorteil, dass keine Credentials übers Netz verschickt werden müssen. Aber im Hinblick auf Forschungseinsätze ist diese Authentifizierungsform nicht geeignet, da z.B. bei der Auswertung großer Datenmengen die Berechnungen Stunden bzw. Tage dauern können. Der Benutzer wäre gezwungen, sich in dieser Zeit bei jedem benötigten Host anzumelden. Das Merken der vielen Passwörter ist nicht benutzerfreundlich und verursacht administrativen Aufwand, wenn Passwörter vergessen werden. So können diese ein Sicherheitsrisiko darstellen, wenn die Passwörter aufgeschrieben werden und unter Umständen für Fremdpersonen zugänglich sind.
Die Replizierungsform hat den Vorteil, dass bei einer einmaligen Anmeldung *Batch-Jobs*[7] ausgeführt werden können. Der Nachteil ist, dass sensible Daten übers Netz ausgetauscht werden. Trotz der Verwendung von kryptographischen Verfahren bei der Übertragung der Daten besteht immer noch ein großes Sicherheitsrisiko. Wenn ein Angreifer es schafft, diese Daten abzufangen und zu entschlüsseln, sind Angriffsformen wie *Masquerading* oder *Man-in-the-Middle* sowie Bedrohungen wie das

[6] Credentials sind Berechtigungsnachweise, die einem System die Identität eines Benutzers oder eines anderen Systems bestätigen sollen.
[7] Batch-Jobs bezeichnen Programme, die ohne Benutzerinteraktion ausgeführt werden können.

Modifizieren und Hinzufügen von Daten möglich. Gleichzeitig wäre das System nicht mehr glaubwürdig.

Single-Sign-On ist ein System, das es dem Benutzer ermöglicht, sich nur einmalig zu authentifizieren. So lange die Sitzung läuft, ist keine weitere Passworteingabe notwendig. Desweiteren soll Single-Sign-On einen einheitlichen Zugang zu verschiedenen Ressourcen bieten [MIN03]. Der Benutzer kann somit mit einmaliger Authentifizierung eine Vielzahl von Diensten in Anspruch nehmen. Meistens erfolgt die Authentifizierung durch Wissen. Der Benutzer muss sich bei dieser Lösung nur ein Passwort merken. Für die Institution/Organisation bietet ein Single-Sign-On-System den Vorteil, dass der Aufwand der Administration reduziert wird.

2.4.2 Autorisierung

Die Autorisierung erfolgt nach einer erfolgreichen Authentifizierung und sie überprüft das Gewähren von Zugriffsrechten auf Ressourcen. Die Authentifizierung beantwortet die Frage: „Wer bist du?“; die Autorisierung die Frage: “Was darfst du?“.

Deswegen setzt die Autorisierung eine erfolgreiche und korrekte Authentifizierung voraus. Wenn also ein Angreifer sich mit der Identität eines Benutzers am System anmelden konnte, sind die Autorisierungsmechanismen nicht in der Lage, das System vor unbefugtem Zugriff zu schützen. Durch die Autorisierung soll sichergestellt werden, dass nur berechtigte Benutzer auf Systemressourcen zugreifen dürfen. Das Prinzip der Autorisierung kann man sich folgendermaßen vorstellen:

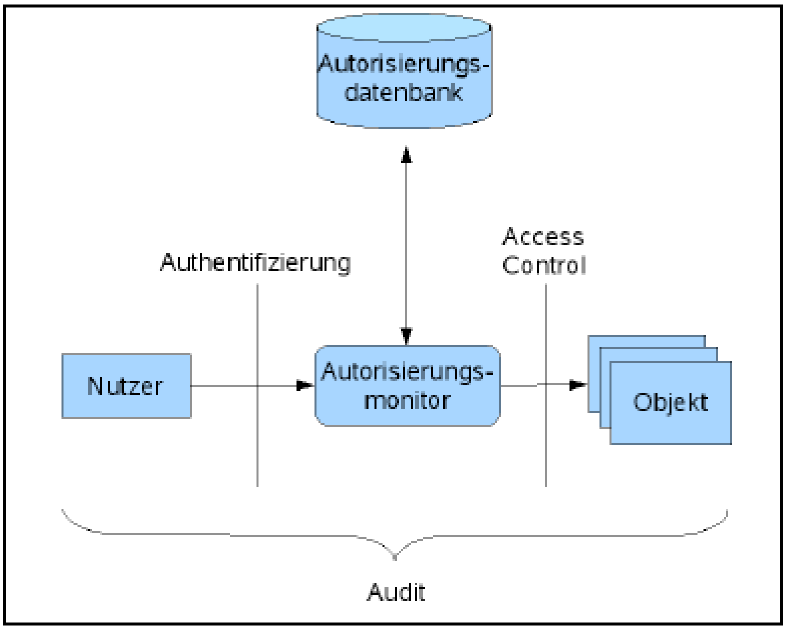

Abb.9: Autorisierung [GRI06]

Damit der Benutzer nicht sofort auf die Objekte des Systems zugreifen kann, werden alle Anfragen an einen *Autorisierungsmonitor* geschickt. Der Autorisierungsmonitor überprüft mit Hilfe der *Autorisierungsdatenbank*, ob der Benutzer die Zugriffsrechte besitzt, auf das Objekt zuzugreifen. Daraufhin wird ihm der Zugriff gewährt bzw. verweigert.

Die *Audit-Komponente* hat die Aufgabe, alle Anfragen zu überwachen und zu protokollieren. Dadurch kann nachträglich analysiert werden, ob die Autorisierungsmechanismen erfolgreich implementiert wurden. Die Audit-Komponente kann auch genutzt werden, um Verstöße gegen die Sicherheitsrichtlinien aufzuspüren. Die Zugriffskontrolle wird im nächsten Unterabschnitt beschrieben.

2.4.3 Zugriffskontrolle

Damit nicht befugte Benutzer auf Daten oder Programme keinen Zugang erhalten, müssen Sicherheitsvorkehrungen getroffen werden, die die Schutzziele Vertraulichkeit, Verfügbarkeit und Integrität gewährleisten. Die Zugriffskontrolle entscheidet darüber, wie auf Ressourcen zugegriffen werden darf und wie diese durch Zugriffsmechanismen geschützt werden.

Die Zugriffskontrolle kann durch ein Grundmodell beschrieben werden. Das *Zugriffsmodell* unterscheidet die Begriffe Objekt, Subjekt und Operationen voneinander. *Objekte* (*o*) kennzeichnen alle Ressourcen, die von Computersystemen verwaltet werden. *Subjekte* (*s*) sind die Benutzer bzw. die vom Benutzer ausgeführten Programme, die durch *Operationen* (*p*) auf die Objekte zugreifen. Bei Subjekten ist zu beachten, dass ein Benutzer durch mehrere Subjekte repräsentiert werden kann, die unterschiedliche Rechte besitzen.

Ein *Zugriffsweg* lässt sich folgendermaßen als Tripel darstellen <s,p,o>. Verbotene Zugriffe werden mit *„deny“* und erlaubte Zugriffe mit *„permit“* gekennzeichnet.

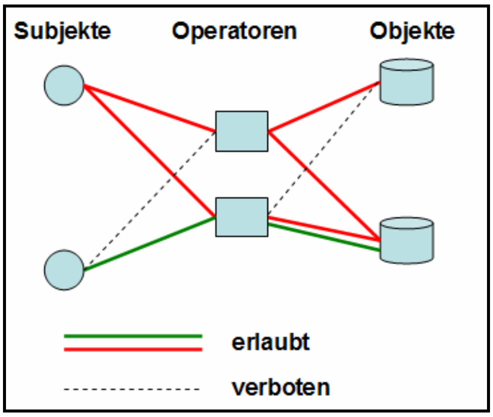

Abb.10: Zugriffswegdarstellung

Identity-based-Access Control

Identity-based-Access Control (IBAC) ist die am meisten verbreitete Zugriffskontrollstrategie.

Sie entscheidet über einen Ressourcen-Zugriff auf Basis der Identität eines Subjektes (Benutzer, Prozess). Die Zugriffsrechte für Ressourcen müssen hierbei für jeden Benutzer/Prozess festgelegt werden.

Role-based-Access Control

Das *Role-based-Access Control* (RBAC) gruppiert die Benutzer eines Systems nach ihren Rollen. Anstatt dem Benutzer direkte Rechte zuzuordnen, werden beim RBAC-Verfahren die Rechte in Rollen zusammengefasst. Das Konzept ist sehr flexibel und gut geeignet für Organisationen mit einer großen Anzahl von Benutzern und Anwendungen.

Einer Rolle können mehrere Subjekte, einem Subjekt mehrere Rollen zugeordnet werden. Zum Beispiel besitzt die Rolle „`Datenbank-Administrator`" die drei Subjekte `a`, `b` und `c`. Subjekt `a` hat weitere Rollen, wie z.B. „`Betriebsleiter`" und „`Globus-Anwender`". Eine Rollenhierarchie ist mit diesem Konzept realisierbar. Aus der technischen Sicht benötigt RBAC ein zentrales Zugriffssystem, worauf die Administratoren die Rollen und die Zuweisung zu den Mitarbeitern verwalten.

Das Konzept ist für die Administration auch ein Vorteil. Der größte Aufwand für Administratoren liegt darin, Einzelrechte den jeweiligen Rollen zuzuordnen. Wenn eine Rolle innerhalb einer Organisation definiert ist, dann ändert sie sich in der Regel nicht mehr bzw. nur geringfügig. Anschließend hat der Administrator die Aufgabe, die Mitgliedschaften in der Menge der spezifizierten Rollen zu verwalten. Diese Verwaltung ist sehr flexibel, da man einem Benutzer Rollen hinzufügen oder entfernen kann.

Attribute-Based Access Control

Attribute-Based Access Control (ABAC) ist vergleichbar mit dem RBAC-Modell. Es unterscheidet sich aber dadurch, dass die Zugriffsrechte nicht anhand von Rollen sondern von Attributen geregelt werden. Im Gegensatz zu den statischen Rollen können Attribute sehr dynamisch sein. Neben den allgemeinen statischen Eigenschaften (wie beispielsweise dem Namen und der Position einer Person) können Attribute auch spezielle dynamische Eigenschaften wie z.B. das Alter oder eine Adresse besitzen.

Aufgrund der Tatsache, dass man eine Rolle als ein Attribut bezeichnen kann, ist das RBAC-Modell als eine Teilmenge von ABAC zu verstehen.

2.4.4 Mögliche Probleme bei der Verwendung von AAI

Bei der Verwendung von einer Authentifizierungs- und Autorisierungsinfrastruktur können folgende Probleme für den Ressourcen-Anbieter und den Anwender entstehen:

- hoher administrativer Aufwand für die Ressourcen-Anbieter bei einer großen Menge an Benutzern.
- Der Benutzer benötigt meist viele Passwörter und ist dadurch überfordert.
- Die Berechtigung ist eventuell nicht standortunabhängig.
- Mitarbeiter, die das Unternehmen bereits verlassen haben, verfügen weiterhin über Zugriff auf Unternehmensapplikationen.
- Personenbezogene Nutzerdaten werden beim Zugang jedem einzelnen Dienstanbieter preisgegeben.
- Unternehmensübergreifende Zusammenarbeit ist schwer realisierbar.
- Kurzfristiges aufbauen von Geschäftsbeziehungen ist kaum möglich.

2.5 Federated Identity Management

Federated Identity Management (FIM) ist eine Erweiterung des Identity Management. Als *Identity Management* wird das Erzeugen, Ändern, Registrieren, das Verteilen, Bereitstellen, Integrieren, Transformieren, die Verwendung, das Terminieren und Archivieren von digitalen Identitäten verstanden. Eine *digitale Identität* wird im Allgemeinen als eine Abbildung eines Individuums verstanden, in der Informationen wie Identifikation (eindeutige Identifikation wie z.B. ID, Name), Beschreibung (rollenunabhängige Attribute wie z.B. Adressinformationen) und Kontext (rollenabhängige Attribute wie z.B. Student, Administrator) eines Individuums gespeichert werden. Ein Identity Management System besitzt einen lokalen Charakter und kann dementsprechend keine Authentifizierung von digitalen Identitäten über Unternehmensgrenzen hinweg ermöglichen. Aus diesem Grund wurde ein neuer Ansatz entwickelt, der eine solche Verwaltung ermöglicht. Dieser wird als Federated Identity Management bezeichnet.

FIM soll gewährleisten, dass Partnerinstitutionen, die als Dienstanbieter agieren, über speziell ausgelegte Protokolle Zugriff auf die notwendigen Autorisierungsinformationen der digitalen Identitäten (sei es in der eigenen Institution oder in den Partnerinstitutionen) erhalten [VED06].

Die wichtigsten Parteien in einem Federated Identity Management sind [VED06]:

- Der *Identity Principal* ist der referenzierte Benutzer.
- Der *Identity Provider* (IdP) ist die Stelle, die Identitäten speichert, authentifiziert und auf Anforderung Identitätsinformationen in Form von einer Zusicherung („*Assertion*“) herausgibt. Ein Identity Principal muss genau einem Identity Provider zugeordnet sein.
- Als Gegenpol zum Identity Provider ist der *Service Provider* der Abnehmer der Zusicherung. Er vertraut auf die Gültigkeit der Zusicherung und bietet geschützte Ressourcen an.

Die Kommunikation mit den Parteien sieht grob wie folgt aus:
Die Service Provider bieten geschützte Dienste an, auf die ein Benutzer Zugriff erhalten möchte. Der Service Provider verweist den Benutzer (Identity Principal) zu seiner Heimateinrichtung (Identity Provider), wo sich der Benutzer zu authentifizieren hat. Nach einer erfolgreichen Authentifizierung stellt der Identity Provider dem Service

Provider Autorisierungsinformationen über den Benutzer zur Verfügung, damit der Service Provider anhand dieser entscheiden kann, ob der Benutzer Zugang auf die gewünschte Ressource erhält oder nicht.

FIM-Ansätze

FIM-Ansätze werden als *zentralisiert* bezeichnet, wenn es nur einen Identity Provider für alle Benutzer gibt, bzw. als *dezentralisiert*, wenn es mehrere IdPs geben kann.

Föderativer Charakter des FIM

Der *föderative Charakter* dieses Management-Systems ist dadurch gegeben, dass sich mehrere Organisationen/Institute auf der Basis gemeinsamer Richtlinien zusammenschließen. Eine Föderation schafft das notwendige Vertrauensverhältnis (*Trust*) zwischen Einrichtungen (Identity Providern) und Anbietern (Service Providern) und einen organisatorischen Rahmen für den Austausch von Benutzerinformationen [GRI06].

Abbildung 11 stellt eine Föderation grafisch dar:

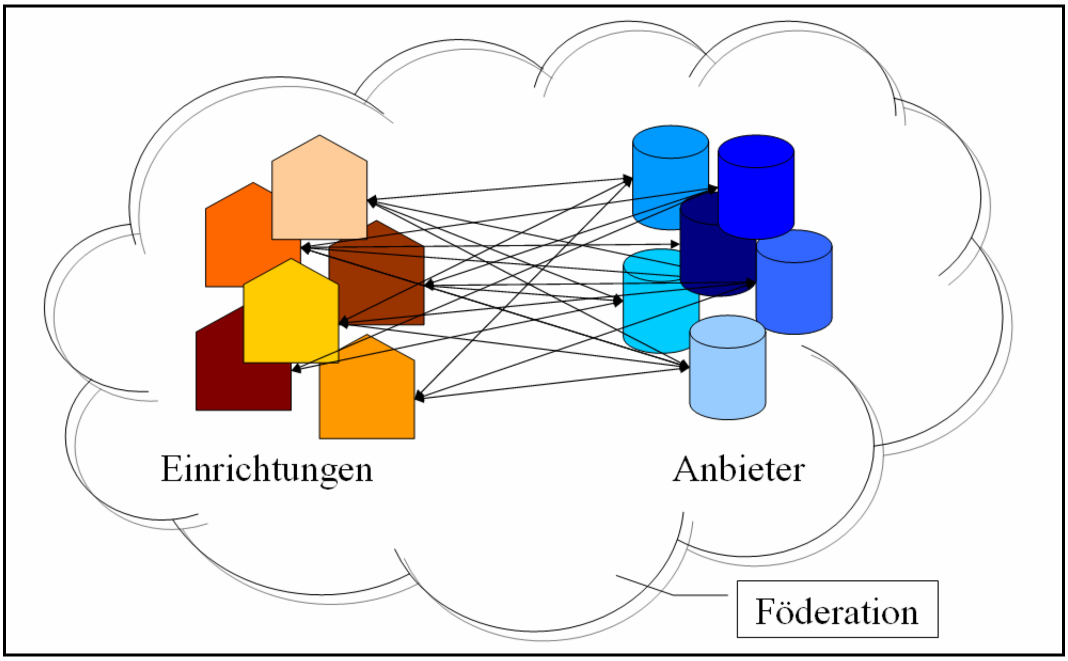

Abb.11: Föderation von Einrichtungen und Anbietern [GRI06]

Durch die Föderation wird ein so genannter *„Circle of Trust“* gebildet, welcher aus einem wirtschaftlichen Verbund mit einer Anzahl an Service- und Identity Providern besteht.

Die Kommunikation zwischen dem Identity Provider und dem Service Provider (SP) muss einen vertrauenswürdigen Austausch von Authentifizierungs- und Autorisierungsinformationen über eine digitale Identität gewährleisten. Das Protokoll *SAML* wurde vom *OASIS*[8]*-Konsortium* entwickelt, um diese Anforderung zu erfüllen.

2.5.1 SAML

SAML (Security Assertion Markup Language) ist eine XML-basierte Beschreibungssprache für Sicherheitsbestätigungen. Durch SAML können Informationen zur Authentifizierung und Autorisierung ausgetauscht werden. SAML ist ein OASIS-Standard, auf dessen Basis eine Identity Federation und Web-Single-Sign-On eingesetzt werden kann.

Der Identity Provider tritt in der Rolle der *Asserting Party (SAML Authority)* auf, welche eine Identität bescheinigt, und der Service Provider tritt in der Rolle der *Relying Party* auf, welche authentifizierte Identitäten empfängt. Die Kommunikation zwischen der Asserting Party und der Relying Party erfolgt durch vertrauensvolle Aussagen (*SAML Assertion*). Die Informationsstruktur der Assertions wird in einer Föderation vertraglich vereinbart.

Die SAML-Komponenten werden wie folgt definiert:

- **SAML- Protokoll:**
  Das *SAML-Protokoll* definiert die Interaktion zwischen den sich gegenseitig als vertrauenswürdig anerkennenden *SAML-Requestern* und den *SAML-Respondern*. Die Anfragen des SAML-Requesters werden von einem SAML-Responder erwidert. Die jeweiligen Nachrichten werden vom *RequestAbsractType* oder dem *StatusResponseType* abgeleitet und können Informationen über die Authentifizierung, die Attribute und den Zugriff einer Identität beinhalten.

- **SAML-Bindings:**
  Die *SAML-Bindings* definieren die Übertragung von SAML-Nachrichten über Standard-Protokolle. Der *SAML-2.0*-Standard unterstützt für die Transport-schicht die Protokolle HTTP und SOAP. Die Bindings sind unter anderem SAML SOAP, HTTP Redirect, HTTP POST, HTTP Artifact und SAML URI.

[8] OASIS (Organization for the Advancement of Structured Information Standards) [10]

- **SAML-Profile**:
  Die SAML-Profile spezifizieren wie SAML Assertions in ein Message-Framework oder in ein Protokoll eingebunden und wieder extrahiert werden. SAML definiert ein Web-Browser-Profil und ein WS-Security-Profil (als Teil der WS-Security-Spezifikation) [GRI06].

SAML Assertion

Die wesentlichen Bestandteile von SAML sind die Assertions. Sie können digital signiert werden und sind daher vertrauenswürdige Aussagen über eine digitale Identität.

SAML unterscheidet drei verschiedene Arten von Assertions [GRI06]:

- **Authentication Assertions** bestätigen, dass bestimmte Benutzer auf geschützte Ressourcen zugreifen dürfen.

- **Attribute Assertions** bestätigen, dass einem Benutzer oder einem Web Service bestimmte statische (Rollen, Funktionen) oder dynamische Attribute (z.B. Kontostand-Informationen) zugeordnet sind. Attributinformationen spielen bei der Zuweisung von Zugangsberechtigungen eine wichtige Rolle.

- **Authorisation Decision Assertions** stellen fest, ob und wie auf eine spezifische Ressource zugegriffen werden darf.

```
<Assertion AssertionID="...." IssueInstant="..." Issuer="Identity
Provider">
  <Conditions NotBefore="2007-10-05T08:30:40.351Z" NotOnOrAfter="2007-10-
   05T16:30:40.351Z">
       <AudienceRestrictionCondition>
          <Audience>Service Provider</Audience>
        </AudienceRestrictionCondition>
   </Conditions>
   <AttributeStatement>
      <Subject>
         Digitale Identität
       </Subject>
    <Attribute xmlns:typens="urn:mace:shibboleth:1.0"
          AttributeName="AttributName">
         <AttributeValue> Wert des Attributes </AttributeValue>
      </Attribute>
    </AttributeStatement>
</Assertion>
```

Listing 1: SAML AttributeStatement-Assertion

In Listing 1 ist allgemein eine Assertion mit Attributinformationen einer Identität dargestellt. Eine Assertion ist eindeutig und wird daher mit einer Identifikationsnummer (*AssertionID*) beschrieben. Der Identity Provider ist in diesem Beispiel der Erzeuger (*Issuer*), der zum Zeitpunkt `z1` die Assertion erzeugt hat (*IssuerInstant*).

Der digitale Benutzer wird nach einer erfolgreichen Authentifizierung mit dem Element *Subject* definiert. Darüber hinaus können Bedingungen (*Conditions*) für den Gültigkeitszeitraum einer Authentifizierung (*NotBefore* und *NotOnOrAfter*) des Subjects existieren. Die digitale Identität ist somit für den Zeitpunkt `z2` bis zum Zeitpunkt `z3` beim Service Provider (*Audience*) authentifiziert.

In Assertions können Statements, wie zum Beispiel das *AttributeStatement* eingefügt werden. Das AttributeStatement beinhaltet eine oder mehrere Attribute. Attribute werden mit dem Namen des Attributes sowie dessen Wert angegeben.

2.6 Grid - Middleware

Unabhängig von der Hardwarearchitektur und der Systemarchitektur von verschiedenen Clustersystemen soll mit Hilfe einer *Grid-Middleware* Aufgaben im Grid auf verschiedene Ressourcen verteilt und dort bearbeitet werden können.
Die Aufgaben einer Grid-Middleware sind unter anderem [GRI06]:

- die Bereitstellung einer AAI,
- die Erstellung und Bearbeitung von Jobs,
- die Abgabe von Jobs (*Job Submission*),
- die Speicherung und Verteilung von Daten,
- die Überwachung von Ressourcen und Jobs (*Job-Monitoring*) und
- die Speicherung von Informationen zur Ressourcennutzung und Rechnungsstellung (*Accounting* und *Billing*).

Die drei bekanntesten Open-Source-Grid-Middleware sind UNICORE, Globus Toolkit und gLite. Im Folgenden werden die drei Middleware-Systeme beschrieben, wobei der Abschnitt konkreter auf die Architektur von UNICORE eingeht, da dieses Grid-Middleware-System das Zielsystem für die Implementierung im Rahmen dieser Arbeit ist.

2.6.1 UNICORE

UNICORE (Uniform Interface to Computing Resources) ist eine Grid-Middleware, die Zugriff auf Grid-Ressourcen ermöglicht.
UNICORE basiert seit der Version 6 auf Web-Services und stellt somit standardisierte Schnittstellen für die Grid-Middleware-Einsätze nach der OGSA des OGF bereit (siehe Unterabschnitt 2.1.3).

Wie aus Abbildung 12 ersichtlich, besteht UNICORE aus drei Schichten, nämlich aus einer User-, Server- und einer Targetsystemschicht.
Die *User-Schicht* repräsentiert den Arbeitsplatz des Benutzers, worauf die *UNICORE Client*-Komponente verwendet wird. Der *UNICORE-Gateway* und der *Network Job Supervisor* (NJS) sind UNICORE-Komponenten, die sich in der UNICORE Server-Schicht befinden. In der Targetsystemschicht wird mit Hilfe des *Target System*

Interface (TSI) auf die gewünschten Grid-Ressourcen zugegriffen. Im Folgenden werden die UNICORE-Komponenten mit ihren jeweiligen Diensten näher erläutert:

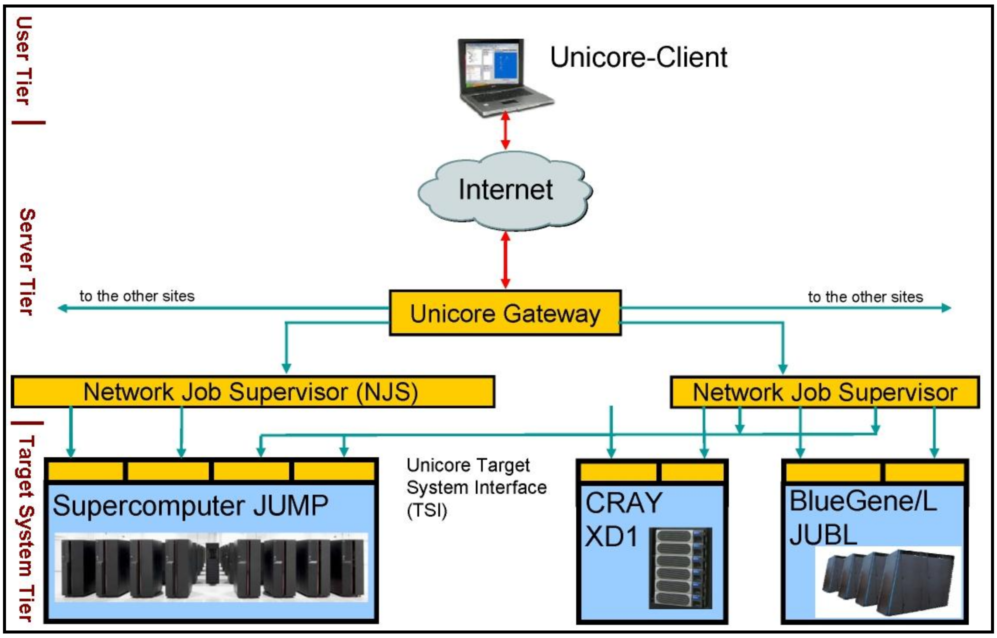

Abb.12: Die Architektur von UNICORE

UNICORE-Client

Der UNICORE-Client bietet ein Java-basiertes GUI (*Grafical User Interface*) an, welches den Benutzer dabei unterstützt, Jobs zu erstellen, zu verwalten und zu submittieren. Diese Unterstützung wird zum einen durch die *Job Preparation Area* (JPA), die für die Erstellung der Jobs zuständig ist, und zum anderen durch den *Job Monitor Controller* (JMC), der zur Kontrolle der abgeschickten Jobs dient, gewährleistet. Für das Ausführen der Jobs, wählt der Benutzer mit der JPA das gewünschte Zielsystem aus. Der Job wird nach der Erstellung in einem *Abstract Job Object* (AJO) abstrahiert, um seine Submittierung zu entfernten Systemen mit unterschiedlicher Software und Hardware-Architekturen zu ermöglichen. Der UNICORE-Client kann mit verschiedenen Servern, den *UNICORE Sites (Usites)*, verbunden werden.

Abbildung 13 stellt die Komponenten des UNICORE-Clients sowie dessen Konnektivität zu den weiteren Schichten dar.

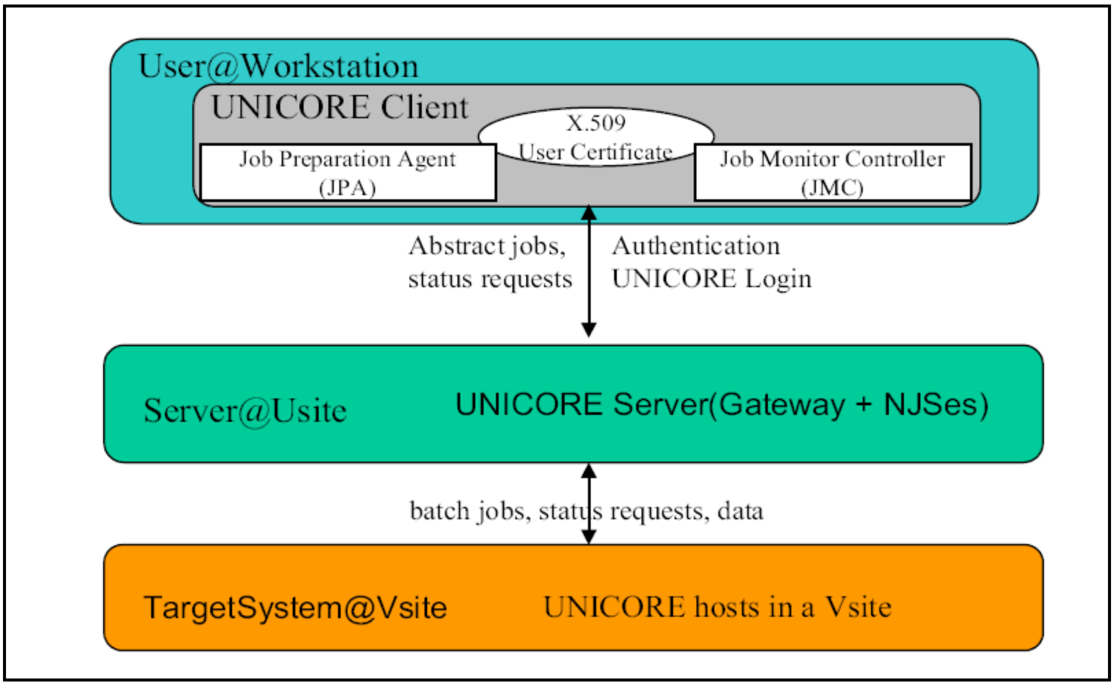

Abb.13: Der UNICORE Client [RAM02]

UNICORE Gateway

Der Gateway bildet mit dem Network Job Supervisor die Server-Schicht. Er authentifiziert den Benutzer mittels eines X.509 Zertifikats und bietet dann dem Benutzer eine Liste von vorhandenen, für ihn nutzbaren Computerressourcen (*Virtual Sites* bzw. *Vsites*) an. Der Gateway dient bei einer erfolgreichen Authentifizierung des Benutzers als Zuweiser, bei einer nicht erfolgreichen Authentifizierung als Abweiser zu Grid-Ressourcen. Ausserdem realisiert der Gateway einen einzigen Zugangspunkt zu den UNICORE Ressourcen durch die jeweilige Firewall.

Network Job Supervisor (NJS)

Der Network Job Supervisor ist für den Transfer der Jobs von einem Client zu einem Zielsystem, der ausgewählten Vsite, verantwortlich. Mit der Hilfe der *UNICORE User Database (UUDB)* wird eine Benutzeridentität autorisiert. Bei einer erfolgreichen Autorisierung bildet NJS eine Benutzeridentität auf ein *Unix-Login* des Zielsystems ab. Ferner übersetzt NJS unter der Verwendung einer *Incarnation Database* (IDB) die abstrakte Job-Beschreibung (AJO) in konkrete Anweisungen für die jeweilige Architektur des Zielsystems. Der NJS überwacht den Job-Status und sendet ihn über den Gateway an den Benutzer zurück.

Target System Interface (TSI)

Das Target System Interface (TSI) ist in Perl implementiert und führt die im NJS konkretisierten Anweisungen als Prozesse auf dem Zielsystem mit dem jeweiligen

Batch-System aus. Es bildet daher die Schnittstelle zwischen dem NJS und dem jeweiligem Grid-Ressourcen-System. Der NJS verschickt bei jedem Job den Benutzernamen mit, durch den der TSI im *Uspace*[9] sämtliche Jobs des Benutzers und dessen Daten ablegt.

2.6.2 Globus Toolkit

Das Globus Toolkit ist ein Open-Source-Projekt der Globus Alliance [11] und ist wohl das bekannteste Grid-Middleware System. Globus Toolkit ging aus einem Projekt namens I-WAY [12], welches 1995 von Ian Foster und Carl Kesselmann durchgeführt wurde, hervor. Die drei wesentlichen Bereiche in Globus sind Ressource Management, Information Services und Data Management. Das *Ressource Management* dient dem Verwalten der Ressourcen; die *Information Services* sind für die Verwaltung der Grid-Informationen zuständig und das *Data Management* hat die Aufgabe, Daten zu transferieren.

Mit Hilfe des serverseitigen *Grid Resource Allocation & Management System (GRAM)* können Jobs erzeugt, eingerichtet, gestartet und gestoppt werden. GRAM wird auch für das Reservieren von Ressourcen verwendet.

Der *Monitoring and Discovery Service (MDS)* ist für das Überwachen und Verwalten der Informationen zuständig. Die Dienste *GridFTP* und *Reliable File Transfer (RFT)* dienen dem Transfer der Daten.

Globus Toolkit der Version 4.0 (GT4) basiert auf Web-Services und ist konform bezüglich der OGSA Spezifikation.

GT4 wird ausschließlich über die Kommandozeile ausgeführt. Durch die zusätzliche Verwendung von *GridSphere* ist eine Bedienung über eine Web-Schnittstelle möglich.

Das Sicherheits-Modell in Globus basiert auf X.509-Zertifikaten.

Einsatz von Proxy-Zertifikaten für SSO und Delegation

In diesem Unterabschnitt wird anhand des Globus-Toolkits-Sicherheitsmodells GSI[10] beschrieben, wie man einen Single-Sign-On-Mechanismus durch den Einsatz von Proxy-Zertifikaten erreichen kann. Mit dem GRAM-Client wird die Proxy-Generierung, die Authentifizierung gegenüber dem *GRAM-Gatekeeper* sowie die Proxy-Delegation, die unten noch beschrieben wird, für den Benutzer transparent ausgeführt. Der Benutzer muss lediglich den Globus-Befehl `grid-proxy-init` eingeben und das Passwort seines Benutzerzertifikats, damit das Proxy-Zertifikat generiert werden kann.

[9] Uspace (User Space) ist ein temporäres Job-Verzeichnis
[10] GSI = Grid Security Infrastructure

GSI verwendet eine Zwei-Wege-Authentifizierung, während sie dafür das Proxyzertifikat des Benutzers verwendet. Dadurch braucht der Benutzer für die Authentifizierung kein Passwort einzugeben. Nach erfolgreicher Authentifizierung verwendet Globus eine Proxy-Delegation. Ziel dieser Delegation ist es, die Benutzeridentität auf den Benutzer abzubilden [RAM02]. Dadurch wird ein SSO-Mechanismus gewährleistet. Der Gatekeeper nimmt die Identität des Benutzers an und kann somit Grid-Dienste sowie Jobs ausführen, ohne dass sensible Daten ausgetauscht werden müssen.

Die Proxy-Delegation geschieht in folgenden Schritten: Der GRAM-Gatekeeper, der sich auf einem entfernten System befindet, erstellt ein zweites Proxy-Zertifikat für den Benutzer. Er speichert den geheimen Schlüssel lokal auf seinem Rechner und schickt ein Proxy-Zertifikats-Request an den GRAM-Client. Der GRAM-Client signiert das neue Proxy-Zertifikat mit dem Proxy-Schlüssel des zuvor erstellten Proxy-Zertifikats. Das neue, signierte Proxy-Zertifikat wird zurück zum Gatekeeper gesendet. Dadurch enthält das Proxy-Zertifikat ebenfalls die Identität des Benutzers und kann für weitere Delegationen verwendet werden.

Das Prinzip der Proxy-Delegation wird in Abbildung 14 veranschaulicht.

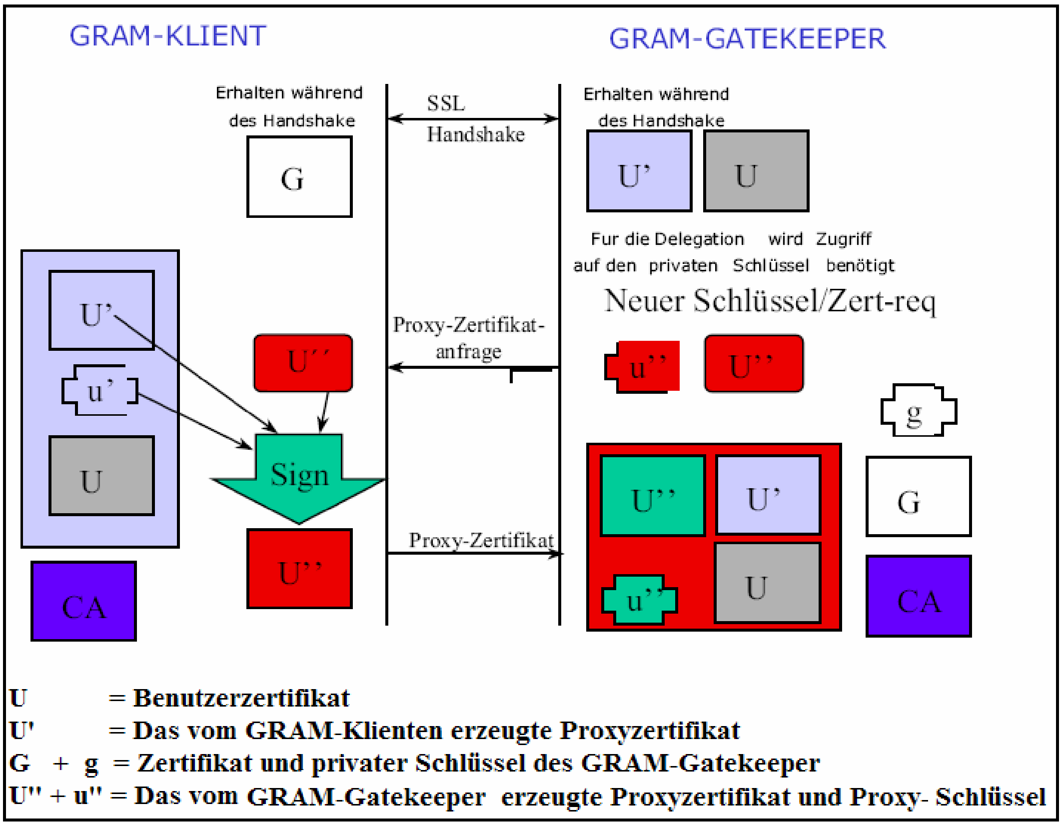

Abb.14: Proxy-Delegation [RAM02]

2.6.3 gLite

Die Middleware gLite wurde im Rahmen des Projektes *Enabling Grids for E-sciencE* (EGEE) entwickelt. gLite kann als eine Weiterentwicklung von bestehenden Grid-Middleware betrachtet werden, die die Datenverarbeitung der Experimente im *Large Hadron Collider* von CERN für die Hochenergiephysik ermöglicht. Ähnlich wie Globus unterstützt gLite Proxy-Zertifikate, Single-Sign-On und das Delegieren. Es verwendet im Gegensatz zu Globus einen Job Scheduler. gLite bietet unter anderem ein Workload Management- und ein leistungsfähiges Datamanagement-System an. Das *Workload Management System* (WMS) ist mit Hilfe des Schedulers für die Verteilung der Jobs zuständig. Das *Data Management System* hingegen ist für die Ablage von Dateien zuständig, das Buchführen über Metadaten wie Dateinamen, Ort der Speicherung und Zugangkontrolllisten. Der Transfer der Jobs kann mit Hilfe vom *GridFTP* realisiert werden. gLite bietet einen *Service Discovery* für die Abfrage der Eigenschaften einer Rechenressource an. Die Authentifizierung, Delegation und Single-Sign-On werden ähnlich wie bei Globus durch Proxy-Zertifikate realisiert.

gLite soll mit der Version 4 eine serviceorientierte Architektur aufweisen. Damit eine Interoperabilität zu anderen Grid-Services nach der Open Grid Service Architecture des Global Grid Forum möglich ist.

3. UNICORE AAI und VO Management

Dieses Kapitel beschreibt zunächst das Sicherheitsmodell von UNICORE. Es geht dann auf die Probleme des identitätsbasierten Ansatzes ein. Zum Schluss werden verschiedene VO-Technologien näher erläutert.

3.1 Das UNICORE-Sicherheitsmodell

Das Sicherheitsmodell von UNICORE basiert auf der Verwendung von permanenten X.509v3-Zertifikaten. UNICORE verwendet Zertifikate für:

- die Authentifizierung und Autorisierung des Benutzers
- die Authentifizierung der Server-Komponenten (Gateway und NJS)
- das Signieren von Jobs und Software Plugins

Zusätzlich bietet UNICORE die Möglichkeit des Single-Sign-On an.

3.1.1 Authentifizierung

Anhand der Authentifizierung werden hierbei UNICORE-Benutzer und die Server-Komponenten eindeutig identifiziert. Die sichere über SSL geschützte Kommunikation der UNICORE-Komponenten wird durch die gegenseitige Authentifizierung gewährleistet. Auch jeder UNICORE-Benutzer muss ein gültiges Zertifikat besitzen, das durch eine vertrauenswürdige CA signiert wurde. Das Benutzerzertifikat ist gleichzeitig die UNICORE-Benutzeridentifikation, die vom Gateway überprüft werden muss. Nur im Falle einer erfolgreichen Authentifizierung des Benutzerzertifikats sind Anfragen vom Client zum Zielsystem möglich. Die Anfragen werden dann an den UNICORE NJS weitergeleitet.

3.1.2 Autorisierung

Wie im Abschnitt 2.6.1 beschrieben, ist die UUDB für die Autorisierung des Benutzers zuständig. Die UUDB bildet nach einer erfolgreichen Autorisierung das Benutzerzertifikat auf einen lokalen Benutzer-Account der jeweiligen UNICORE-Site ab. In Abbildung 15 wird die AAI von UNICORE dargestellt.

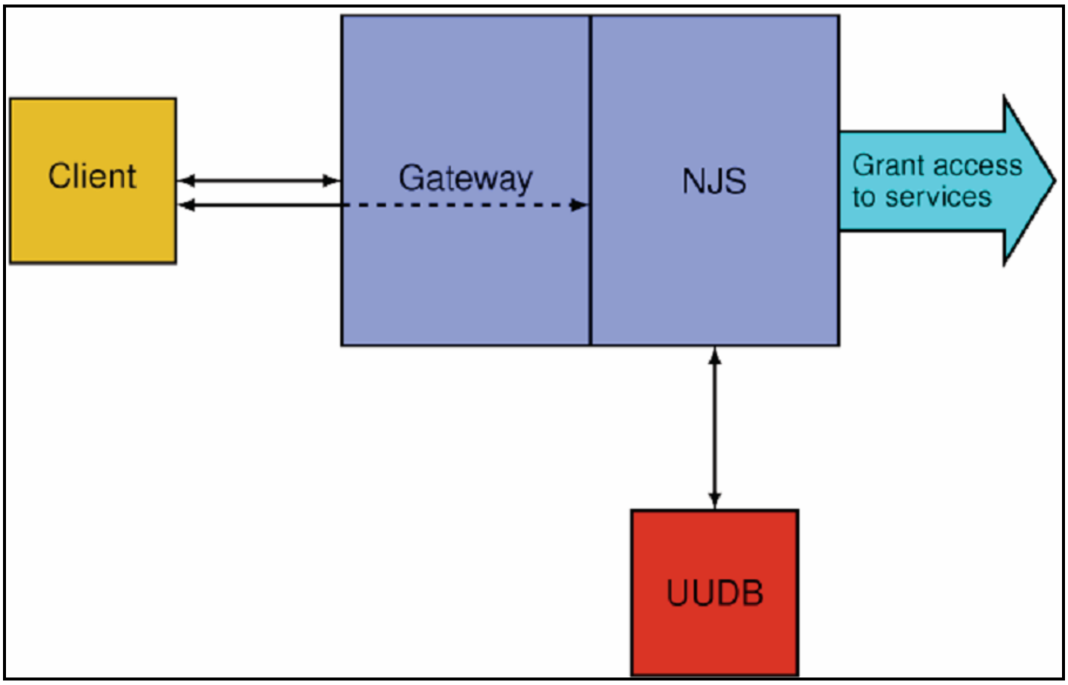

Abb.15: Die AAI von UNICORE [GRI06]

3.1.3 Integrität der Jobs

UNICORE signiert jeden Job mit dem privaten Schlüssel des Benutzerzertifikates und garantiert auf diese Weise die Integrität der Jobs bei deren Übermittlung zum Gateway. Ferner werden die Software Plugins signiert, um die vertrauenswürdige Erweiterung des UNICORE-Clients zu bestätigen.

3.1.4 Single-Sign-On in UNICORE

Die Unterstützung von Single-Sign-On in UNICORE wird dadurch realisiert, dass der Benutzer beim Starten seines UNICORE-Clients nur einmal aufgefordert wird, sein Passwort anzugeben. Durch das Passwort wird sein *Keystore* entsperrt und seine erstellten Jobs mit dem im Keystore vorhandenen privaten Schlüssel signiert. Der Benutzer muss sich für die Anwendung von UNICORE nur ein Passwort merken, der Austausch von Passwörtern über das Netz entfällt.

3.2 Probleme von identitätsbasierter Zugangsentscheidung

In den heutigen Grid-Infrastrukturen werden die Authentifizierungs- sowie Autorisierungsmechanismen mit X.509-Zertifikaten realisiert [FLT07]. Die Grid-Middleware UNICORE verwendet einen identitätsbasierten Autorisierungsansatz. Die UUDB ist vom Ansatz her vergleichbar mit der von Globus eingesetzten *Grid-Mapfile*. Beide Realisierungen binden die Subjekte s_i direkt an lokale Berechtigungen p_i. Die Berechtigungen beschreiben, ob dem Benutzer Zugang zu Objekten, wie zum Beispiel Dateien oder Services gewährt, oder verweigert wird. Konkreter formuliert realisieren diese beiden Ansätze ein Mapping zwischen den DN-Subjekten der Benutzerzertifikate und den lokalen Accounts (*userids*).
Abbildung 16 illustriert das Autorisierungsmodell:

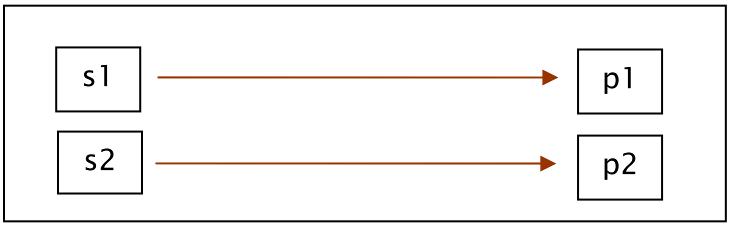

Abb.16: identitätsbasiertes Mapping

Durch diesen Ansatz muss der Administrator für jeden Grid-Benutzer auf jeder lokalen Maschine einen lokalen Account mit den dazugehörigen Berechtigungen erstellen. Die dadurch entstehenden Kosten sind proportional zu $U \times P$ [JIN05]. U steht für die Anzahl der Personen, die in einer Rollen-Gruppe einzuordnen sind und P für die Summe von Berechtigungen, die für eine Rolle benötigt werden.
Der dadurch entstehende administrative Aufwand ist gerade für Grid-Umgebungen, die eine große Anzahl von Grid-Benutzern aufweisen, viel zu hoch und dadurch ungeeignet. Der identitätsbasierte Autorisierungsansatz ist für Grid-Umgebungen zu statisch und unflexibel.

Aus diesem Grund sollen Grid-Systeme mit anderen Autorisierungsmethoden erweitert werden, die eine größere Flexibilität ermöglichen und den administrativen Aufwand reduzieren. Die Autorisierungsmethoden ABAC sowie RBAC sollen diese Anforderungen erfüllen. Da ABAC ein Oberbegriff für RBAC ist, wird in der weiteren Beschreibung nur der Begriff ABAC verwendet.
Dieses Autorisierungsmodell wird in Abbildung 17 verdeutlicht:

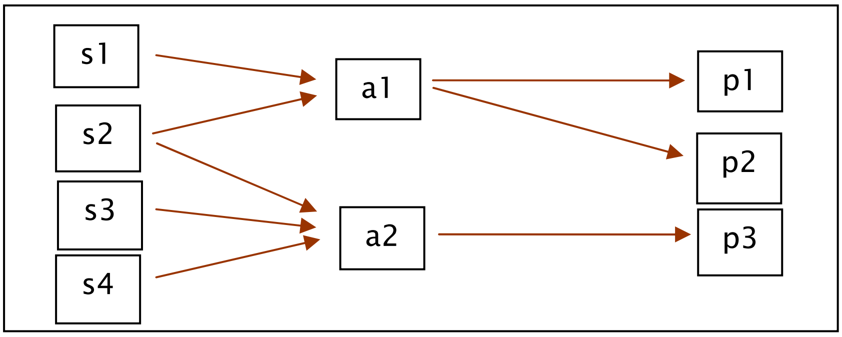

Abb.17: ABAC-Bindungen

Wie hier zu erkennen ist, werden die Subjekte s_i nicht direkt auf die Berechtigungen p_i, sondern auf die Attribute a_i abgebildet. Dadurch kann ein Subjekt mehrere Attribute besitzen und ein Attribut kann mehreren Subjekten zugeordnet werden. Die jeweiligen Attribute a_i werden auf die Berechtigungen p_i abgebildet. Dieses Modell erhöht die Flexibilität für die Verwaltung vieler Grid-Benutzer. Damit müssen die Verwalter nicht mehr die Identität ihrer Nutzer kennen, sondern nur noch ihre Attribute. Die dadurch entstehenden Verwaltungskosten für die Administration sind um einiges geringer als beim identitätsbasierten Ansatz. Sie betragen proportional lediglich $U + P$ [JIN05].

3.3 VO-Management

VO–Management bedeutet das Verwalten von Mitgliedschaften bezüglich ihrer Gruppen, Rollen und Rechte. Es regelt die Bildung von Kollaborationen [13].

3.3.1 Anforderungen an ein VO-Management-System

In Abschnitt 2.2 wurden die Grundlagen für die Virtuelle Organisation beschrieben. Dieser Abschnitt beschreibt, welche Anforderungen an ein Management-System bestehen. Die prinzipiellen Anforderungen sind nach [14] in vier verschiedenen Perspektiven herausgearbeitet worden:

- **Allgemeine Anforderungen:**
    - Bei einem VO-Management-System muss die Kommunikation mit einem Grid-Middleware-System gewährleistet sein. Deshalb sollte das VO-System auf existierenden Standards aufbauen. Solche Standards könnten X.509-Zertifikate oder SAML sein.
    - Ein VO-Management-System sollte möglichst mit mehreren Grid-Middleware zusammenarbeiten können, da die Wahrscheinlichkeit gering ist, dass sich ein einziges Middleware-System durchsetzen wird [14]. Dies würde die *Interoperabilität* zu verschiedenen Middleware-Systemen steigern.

- **Anforderungen der Benutzer:**
    - Die Benutzung des VO-Management-Systems muss möglichst einfach sein. Zum Beispiel soll ein Single-Sign-On-Mechanismus unterstützt werden. Der Benutzer soll nur einmal aufgefordert werden, sein Passwort einzugeben, um Zugang zu allen erlaubten Ressourcen bei seinem VO zu erhalten.
    - Der Benutzer muss die Möglichkeit haben, gleichzeitig Mitglied in verschiedenen VOs zu sein. Da Wissenschaftler an mehreren Projekten beteiligt sein können, sind solche Optionen unverzichtbar.

  - Der Benutzer soll verschiedene Rollen innerhalb einer VO besitzen können.

  - Eine Kopplung von VOs soll möglich sein, damit mehrere Projekte für bestimmte Zwecke zusammenarbeiten können.

  - Das Bilden einer VO muss einfach und schnell erfolgen können, damit eine kurzfristige Zusammenarbeit möglichst ohne Hindernisse erfolgen kann.

  - Die Benutzung von einem VO-Managament-System soll eine hohe Verfügbarkeit garantieren. Bei Ausfällen des Systems könnte sich der Benutzer nicht mehr gegenüber den Service-Anbietern autorisieren. Dieses Problem könnte durch Spiegelung oder durch eine dezentrale Struktur behoben oder stark minimiert werden.

- **Anforderungen der Service Provider**

  - Eine wesentliche Anforderung für den Service Provider ist die Identifizierbarkeit von Benutzern. Dadurch kann gewährleistet werden, dass nur berechtigte Personen Zugang zu Ressourcen des SPs haben.

  - Der SP soll auch bei einer dezentralen VO-Management-Struktur eine lokale Autonomie über Zugriffsautorisierungen besitzen. Er soll die letztendliche Entscheidung über den Zugriff auf seine Ressourcen treffen.

- **Anforderungen des VO-Administrators**

  - Ein VO-Administrator muss die Möglichkeit haben, verschiedene Zugriffsrechte strukturieren zu können. Es soll auch eine Einteilung in verschiedenen Gruppen möglich sein.

- Eine Virtuelle Organisation sollte *skalierbar* sein. Es muss möglich sein, eine Zusammenarbeit von vielen verschiedenen Wissenschaftlern in einer VO zu realisieren.

- Für den Administrator sollten *Logging*-Informationen des UNICORE-Systems und des VO-Management-System zugänglich gemacht werden. Alle Änderungen und Zugriffe auf das System sollten protokolliert werden, damit beim Auftreten eines Fehlers alle Vorgänge für den Administrator nachvollziehbar sind.

3.4 VO-Management-Systeme

In diesem Abschnitt werden VO-Management-Technologien beschrieben, die den Wissenschaftlern eine Kollaboration mit Grid Ressourcen ermöglichen sollen.

3.4.1 Shibboleth

Der Abschnitt geht zunächst allgemein auf das föderative Identitätsmanagement-System Shibboleth ein, beschreibt dann die Architektur von Shibboleth und schließlich die Interaktionen der wichtigen Shibboleth-Komponenten.

Was ist Shibboleth?

Shibboleth wurde vom Middleware Architecture Committee for Education (MACE) im Internet2 Konsortium [15] entwickelt. Shibboleth ist eine Open-Source-Software und unterstützt Single-Sign-On Mechanismen. Es ist ein verteiltes System zur inter-institutionellen Nutzung von zugangsgeschützten Web-Ressourcen. Shibboleth verwendet die Security Assertion Markup Language[11] (SAML) für die Kommunikation zwischen Identity- und Service Provider.
Shibboleth erfreut sich immer größerer Beliebtheit und Beachtung. In vielen Ländern werden bereits nationale Shibboleth-Infrastrukturen aufgebaut. So beispielsweise in den USA (Internet2), Australien (Projekt MAMS), Skandinavien, der Schweiz (SWITCH-AAI) und seit 2007 in Deutschland (DFN-AAI).

Der Name Shibboleth ist hebräisch und bedeutet wörtlich „Getreideähre". In diesem Zusammenhang ist die Bedeutung jedoch „Kennwort" oder „Codewort" [16]. Die Bezeichnung Shibboleth hat ihren Ursprung im Alten Testament. Im Buch Richter, Kapitel 12, Vers 5ff heißt es:

„Und die Gileaditer nahmen ein die Furt des Jordans vor Ephraim. Wenn nun sprachen die Flüchtigen Ephraims: Laß mich hinübergehen, so sprachen die Männer von Gilead zu ihm: Bist du ein Ephraiter? Wenn er dann antwortete: Nein, so hießen sie ihn sprechen: Schiboleth, so sprach er: Siboleth, und konnte es nicht recht reden. So griffen sie ihn und schlugen ihn an der Furt des Jordans, daß zu der Zeit von Ephraim fielen zweiundvierzigtausend."
[17]

Die „Authentifizierung" erfolgte aufgrund der Art und Weise der Aussprache von „Shibboleth". Nur „identifizierte" Männer aus Gilead waren „autorisiert", den Fluss Jordan zu überqueren. Die Bezeichnung Shibboleth wurde somit zu einem System für

[11] SAML wird im Abschnitt 2.5.1 detaillierter erläutert.

die Authentifizierung und Autorisierung von Individuen. Diese biblische Geschichte ist Ursprung der Bezeichnung „Shibboleth“ für das von Internet2 entwickelte System.

Shibboleth-Architektur

Die drei wesentlichen Komponenten von Shibboleth sind:

- **Identity Provider:**
  Der Identity Provider ist verantwortlich für das Versenden von Authentifizierungs- und Autorisierungsinformationen über seine Shibboleth-Benutzer. Dies geschieht in Form von Assertions. Der Identity Provider ist lokalisiert an der Heimateinrichtung von jedem Benutzer, an der dieser identifiziert wird. Der IdP beinhaltet zwei Dienste, den Handle Service (HS) und die Attribute Authority (AA).
  Der *Handle Service* authentifiziert den Benutzer und erstellt eine Attribute Query Handle in Form einer signierten SAMLResponse.
  Die *Attribute Authority* nimmt die Anfragen von dem Dienst Attribute Requester des Service Providers entgegen und beantwortet sie mit Attributinformationen des Benutzers.

- **Service Provider:**
  Der Service Provider ist für das Anbieten und das Schützen von Ressourcen zuständig. Er entscheidet nach den vom IdP geschickten Attributinformationen des Benutzers, ob der Benutzer autorisiert ist, Zugang zu einer Ressource zu erhalten. Die wichtigsten Dienste des Service Providers sind: der Assertion Consumer Service (ACS), der Attribute Requester (AR) und der Ressource Manager (RM).
  Der *Assertion Consumer Service* überprüft die Authentifizierungs-Assertion vom Handle Service.
  Der *Attribute Requester* ist dafür verantwortlich, Attributanfragen beim Identity Provider zu stellen.
  Der *Ressource Manager* schließlich entscheidet, basierend auf den Attributen des Benutzers, ob dem Benutzer der Zugang zu einer Ressource verweigert oder gestattet wird.

- **Where Are You From Service:**
  Der *Where Are You From Service* (WAYF) unterstützt die Kommunikation zwischen dem Identitiy Provider und dem Service Provider. Er bietet einen Mechanismus an, um den Benutzer von einer angeforderten Ressource, einem angeforderten Service zu seiner Heimateinrichtung weiterzuleiten. Der WAYF

Service zeigt dem Benutzer eine Liste von IdPs, aus der er dann seinen eigenen Identity Provider selektieren kann.

In der Abbildung 18 wird die Interaktion der einzelnen beschriebenen Komponenten deutlich.

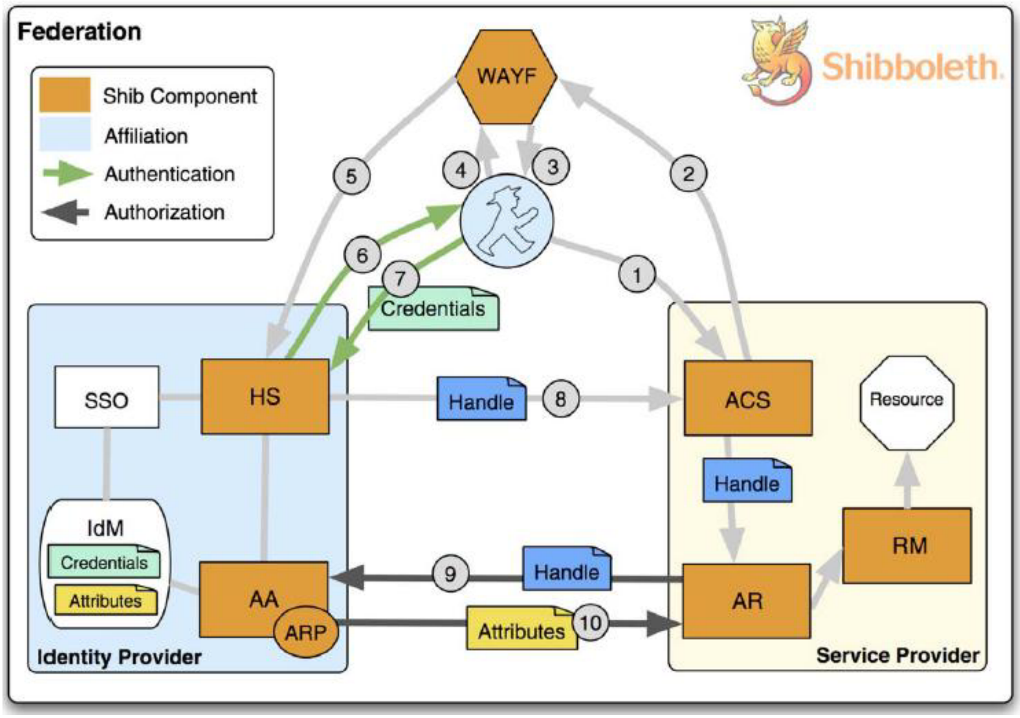

Abb.18: Shibboleth-Framework [GRI06]

Die Übergabe von Attributen des Benutzers an einen Service-Provider erfolgt durch Shibboleth in 10 Schritten:

1. Der Benutzer versucht Zugang, zu einer vom Service Provider geschützten Ressource zu erhalten.

2. Der Service Provider benötigt vom Benutzer eine erfolgreiche Authentifizierung und leitet ihn daher mit einem Browser-Redirect zum „Where Are You From“-Service weiter, damit er sich danach bei seiner Heimateinrichtung authentifizieren kann.

3. Der WAYF Service listet dem Benutzer die in der Föderation vorhandenen Identity Provider auf.

4. Der Benutzer wählt seinen Identity Provider aus.

5. Der WAYF Service leitet wiederum per Redirect den Benutzer zu seinem gewählten Identity Provider (Handle Service) weiter.

6. Der Handle Service fordert den Benutzer auf, sich zu authentifizieren.

7. Der Benutzer authentifiziert sich auf vertrautem Wege gegenüber seinem IdP.

8. Der Handle Service überprüft die Identifizierung des Benutzers. Nach einer erfolgreichen Authentifizierung erstellt er einen eindeutigen Handle und sendet diesen zum Service Provider.

9. Der Attribute Consumer Service des Service Providers generiert eine Session und übergibt den erhaltenden Handle an den Attribute Requester. Der AR nutzt das Handle, um Attribute des Nutzers bei der Attribute Authority (AA) des IdP abzufragen.

10. Die Attribute Authority liefert unter Berücksichtigung der Attribute Release Policy (ARP) eine Attribute Assertion mit den Attributen des Benutzers an den AR zurück. Der SP entscheidet anhand der erhaltenen Attribute über die Gewährung und die Art des Zugangs.

Die Vorteile Shibboleths gegenüber identitätsbasierter AAI

Vergleicht man Shibboleth mit identitätsbasierter AAI, ergeben sich für Shibboleth nach [GRI06] folgende Vorteile:

- Der verteilte Ansatz von Shibboleth vereinfacht ein Management-System gegenüber einer zentralen Management-Lösung.

- Durch die dezentrale Struktur und die attributbasierte Autorisierung erzielt Shibboleth eine hohe Skalierbarkeit.

- Shibboleth bietet einen guten Schutz der Privatsphäre, zum einen wegen der ausschließlichen Pflege und der Vorhaltung der Benutzerinformationen an dessen Heimatorganisation und zum anderem, weil ausschließlich der Benutzer die Kontrolle über die für die Autorisierung verwendeten Informationen hat.

- Da Shibboleth über SAML Informationen über Authentifizierung und Autorisierung des Benutzers austauscht, kann eine hohe Zuverlässigkeit der Benutzerinformationen erreicht werden, sofern ein gut gepflegtes und sicheres Identity-Management-System eingesetzt wird.

- Durch die Verwendung von Metadaten für die Bildung einer Föderation ist eine unmittelbare Einbindung großer Nutzerdatenbanken in einer Föderation ohne hohen Aufwand zu bewerkstelligen.

3.4.2 Shibbolisierung des Grids

Das Shibboleth-System wurde nicht für das Management von Virtuellen Organisationen oder für die Grid-Anwendung entwickelt, vielmehr für das Bilden einer Infrastruktur zur einheitlichen und organisationsübergreifenden Authentifizierung und Autorisierung von Nutzern. Shibboleth ist aber nicht nur für Organisationen, Institute und Universitäten interessant, sondern weckt auch großes Interesse für die Grid-Anwendung. Daher wird zunehmend von der „Shibbolisierung des Grids" gesprochen [GRI06]. Die folgenden Projekte

- GridShib (USA),
- MAMS (Australien),
- SHEBANGS/GridSite (Schweiz),
- Shibboleth in EGEE2 (Schweiz) und
- ShibGrid (Großbritannien)

versuchen eine Integration von Shibboleth in das Grid zu erreichen. Die Vielzahl der Projekte verdeutlicht, dass der Einsatz von Shibboleth in Grids zunehmend von Interesse ist. Shibboleth könnte durch die Integration im Grid die Authentifizierung der Grid-Benutzer übernehmen.
Zukünftig soll Shibboleth auch für das Managen von VOs verwendet werden.

3.4.3 VO-Management per Shibboleth

Shibboleth stellt keine direkten Funktionen für das Managen von Virtuellen Organisation bereit. Deshalb müssen Ansätze entwickelt werden, mit denen Shibboleth um VO-Mechanismen erweitert werden kann.

Zu beachten ist, dass das Konzept von Shibboleth nur eine Attribute Authority (AA) für die Heimateinrichtung des Nutzers gestattet. Eine Virtuelle Organisation benötigt aber ebenfalls eine AA. In Hinblick auf diese Problematik wurden nach [GRI06] einige Ansätze diskutiert und vorgeschlagen:

1. Bei diesem Ansatz wird das Managen von VO bei allen Identity Providern betrachtet. Die IdPs der Virtuellen Organisation einigen sich auf bestimmte Schemata und Attribute für deren VO-Benutzer.

2. Die VO wird durch einen eigenen IdP realisiert. Dieser Ansatz entspricht den Vor- und Nachteilen von VOMS (siehe Abschnitt 3.4.5).

3. Die VO-spezifischen Attribute eines VO-Mitgliedes werden im Identity Management der Heimateinrichtung verwaltet. Dies bedeutet aber, dass die VO bestimmte Schreibrechte auf dem IdM besitzen muss.

Abbildung 19 stellt Shibboleth mit zentralem VO–Management dar. Diese Architektur entspricht dem zweiten Ansatz. Die VO ist selbst ein Identity Provider, welcher die VO-Attribute seiner Mitglieder verwaltet. Das VO-Management dient als Proxy für die Kommunikation von der Heimatorganisation des Benutzers bis zu den gewünschten Ressourcen des Service Providers. Das VO-Management dient der Heimatorganisation des Benutzers als Service Provider und den gewünschten Ressourcen des Service Providers dient es als Identity Provider.

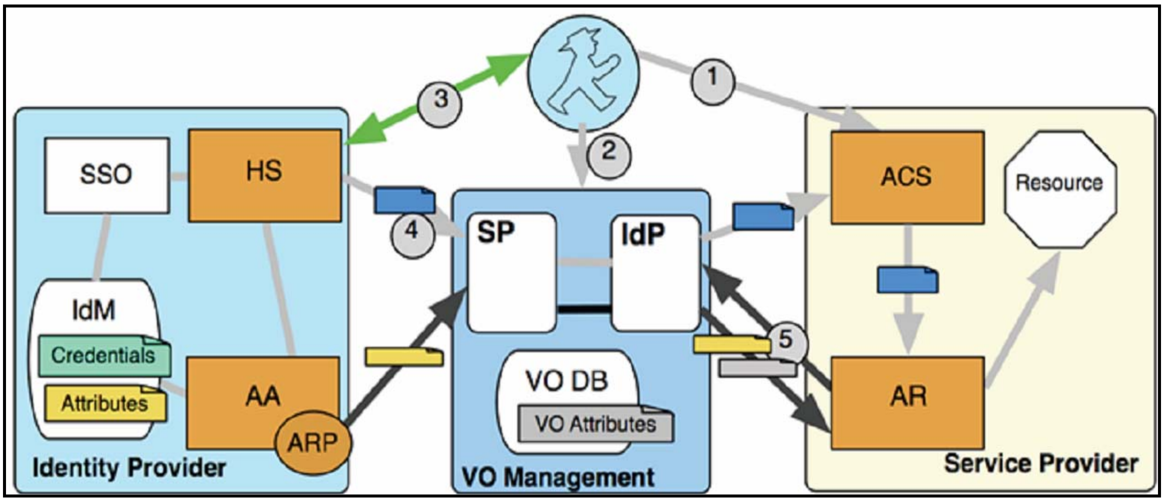

Abb.19: Shibboleth mit zentralem VO Management [MAK06]

Die Ansätze 1 und 3 entsprechen dem dezentralen Ansatz des VO-Management-Systems und somit dem Konzept von Shibboleth, in dem Identitäten sowie deren Rechte und Rollen an einem Ort verwaltet werden.

3.4.4 MyVocs

Ein Projekt, das Shibboleth um VO-Mechanismen erweitert, ist MyVocs.
*MyVocs (**V**irtual **o**rganization **c**ollaboration **s**ystem)* wird von der University of Alabama (USA) entwickelt und erlaubt einen Aufbau von virtuellen Organisationen (VO). MyVocs bietet in Kombination mit GridShib VO-Zugriff auf Grids.
MyVocs basiert auf dem zentralen VO-Management-Ansatz und dient als Brücke zwischen den Identity Providern einer Föderation und den Service Providern einer VO. Dies wird in der folgenden Abbildung deutlich.

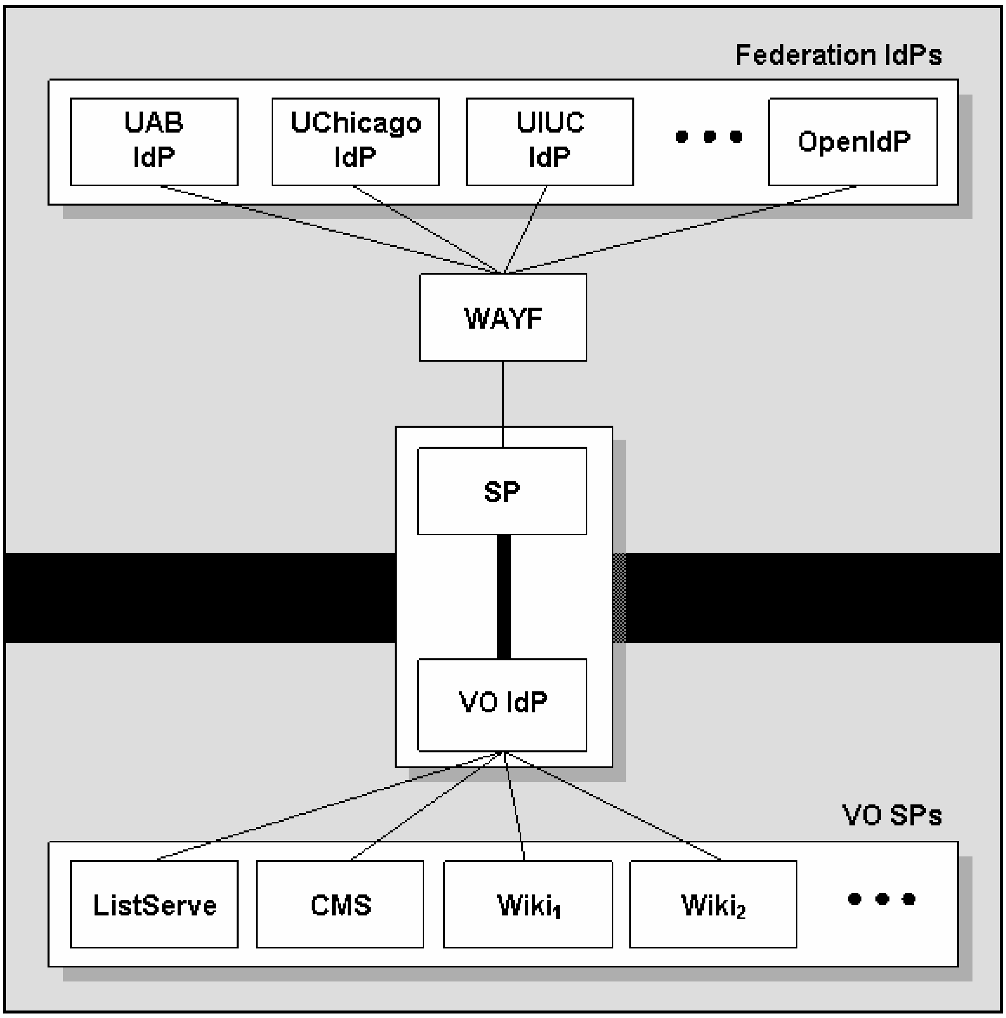

Abb.20: VO-Management durch MyVocs [18]

Für die Identity Provider einer Föderation dient MyVocs als Service Provider. Aufgrund dessen können sich die Dienste der VO sicher sein, dass der Benutzer authentisiert ist. MyVocs kann VOs bilden und die VO-Attribute der VO-Benutzer managen. Das Projekt

erlaubt verteilten Anwendungen, die einer VO angehören (z.B. CMS, Wiki und Maillist Service) als Service Provider zu fungieren. Diese Anwendungen werden als Service Provider einer VO bezeichnet.
MyVocs dient den Service Providern der VO als VO-Identity Provider. Dieser gibt Attributinformationen über einen VO-Benutzer an die SPs der VO weiter. Die SPs der VO benötigen die Attribute der VO-Benutzer, um Autorisierungsentscheidungen treffen zu können. Ein wichtiges Merkmal von MyVocs ist, dass ein einzelner Service Provider mehrere verschiedene VOs besitzen kann.

3.4.5 GridShib

Die Open-Source-Software GridShib ist ein Projekt der *NCSA* (National Center for Supercomputing Applications) und der Universität Chicago. Durch GridShib soll das Grid-Middleware Globus-Toolkit mit dem System Shibboleth kombiniert werden. GridShib wird in drei Bereiche unterteilt:

- **GridShib-for-Globus-Toolkit**
  GridShib-for-Globus-Toolkit ist ein Plugin für GT 4, welches die SAML-Assertions, die in den SLC-Extensions gespeichert sind, konsumieren kann.

- **GridShib-for-Shibboleth**
  GridShib-for-Shibboleth ist ein Plugin für den Shibboleth Identity Provider. Durch das Plugin wird die Attribute-Authority-Komponente um die Möglichkeit erweitert, Attributanfragen vom GridShib-for-Globus-Toolkit in Form von SAML-Assertions beantworten zu können.

- **GridShib-CA**
  Die GridShib-CA erstellt SLCs für die Shibboleth-Benutzer. Die Credentials können für die Verwendung von Grid-Ressourcen benutzt werden.

Funktionsweise der GridShib-CA

Die Kluft zwischen einer Shibboleth- und einer Grid-Identität wäre mit der Erstellung von einem SLC überbrückt. Der DFN betreibt seit 2007 mit der Verwendung der GridShib-CA einen SLCS [GRO06]. Die GridShib-CA bietet zusätzlich ein Demo an [20], das für einen Shibboleth-Benutzer für weniger als einen Tag SLC erstellt. Voraussetzung hierfür ist, dass der Benutzer eine Mitgliedschaft in der Inqueue-

Föderation[12] [23] oder in einem der beiden öffentlichen Identity Provider OpenIdp [21] bzw. Protect-Network besitzt [22].

Die GridShib-CA wurde in Java und durch Perl-Skripte implementiert. Die untere Abbildung stellt die wichtigen Skripte, die für die Erstellung eines SLC zuständig sind, dar. Unterschieden wird bei der GridShib-CA zwischen ungeschützten und durch shibboleth geschützten Bereichen.

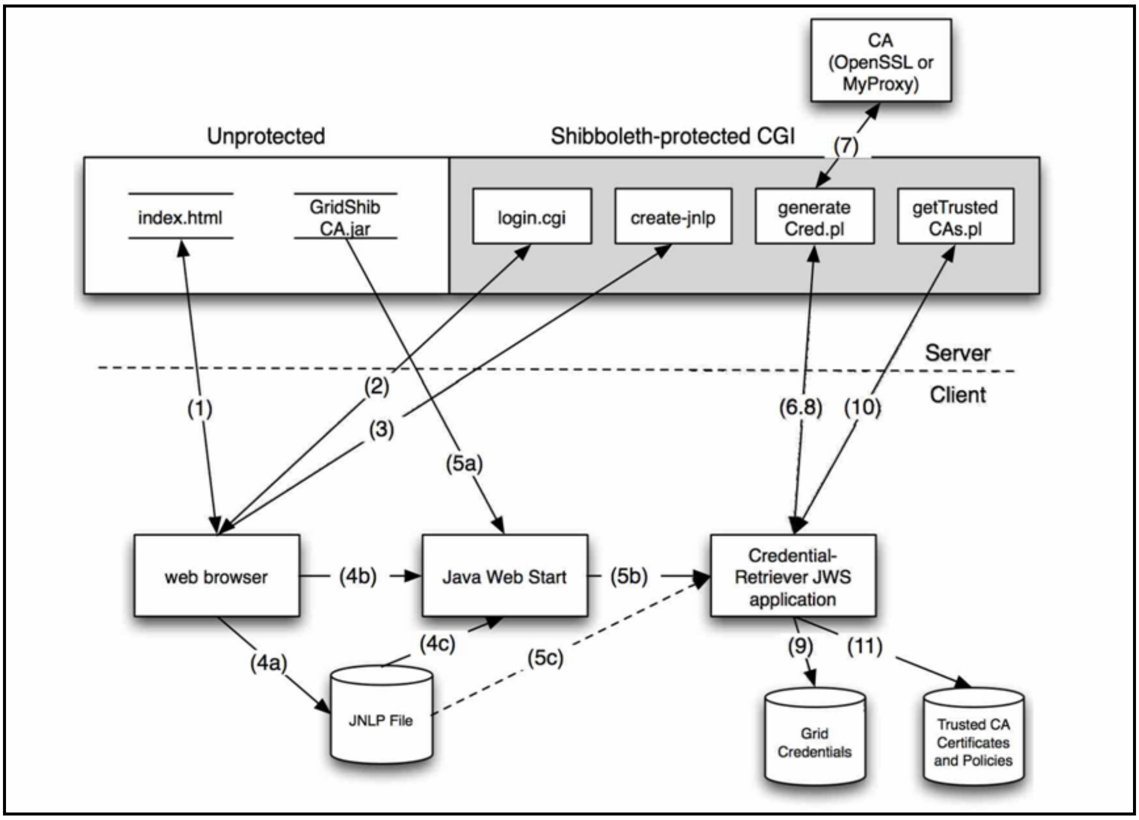

Abb.21: Funktionsweise der GridShib-CA [19]

Die Prozess-Schritte für die Erstellung einer SLC sind:

1. Der Benutzer besucht mit seinem Browser die `index.html-Seite` der GridShib-CA. Dort selektiert er seine Föderation.

2. Der Benutzer wird von der Föderation zu seiner Heimatorganisation weitergeleitet, wo er sich durch seine Credentials authentifiziert. Bei einer erfolgreichen Authentifizierung wird er zum `login.cgi` weitergeleitet. Die Aufgabe des Skriptes `login.cgi` ist es, sicherzustellen, dass es sich um einen gültigen Shibboleth-Benutzer handelt. Dort kann der Benutzer die

[12] Die Inqueue-Föderation ist seit Juni 2007 nicht mehr im Betrieb.

Gültigkeitsdauer seines SLC eingeben und durch einen Button die GridShib-CA auffordern, SLC zu erstellen. Danach wird das Skript `create.jnlp` gestartet.

3. `create-jnlp` erstellt eine `JNLP`-Datei um eine JAVA Web-Anwendung zu starten. Diese Datei wird für die weitere Ausführung mit Argumenten (wie z.B. Shibboleth-Session, Token) gefüllt.

4. Nach der Erstellung der `JNLP`-Datei wird durch den Web-Browser des Benutzers die JAVA-Anwendung gestartet.

5. Die Java-Web-Anwendung lädt die `GridShib.jar` herunter und startet dann die Klasse `CredentialRetriever`.

6. Der `CredentialRetriever` führt das Skript „`generateCred.pl`“ aus.

7. `generateCred.pl` validiert den erhaltenden Token, generiert ein Schlüsselpaar, erstellt eine `Certificate.Request` und gibt diese dem GridShib-CA weiter.

8. Die GridShib-CA erstellt eine SLC und gibt diese dem `Credential-Retriever` zurück.

9. Der `CredentialRetriever` schreibt das Zertifikat und den privaten Schlüssel des Benutzers in dessen Tmp-Directory.

10. Danach wird das Skript `getTrustedCAs.cgi` ausgeführt, in dem eine Liste der vertrauenswürdigen CA dem Benutzer weitergegeben wird.

11. Die vertrauenswürdigen CAs werden dann im Client-Rechner gespeichert.

3.4.6 VOMS

Das System VOMS[13] ermöglicht es, virtuelle Organisationen zu erstellen und zu verwalten. Es ist extra für verteilte Systeme (Grid-Computing) konzipiert. Die Autorisierung basiert auf RBAC und wird mit Hilfe von X.509v3-Proxy-Zertifikaten realisiert. Die X.509v3-Zertifikate besitzen die Möglichkeit, ein Zertifikat mit Extensions (Erweiterungen) zu füllen. Diese Möglichkeit nutzt VOMS, um Proxy-Zertifikate zu

[13] VOMS = Virtual Organization Membership Service

erstellen. Diese sind um eigene VOMS-Informationen erweitert. Im Proxy-Zertifikat sind die VO-Rollen des Benutzers gespeichert. Dies geschieht folgendermaßen:
Ein Benutzer generiert zunächst mit dem VOMS-Befehl `voms-proxy-init` sein Proxy-Zertifikat. Mithilfe vom VOMS Core Service wird überprüft, ob er Mitglied einer VO ist und welche Rollen er besitzt. Die Autorisierungsdaten werden in VOMS in Form von 3-Tupel gespeichert:
„/VO[/group[/subgroup(s)]][/Role=role][/Capability=cap"]. Diese werden auch FQAN[14] genannt und dem Proxy-Zertifikat des Benutzers hinzugefügt. Das Proxy-Zertifikat kann so für die Authentifizierung sowie Autorisierung verwendet werden. Nachdem sich der Benutzer damit zum Beispiel auf `Host c` authentifiziert hat, untersucht `Host c`, ob die Rolle (FQAN) des Benutzers in seiner Datenbank (*gridmap file*) abgelegt ist. Die Datenbank bildet die FQAN auf gewöhnliche UNIX-Pool-Accounts ab. Durch dieses Mapping kann der Benutzer mit den Zugriffsrechten des Pool-Accounts auf Daten zugreifen.

VOMS-Architektur

Das VOMS-System besteht aus vier Basiskomponenten [ALF05]:

- **User Server:**
  Die Aufgabe des User Servers ist, Client-Anfragen zu empfangen und Informationen über den angefragten Benutzer zurück zu senden.

- **User Client:**
  Der User Client kontaktiert den User Server mit dem Benutzerzertifikat, authentifiziert den Benutzer gegenüber dem Server und erstellt ein Proxy-Zertifikat mit VO-FQAN-Extensions.

- **Administration Client:**
  Der Administration Client wird vom VO-Administrator verwendet, um Änderungen an den VOs vorzunehmen. Zum Beispiel können in der VO-Datenbank VO-Rollen verändert, gelöscht oder hinzugefügt werden.

- **Administration Server:**
  Der Administration Server hat die Aufgabe, Administrationsanfragen entgegen-zunehmen und die Datenbank zu aktualisieren.

Folgende Abbildung beschreibt das Client-/Server-Modell von VOMS:

[14] FQAN = Full-Qualified Attribute Name

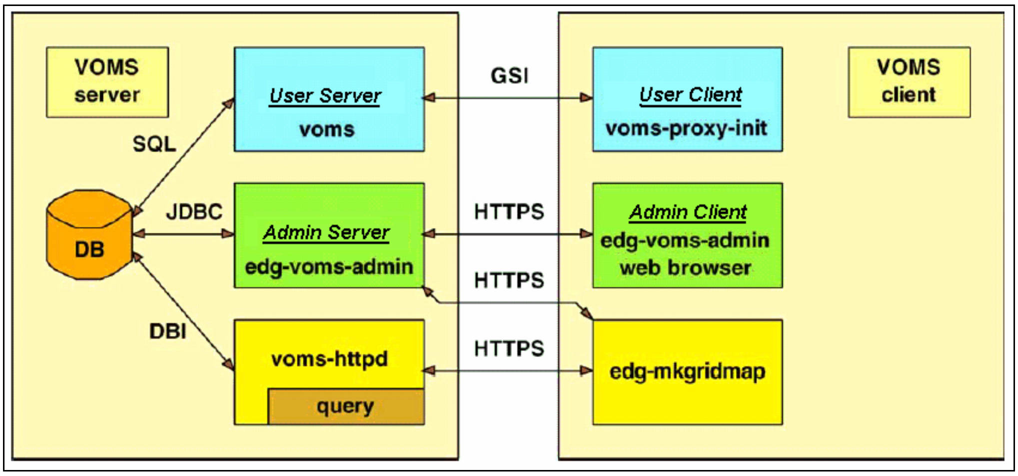

Abb.22: Das VOMS System [ALF05]

In der Abbildung werden unter anderem die Komponenten `voms-httpd` und `edg-mkgridmap` dargestellt. Sie sind für die Erstellung und Aktualisierung des Grid-Mapfiles zuständig, aber nicht zwingend für ein VOMS-System notwendig. Wie man sieht, wurden die VOMS-Komponenten in ihre Client- und Server-Bestandteile zerlegt.

Wie schon beschrieben, muss der Benutzer `voms-proxy-init` ausführen, bevor er Zugang zum Grid bekommt. Die folgende Liste sowie die Abbildung 23 beschreiben die Prozedur detaillierter:

- Mittels Zertifikaten authentisieren sich der Client und der VOMS-Server gegenseitig;
- Der Client sendet den Request an den VOMS-Server;
- Der VOMS-Server verifiziert die Identität des Benutzers und prüft die Anfrage auf ihre syntaktische Korrektheit;
- Der VOMS-Server sendet die geforderten Autorisierungsinformationen in Form eines Attributzertifikats (RFC 3281) an den Client. Das Attributzertifikat (AC) wird vom VOMS-Server signiert;
- Der Client prüft das empfangene Attributzertifikat auf seine Korrektheit;
- Optional können diese Schritte für andere VOMS-Systeme wiederholt werden;

- Der Client erstellt ein Proxy-Zertifikat und fügt die empfangenen Autorisierungsinformationen als (non-critical) extension im Proxy-Zertifikat hinzu;

- Auch andere Authentifizierungsinformationen können an das Proxy-Zertifikat angefügt werden;

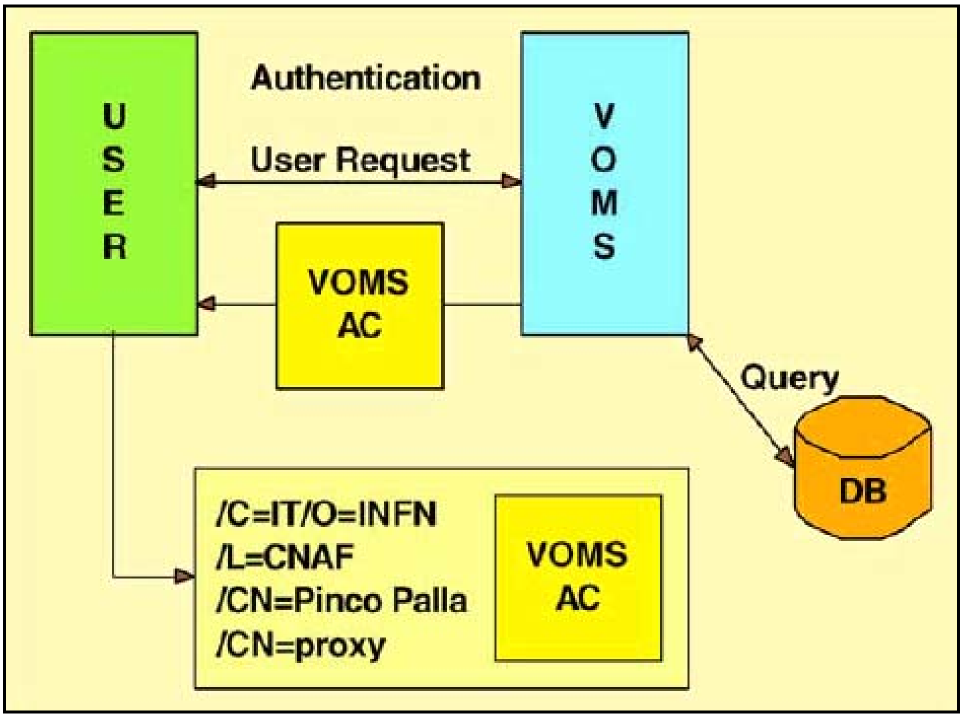

Abb.23: VOMS Zertifikatserstellung [ALF05]

3.4.7 VOMRS

Virtual Organisation Membership Registration Service (VOMRS) wurde im Rahmen des VOX-Projektes (VO eXtension) vom *Fermilab* entwickelt. VOMRS ist ein zu VOMS komplementäres System, das als eine Erweiterung von VOMS angesehen werden kann.

Es verwendet im Gegensatz zu VOMS automatisierte Registrierungsprozesse. Dies reduziert den Aufwand des VO-Administrators. Desweiteren bietet VOMRS die Möglichkeit, mehr Informationen über ein VO-Mitglied zu speichern. VOMRS gewährleistet eine Kommunikation mit einem VOMS-System bzw. mit dem VOMS-Administrator. Eine Verbesserung gegenüber dem VOMS-System besteht auch darin, dass einem VO-Mitglied die Zugehörigkeit zu einer VO vorübergehend entzogen werden kann, anstatt wie ein VO-Mitglied vollständig aus der VO zu entfernen.

3.5 Motivation für die UNICORE-Erweiterung

Das UNICORE-Sicherheitsmodell basiert auf X.509-Zertififikaten, die für die Authentifizierung sowie Autorisierung Verwendung finden. Die UNICORE-Authentisierung und Autorisierung verfolgt einen identitätsbasierten Ansatz. Dieser führt jedoch zu Problemen: Zum einen erfordert dieser Ansatz einen hohen administrativen Aufwand im Verwalten von Identitätszertifikaten und zum anderen erweist sich eine solche Autorisierung als nicht skalierbar, da für die Abbildung von Zertifikaten auf einen lokalen Account die Identitäten aller Benutzer bekannt sein müssen. Aufgrund dieser Probleme wird eine attributbasierte Autorisierung in UNICORE angestrebt.

Die attributbasierte Autorisierung soll das Verwalten der UNICORE User Database (UUDB) von der Last befreien, Zugangsentscheidungen auf der Basis der Identität eines Benutzers zu treffen. Die Entscheidung soll nun aufgrund der Benutzerattribute erfolgen. Dadurch müssen in der UUDB nur noch die Attribute der Benutzer bekannt sein.

Mit der Integration von UNICORE und den VO-Technologien kann eine identitätsbasierte Autorisierung durch eine attributbasierte ersetzt werden. UNICORE ist von den gängigen Middleware-Systemen das einzige, welches noch keine Integration von Shibboleth oder VOMS anbietet. Globus Toolkit und gLite bieten hingegen eine Integration von VOMS an. In der Distribution von gLite ist VOMS integriert und für die Autorisierung ein wichtiger Bestandteil. Globus Toolkit und gLite ermöglichen eine Integration von Shibboleth. Bei einer erfolgreichen Integration von VOMS und Shibboleth in UNIOCRE wäre die Möglichkeit eines übergreifenden VO-Managements gegeben, das auch unabhängig vom jeweilig verwendeten Grid-Middleware benutzt werden kann.

4. UNICORE-Erweiterung durch VO-Technologien

Für die UNICORE-Erweiterung durch VO-Technologien werden in diesem Kapitel verschiedene Anforderungen vorgestellt, die bei der späteren Konzeption sowie Implementierung berücksichtigt werden. Desweiteren beschreibt das Kapitel die Einschränkungen der Kreativität bezüglich der Konzeptionsmöglichkeiten.

4.1 Anforderungen für die UNICORE-Integration durch VO-Technologien

Für die Integration der VO-Technologien in UNIOCRE wurden Anforderungen aufgestellt, die in drei Kategorien unterteilt werden:

- Allgemeine Anforderungen an die Integration
- Anforderungen aus Benutzer-Sicht
- Anforderungen aus Administrations-Sicht

4.1.1 Allgemeine Anforderungen an die Integration:

Im Allgemeinen lassen sich folgende Anforderungen an die Integration selbst aufstellen:

1. UNICORE soll durch Plugins und veränderbare Komponenten erweitert werden. Der UNICORE-Kern soll hierbei unverändert bleiben.
2. Die Autorisierung von UNICORE soll anhand von Benutzerattributen erfolgen. Sie soll nach dem ABAC-Modell fungieren.
3. Die UNIOCRE-Authentifizierung soll möglichst nicht verändern werden.
4. Es sollen Möglichkeiten zur Anpassung der VO-Autorisierung angeboten werden.
5. Die Veränderungen sollen das UNICORE-Sicherheitsmodell nicht verletzen.
6. Eine Interoperabilität zu anderen Grid-Middleware-Systemen sowie zu verschiedenen VO-Technologien soll gewährleistet sein.

7. Es sollen Lösungen gefunden werden, die eine direkte Verwendung der UNICORE-Erweiterung in D-Grid ermöglichen.

Punkt 1 besagt, dass UNICORE nach Erweiterungsmöglichkeiten untersucht werden muss, um nicht die Kern-Implementierung des Systems zu modifizieren. Die Integrationsimplementierung soll nicht dazu führen, dass eine neue Version von UNICORE bzw. seiner Systemkomponenten notwendig wird. Sie soll das bereits bestehende UNICORE-System um zusätzliche Funktionen erweitern.

Die Anforderung, die *Punkt 2* beschreibt, entspricht dem Haupt-Ziel der Integration: UNICORE soll eine fein-granulare Autorisierung auf der Grundlage von Attributen ermöglichen. Dadurch sollen die vorherigen Probleme bezüglich der Skalierbarkeit und der aufwendigen Administration reduziert werden und eine gute Basis für die Integration eines VO-Management-Systems gewährleistet sein.

Punkt 3 beschreibt, dass die durch X.509-Zertifikate geregelten Authentifizierungsmechanismen von UNICORE soweit wie möglich unverändert bleiben sollen. Das System soll nur um VO-Autorisierungsmechanismen sollen erweitert werden.

Unter *Punkt 4* wird die Möglichkeit der Communities beschrieben, die UNICORE-Autorisierungsentscheidungen, basierend auf deren VO-Umgebung, anpassen zu können.

Durch das Sicherheitsmodell von UNIORE werden die Schutzziele Authentizität, Vertraulichkeit und Integrität erreicht. Die Notwendigkeit der Erhaltung dieser Schutzziele als Anforderung an die Integrationslösung wird unter *Punkt 5* aufgeführt.

Das D-Grid-Projekt IVOM hat die Aufgabe, für die Communities in D-Grid eine gemeinsame Grid-Infrastrukur aufzubauen. Für die Lösung dieser Aufgabe muss eine Interoperabilität der im Community-Grid eingesetzten VO-Management-Systeme vorausgesetzt werden. Die in den Communities eingesetzten Technologien für das VO-Management sind zurzeit stark von der jeweiligen Middleware abhängig. Ein Schwerpunkt von IVOM und das Ziel der hier vorliegenden Arbeit ist es, UNICORE um die bislang fehlende Funktion, die auf einer VO-basierten Autorisierung beruht, zu erweitern. Demzufolge soll den Communities die Möglichkeit gegeben werden, VOs unabhängig von Grid-Middleware zu bilden. Es soll eine Interoperabilität zu anderen Middleware, die VO-Technologien verwenden, gefunden werden (*Punkt 6*).

4.1.2 Anforderungen aus Benutzer-Sicht

Zunächst wird vorausgesetzt, dass der Benutzer Erfahrung im Umgang mit dem UNICORE-System sowie dem eingesetzten VO-System hat. Folgende Anforderungen wurden erarbeitet, die dem Benutzer den Umgang mit dem System erleichtern sollen:

1. Die „erweiterte“ UNICORE-AAI soll für den Benutzer möglichst transparent sein.

2. Der Benutzer soll seine Identität durch wenige Benutzerinteraktionen um seine VO-Credentials erweitern können.

3. Es muss dem Benutzer möglich sein, den UNICORE-Client ohne Probleme mit der Integrationsimplementierung erweitern zu können.

4. Der Benutzer soll durch Rückmeldungen über fehlerhafte Benutzer-Interaktionen benachrichtigt werden

Punkt 1 beschreibt, dass das erweiterte UNICORE-System aus Sicht des Benutzers in der Bedienung möglichst wenige Veränderungen aufweisen soll, damit er über dieselben Standard-UNICORE-Schritte Jobs submittieren kann.
Die Erweiterung der Benutzeridentität um VO-Credentials, die in *Punkt 2* genannt wurde, soll mit wenigen Benutzer-Schritten gewährleistet werden. Diese Erweiterung soll für den Benutzer einfach und verständlich gestaltet sein und ihn bei seiner Interaktion unterstützen.
In *Punkt 3* wird die Anforderung erörtert, eine möglichst problemlose Einfügung in bereits bestehende UNICORE-Systeme zu gewährleisten.
Feedback-Benachrichtigungen, in *Punkt 4* aufgeführt, sind von besonderer Wichtigkeit, damit der Benutzer über fehlende oder fehlerhafte Interaktionen benachrichtigt wird.

4.1.3 Anforderungen aus Administrations-Sicht

Aus Sicht des UNICORE-Administrators sollen folgende Punkte in der Integrations-realisierung berücksichtigt werden:

1. Die neue Autorisierungsimplementierung soll möglichst einfach in ein bestehendes UNICORE-System eingefügt werden können.

2. Der Administrator kann zwischen der identitätsbasierten und der attributbasierten Autorisierungsform wählen.

3. VO-basierte Systemzustände sollen protokolliert werden.

4. Die Realisierung der Autorisierungsentscheidung soll erweiterbar sein.

Punkt 1 ist ein wichtiges Kriterium dafür, dass Communities, die bereits UNICORE verwenden, möglichst einfach deren UNICORE-System um die Integrationslösung erweitern können. Eine Neu-Installation des UNICORE-Systems soll nicht notwendig sein.

Durch *Punkt 2* wird deutlich, dass Mechanismen gefunden werden müssen, die es dem Administrator erlauben, zwischen den verschiedenen Autorisierungsformen wählen zu können.

Punkt 3 beschreibt, dass die Systemzustände, die durch die Integrationslösungen entstehen, auch protokolliert werden müssen. Falls Fehler während der UNICORE-Anwendung auftauchen, soll der Administrator hierdurch in der Lage sein, den Fehlerraum einzugrenzen und den Fehler zu erkennen.

Da jede Community unterschiedliche Policies besitzt, müssen die Mechanismen, welche über die Autorisierung entscheiden, vom Administrator erweiterbar und gut zu verwalten sein (*Punkt 4*).

Eine Schwierigkeit bei der Realisierung dieser Anforderungen ist, dass diese in negativer Korrelation zueinander stehen können. Die Realisierung einer attributbasierten Autorisierung (Punkt 2 aus der allgemeinen Integrationsarchitektur) muss die AAI von UNICORE verändern. Die UNICORE Authentifizierung hingegen soll gemäß Punkt 3 (allgemeine Integrationsarchitektur) nicht verändert werden. Es müssen Konzepte gefunden werden, die diese Anforderungen soweit wie möglich erfüllen.

4.2 Kreativitätsraum der UNICORE-Integration

Die Konzeption für die UNICORE-Erweiterung bietet keinen großen Spielraum für Kreativität und ist durch einen Rahmen begrenzt. Folgende Punkte verdeutlichen dies:

- die Integration von zwei Systemen,
- die zur Verfügung stehenden Schnittstellen,
- das Berücksichtigen der Anforderungen.

Die Integration von Systemen zeichnet sich dadurch aus, dass sie aus bereits vorhandenen Architekturen, Sprachen und Infrastrukturen besteht. Daher kann die Konzeption nicht „auf der grünen Wiese“ begonnen werden, sondern man muss UNICORE um neue VO-Funktionen erweitern.

Der UNICORE-Client und die Server-Komponenten sind in der Programmiersprache JAVA implementiert. Für das Abbilden der AAI des VO-Managements in UNICORE müssen folgende Fragestellungen berücksichtigt werden:

- Welche Schnittstellen bietet das UNICORE-System an?
- Welche UNICORE Komponenten müssen verändert werden?
- Welche Benutzerinformationen werden vom Client zu den Server-Komponenten übertragen?
- Wie können die VO-Credentials der Benutzer in UNIOCRE repräsentiert werden?
- Wie gelangen die Attribute der Benutzer zum Autorisierungsdienst UUDB?
- wie soll eine attributbasierte Autorisierung in der UUDB erfolgen?

5. Konzeption

In diesem Kapitel wird beschreiben, wie die AAI von UNICORE um VO-Autorisierungsmechanismen erweitert werden muss. Es lassen sich zwei Fragestellungen für die Konzeption formulieren:

- Wie muss der UNICORE-Authentifizierungsmechanismus erweitert werden?

- Wie kann gewährleistet werden, dass die Autorisierung in UNICORE anhand von VO-Attributen realisiert wird?

Daher werden die Bereiche der Authentifizierung und Autorisierung getrennt voneinander behandelt. Zu dem jeweiligen Bereich werden die zunächst relevanten VO-Konzepte vorgestellt. Auch wird beschrieben, wie man diese Konzepte in eine UNICORE-Umgebung integrieren kann. Anschließend geht das Kapitel auf die konkreten Integrations-Konzepte von UNICORE und VOMS sowie UNICORE und Shibboleth ein. Zum Schluss werden diese Integrationskonzepte wieder verallgemeinert, damit eine Interoperabilität zu anderen VO-Technologien gegeben ist und um eine flexible Erweiterung anzubieten.

5.1 Allgemeine Integrationsarchitektur

Wie in Abschnitt 2.6.1 beschrieben, besteht UNICORE aus verschiedenen Systemkomponenten. Um die AAI von UNICORE um VO-Mechanismen zu erweitern, stellt sich zunächst die Frage, welche UNICORE-Komponente für die Authentifizierung und welche für die Autorisierung zuständig ist. Desweiteren muss überprüft werden, welche Implementierungsmöglichkeiten UNICORE für diese Komponenten bietet, die die genannten Anforderungen aus Abschnitt 4.1 nicht verletzen.

Die *Authentifizierung* des Benutzers geschieht in der UNICORE-Client-Komponente. Der UNICORE-Client bietet durch ein Client-Plugin-Interface die Möglichkeit, das UNICORE-System um zusätzliche Funktionen zu erweitern, ohne dabei die Basis-Software zu verändern.
Der *Autorisierungsmechanismus* erfolgt in der NJS-Komponente. NJS autorisiert eine Benutzeridentität mit Hilfe der UNICORE User Database (UUDB). NJS bietet die Möglichkeit, eine eigene UUDB-Implementierung in das UNICORE-System zu integrieren.

Eine allgemeine Integrationsarchitektur lässt sich dadurch wie folgt beschreiben:

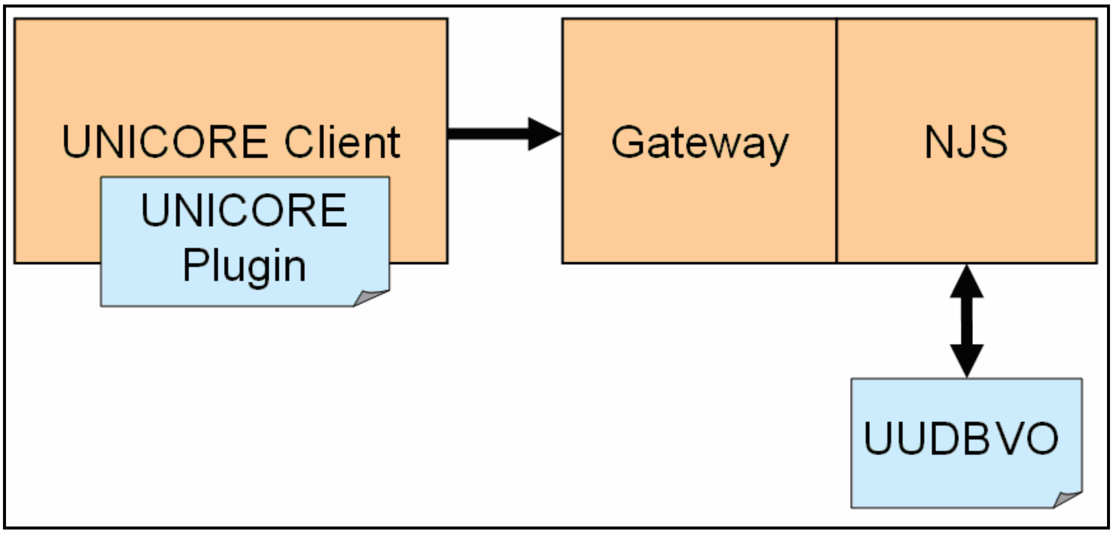

Abb.24: Konzeptionsmöglichkeit der Integration

- Der UNICORE-Client wird um ein VO-Plugin erweitert, das die Aufgabe hat, einen VO-Benutzer eindeutig in der UNICORE-Umgebung zu identifizieren.

- Damit der Benutzer durch VO-Attribute autorisiert wird, muss eine neue UUDB implementiert werden (UUDBVO).

5.2 Konzeption der VO-Authentifizierung

Dieser Abschnitt beschreibt zunächst die relevanten Begriffe, die für die Konzeption der VO-Authentifizierung benötigt werden. UNICORE muss einen Mechanismus anbieten, der die VO-Benutzer eindeutig identifizieren kann. Deswegen werden erst die benötigten Schritte beschrieben, die der Benutzer ausführen muss, um Zugang zum Standard-UNICORE-System zu erlangen. Anschließend werden Konzepte für die Integration von verschiedenen VO-Ansätzen in UNICORE vorgestellt.

5.2.1 Begrifflichkeiten der Authentisierung

Die Unterschiede zwischen den Benutzer- sowie VO-Credentials werden hier herausgearbeitet, da diese für die Konzeption von besonderer Bedeutung sind. Der Begriff *VO-Credentials-Service* wurde definiert, um das allgemeine Prinzip der Credential-Erstellung zu beschreiben.

Benutzer-Credentials

Der Benutzer benötigt *Credentials* (Berechtigungsnachweise), damit so seine angegebene Identität für die Authentifizierung überprüft werden kann. Zum Beispiel können Benutzer-Credentials die Kombination von User-ID und Passwort sein. Diese können als *Identitäts-Credentials* bezeichnet werden. In den meisten Grid-Umgebungen kommen X.509-Zertifikate als Identitäts-Credentials zum Einsatz. Benutzerzertifikate haben gegenüber den Userlogin/Password-Credentials den Vorteil, dass die Authentifizierung nicht nur über Wissen, sondern auch über Besitz realisiert wird. Nur der Benutzer ist im Besitz des privaten Schlüssels, der zusätzlich auch passphrase-geschützt ist. Dadurch verringert sich die Möglichkeit der Kompromittierung. Die digitale Identität eines Benutzers mit seinen Merkmalen (wie zum Beispiel Vor- und Nachname, Email-Adresse, Anschrift, Organisation) ändert sich kaum, daher haben Benutzer-Credentials eine lange Gültigkeit. Benutzerzertifikate zum Beispiel haben eine Gültigkeitsdauer von mehreren Jahren.

VO-Credentials

VO-Credentials fassen im Allgemeinen die VO-Attribute eines Benutzers zusammen. Sie müssen die Anforderungen an eine VO-Umgebung erfüllen, die durch die Eigenschaften von VO-Attributen gegeben sind. Deswegen werden zunächst die Eigenschaften der VO-Attribute aufgelistet und erläutert:

1. VO-Attribute repräsentieren Autorisierungsinformationen
2. Sie haben eine dynamische Struktur
3. Sie müssen dem Besitzer eindeutig zugeordnet werden können
4. Sie müssen jederzeit ersetzbar sein

Punkt 1 beschreibt, dass VO-Attribute für die Autorisierung angewendet werden sollen. Anhand der Autorisierungs-Informationen soll das VO-System Berechtigungsentscheidungen treffen können. Dadurch soll ABAC sowie RBAC (siehe Abschnitt 2.4.3) ermöglicht werden. VOMS beschreibt VO-Attribute in einer 3-Tupel-Form mit VO-Gruppen- und Rolleninformationen. Shibboleth dagegen kann Benutzer-Attribute z.B. durch das eduPerson[15]-Schema beschreiben.

Die Eigenschaft, die *Punkt 2* erläutert, entsteht aus den Charakteristiken der VO. Eine VO befindet sich in ständiger Umstrukturierung. Rechte von VO-Mitgliedern können sich im Laufe der Zeit ändern.

Punkt 3 ist von besonderer Wichtigkeit, da zum einen der Besitzer sicherstellen muss, dass es seine Attribute sind und zum anderen müssen die Grid-Entitäten und Komponenten sicherstellen, dass die VO-Attribute tatsächlich dem Besitzer gehören.

Punkt 4 resultiert aus der VO-Eigenschaft, dass sich VO-Attribute ad hoc ändern können. Wie in Abschnitt 2.2 beschrieben, werden die VOs manuell verwaltet und deswegen ist die Erfüllung dieser impliziten Anforderung nicht immer realisierbar. Grundsätzlich sollen Mechanismen eingesetzt werden, die diese Eigenschaft aufweisen.

Aus diesen Eigenschaften entstehen folgende Anforderungen an VO-Credentials:

- VO-Credentials werden ausschließlich für die Autorisierung verwendet.
- VO-Credentials müssen von einer vertrauenswürdigen Instanz signiert werden. So wird gewährleistet, dass sowohl der Benutzer als auch die Grid-Entitäten/Grid-Komponenten den VO-Attributen des Benutzers vertrauen können. Außerdem lassen sich die VO-Attribute dem Benutzer eindeutig zuordnen.

[15] eduPerson ist ein Objektklassen-Schema für Verzeichnisdienste. Es besteht aus einem Satz von Attributen für Personen im wissenschaftlichen oder universitären Umfeld.

- VO-Credentials müssen eine kurze Gültigkeitsdauer haben. Diese Anforderung resultiert aus den Eigenschaften, die in den Punkten 2 und 4 beschrieben wurden.

- VO-Credentials müssen jederzeit erneuert werden können.

VO-Credential-Service

Aus der zweiten Anforderung an die VO-Credentials (VO-CS) wird ersichtlich, dass eine VO-Instanz benötigt wird. Dem Benutzer muss ein VO-Credential-Service angeboten werden, der die VO-Attribute des Benutzers permanent speichert und auf Wunsch ein VO-Credential für den Benutzer erstellen kann. Durch die Zwei-Wege-Authentisierung wird gewährleistet, dass sich der Benutzer sowie der VO-Credential-Service gegenseitig vertrauen. Der Benutzer verwendet dafür seinen Benutzer-Credential. Nach der Authentifizierung kann er ein VO-Credential beim hierfür verantwortlichen Service beantragen:

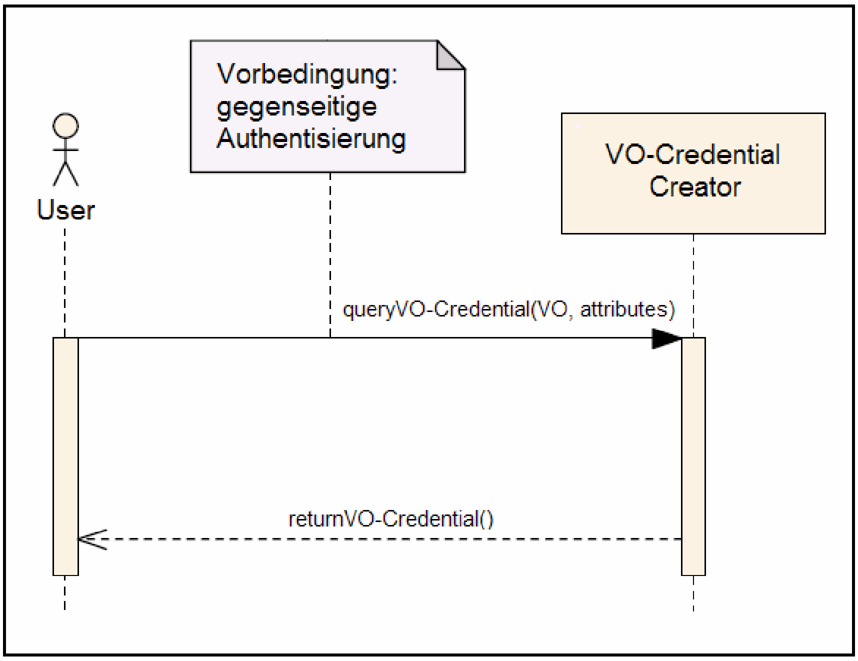

Diagramm 1: Sequenzdiagramm für VO-Credential-Erstellung

Das obige Sequenzdiagramm veranschaulicht das allgemeine Prinzip, nach dem der Benutzer sein VO-Credential erhalten kann. Prinzipiell muss die Anfrage für das Erstellen eines VO-Credentials nicht vom Benutzer kommen, sondern kann auch von einer Grid-Komponente bzw. einem Service gestellt werden, welche bei Bedarf nach VO-Attributen fragen. Wie in Abschnitt 2.2 beschrieben, kann der Benutzer Mitglied mehrerer VOs sein und unterschiedliche Attribute besitzen. Deswegen benötigt der VO-Credential-Service Informationen über die gewünschte VO und die Attribute.

Das untere Use-Case-Diagramm illustriert unter anderem die beschriebene Anforderung des permanenten Speicherns und das Verwalten der VO-Attribute durch einen VO-Administrator:

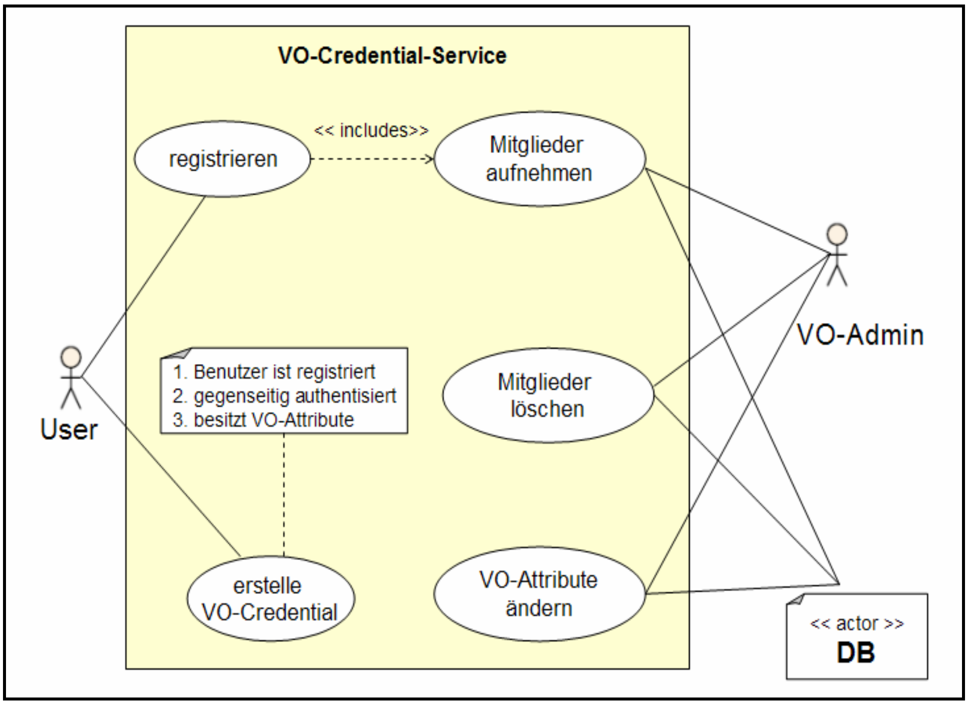

Diagramm 2: Use Case-Diagramm für VO-Credential-Service

Ein VO-Credential-Service benötigt eine Datenbank, um die VO-Attribute sowie die VO-Mitglieder persistent zu speichern. Der VO-Administrator verwaltet VO-Mitglieder und ändert bei Bedarf ihre VO-Attribute. Der Benutzer muss im Allgemeinen zwei Schritte durchführen: Zunächst muss er sich gegenüber dem VO-Credential-Service registrieren. Anschließend kann er sein VO-Credential anfordern. Voraussetzung hierfür ist, dass er bereits registriert ist, sich beide Parteien gegenseitig authentisiert haben und dass er VO-Attribute besitzt. Nachdem für den Benutzer ein VO-Credential erstellt wurde, kann dieses Credential verwendet werden, um ihn bei anderen Grid-Systemen als *VO-Benutzer* zu kennzeichnen. Wie in den Anforderungen beschrieben, muss der VO-Credential-Service auch das VO-Credential signieren und dem Credential eine kurze Lebensdauer geben.

Es ist auch vorstellbar, dass ein VO-Credential-Service mit mehreren anderen VO-Credential-Services kommuniziert. Dies ist nötig, wenn der VO-Benutzer verschiedene VO-Attribute in unterschiedlichen Credentialformen hat. In diesem Fall wird ein übergeordneter VO-Credential-Service benötigt, der die verschiedenen Credentials in einem VO-Credential zusammenfasst. Das untere Diagramm veranschaulicht dieses Prinzip:

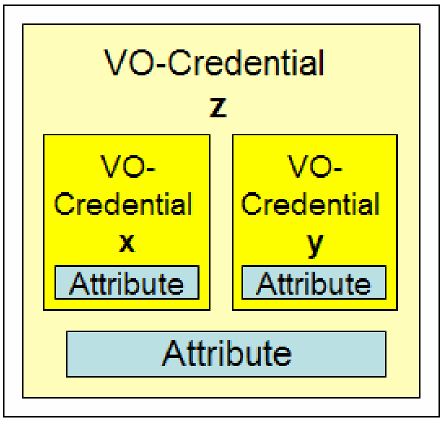

Diagramm 3: zusammengefasste VO-Credentials

5.2.2 Benötigte Benutzer-Schritte für den UNICORE-Zugang

Das UNICORE-System verwendet ein Sicherheitsmodell, das auf X.509-Zertifikaten beruht. Jeder Benutzer und jede System-Komponente werden mittels Zertifikaten eindeutig identifiziert und authentisiert.
Die X.509-Zertifikate haben in UNIOCRE folgende Aufgaben:

- Authentisierung des Benutzers
- Authentifizierung des Gateways
- Authentifizierung des NJS
- Signieren von Jobs
- Signieren von Software

Zwei verschiedene Zertifikatsarten werden benötigt:

- Benutzerzertifikate (damit an einer Usite die Identität des Benutzers überprüft werden kann)
- Server-Zertifikate
    - für die Identifizierung des Gateways
    - für die Identifizierung des NJS
    - für die Identifizierung des TSIs

Damit der Benutzer mit seinem Zertifikat UNICORE nutzen kann, muss er folgende Schritte durchführen:

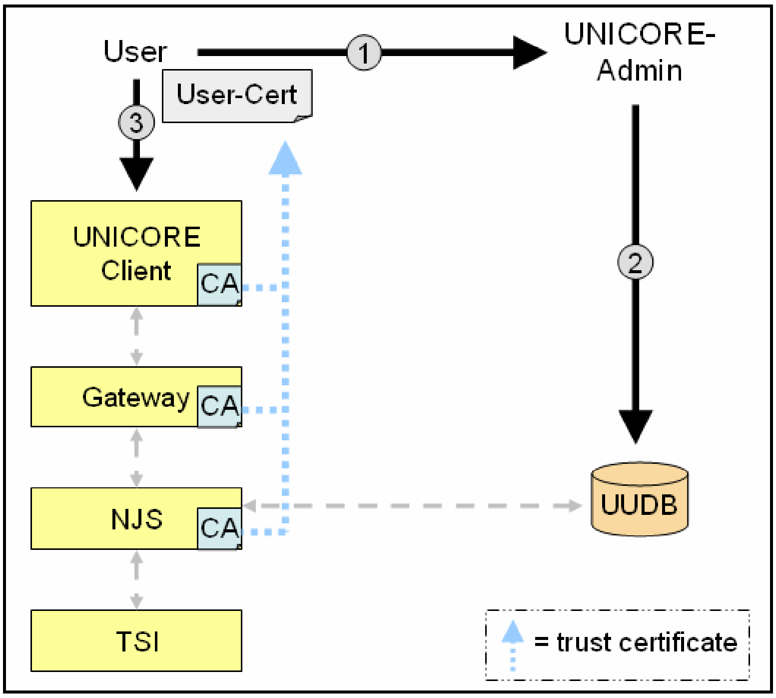

Diagramm 4: UNICORE 5-Zugangsschritte für den Benutzer

Grundvorausetzung für diese Schritte ist, dass die UNICORE-Komponenten Client, Gateway und NJS der Certificate Authority, die das Benutzerzertifikat signiert hat, vertrauen. Falls dies noch nicht der Fall ist, muss die CA als vertrauenswürdige Instanz den Komponenten bekannt gemacht werden.

1. Der Benutzer sendet sein Benutzerzertifikat dem UNICORE-Administrator zu. Dies geschieht in der Regel über E-Mail.

2. Der UNICORE-Administrator fügt das Benutzerzertifikat in die UUDB ein.

3. Der Benutzer konvertiert sein Benutzerzertifikat mit seinem privaten Schlüssel als *P12-keystore* und fügt es in den UNICORE-Client-Keystore ein. Das CA-Zertifikat, das sein Benutzerzertifikat signiert hat, muss im UNICORE-Client-Truststore hinzugefügt werden.

5.2.3 Integrationskonzepte für die Authentisierung

Im Allgemeinen lässt sich festhalten, dass der Benutzer Zugang zu UNICORE sowie zu seinem *VO-Credential-Service* haben muss. Der Benutzer muss sich bei beiden

Systemen mit seinem Benutzer-Credential authentifizieren. Das Sicherheitsmodell von UNICORE basiert auf X.509-Zertifikaten, d.h. UNICORE setzt voraus, dass es sich bei dem Benutzer-Credential um ein Zertifikat handelt. Dies muss jedoch nicht der Fall bei einem Benutzer-Credential innerhalb einer VO-Umgebung sein. Deswegen unterscheiden wir zwei Credential-Formen voneinander:

Das Benutzer-Credential des Benutzers ist

- ein X.509-Benutzerzertifikat
- kein X.509-Zertifikat

Die erste oben beschriebene Form stellt die einfachste Integrationsform dar, die im folgenden Diagramm veranschaulicht wird:

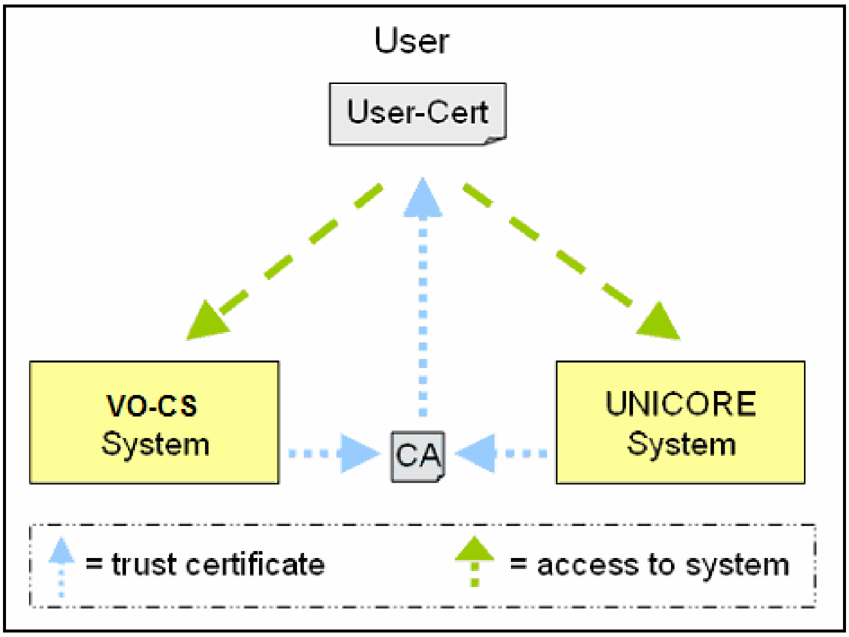

Diagramm 5: Benutzer-Zugang zu UNICORE & VO-CS durch ein BZ

Der Benutzer verwendet sein BZ, um Zugang zu beiden Systemen zu erlangen. Zusätzlich müssen beide Systeme der CA des Benutzers vertrauen. UNICORE muss das CA-Zertifikat des VO-CS als vertrauenswürdige Instanz einfügen, um die VO-Attribute des Benutzers verifizieren zu können.

Zusammengefasst benötigt UNICORE für die Integration mit dem VO-CS folgende Zertifikate:

- Benutzerzertifikat

- CA des Benutzerzertifikats
- CA-Zertifikat bzw. Server-Zertifkat des VO-CS

Wenn das Benutzer-Credential nicht in Form eines Zertifikats vorliegt, wird folgende Integration benötigt:

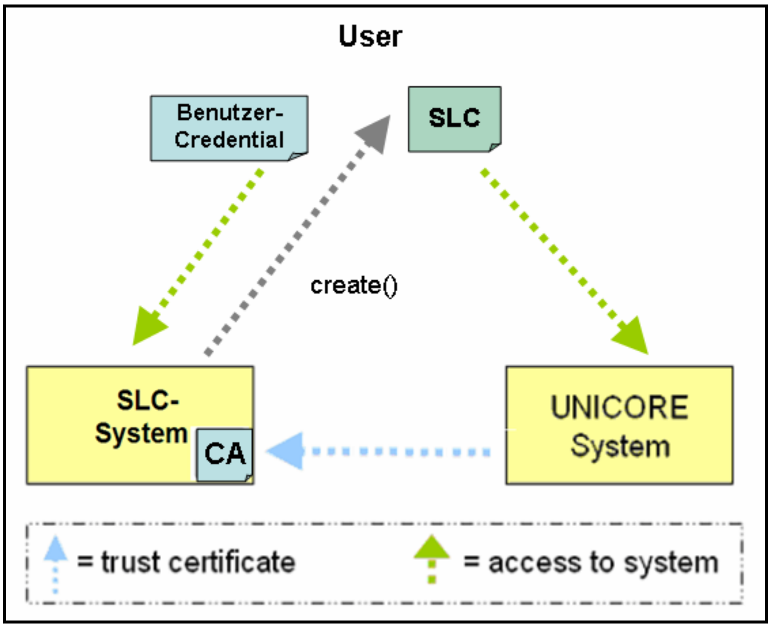

Diagramm 6: Benutzer-Zugang zu UNICORE & VO-CS durch SLC

Ein *Short-Lived-Credential-Service,* der das Benutzer-Credential auf ein X.509-Zertifikat mit kurzer Lebensdauer abbildet, muss hier zum Einsatz kommen. SLCS kann gleichzeitig auch als ein VO-Credential-Service fungieren, der in den Extensions des SLC die VO-Attribute bzw. die VO-Credentials einfügt. Dies ist aber im Allgemeinen nicht die Aufgabe eines SLCS. Damit für den Benutzer trotzdem VO-Credentials erstellt werden, muss dieser anschließend mit einem VO-Credential-Service kommunizieren.

Damit UNICORE den Benutzer anhand der VO-Attribute autorisiert, muss die Identität des Benutzers im UNICORE-System um VO-Credentials erweitert werden. Daraus folgt, dass der Benutzer erst das VO-System ausführen muss, bevor er UNICORE startet.
Die folgende Abbildung veranschaulicht das Prinzip:

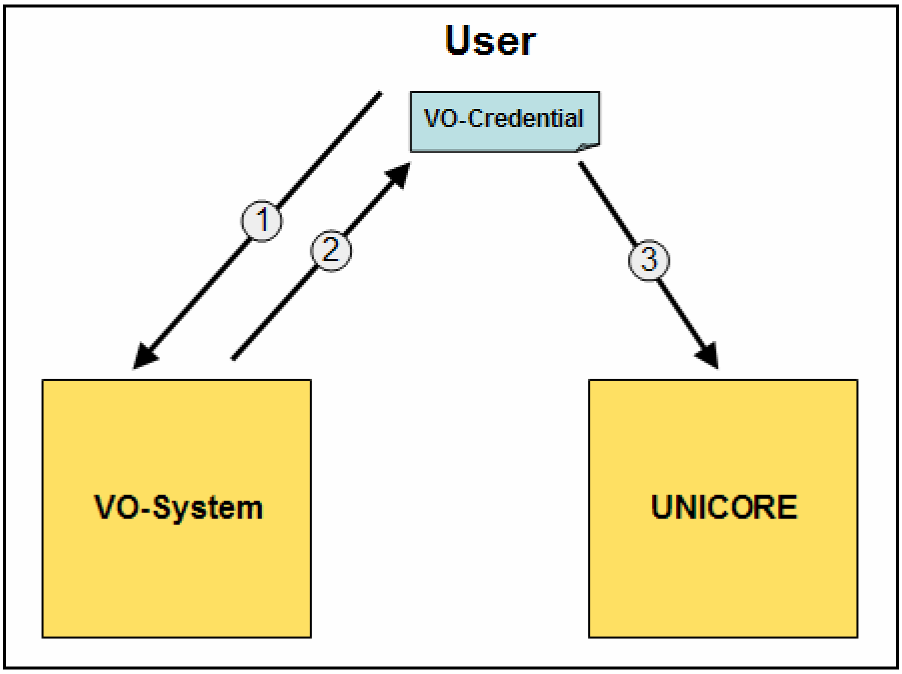

Diagramm 7: Nutzungs-Reihenfolge vom VO- sowie UNICORE-System

1. Der Benutzer authentifiziert sich gegenüber dem VO-System und sendet seinen VO-Request an das VO-System.

2. Das VO-System überprüft die Syntax des Requests. Anschließend erstellt es für den Benutzer ein VO-Credential und sendet es an den Benutzer weiter.

3. Der Benutzer führt das UNICORE-System aus und erweitert seine Benutzer-Identität mit seinem VO-Credential.

5.3 Konzeption der VO-Autorisierung

Der Abschnitt benennt zuerst die relevanten Parteien und Objekte der VO-Autorisierung. Später beschreibt es die Strategien und Richtlinien einer Autorisierung und geht zum Schluss auf die Konzeption einer VO-basierten UUDB ein.

5.3.1 Parteien und Objekte einer VO-Autorisierung

Folgende Parteien und Objekte spielen für die VO-Autorisierung eine wesentliche Rolle:

- **VO-Benutzer:**
  Der VO-Benutzer ist Mitglied einer Virtuellen Organisation und möchte Zugang zu VO-Ressourcen erhalten.

- **VO-Attribute:**
  Die VO-Attribute geben Auskunft über die Rechte und Rollen eines Benutzers.

- **VO-Credentials:**
  VO Credentials fassen die VO-Attribute des Benutzers zusammen.

- **VO-Ressourcen:**
  VO-Ressourcen entsprechen Ressourcen, die aber nur von Mitgliedern einer VO genutzt werden können.

- **VO-Service Provider:**
  Der VO-Service Provider ist zuständig für das Anbieten von VO-Ressourcen.

- **VO-Autorisierungsdienst:**
  Der VO-Autorisierungsdienst gewährt oder verweigert dem VO-Benutzer anhand seiner VO-Attribute den Zugang zu einer VO-Ressource.

- **VO-Richtlinien:**
  VO-Richtlinien legen fest, welche Ressourcen geteilt werden, wem die gemeinsame Nutzung erlaubt ist und unter welchen Konditionen die gemeinsame Nutzung erfolgt.

5.3.2 Autorisierung durch VO-Credentials

Für die Autorisierung eines VO-Benutzers werden seine Benutzerattribute hinzugezogen. Diagramm 8 beschreibt allgemein die Autorisierung durch die Attribute des Benutzers. Bei einer erfolgreichen Autorisierung wird einem VO-Benutzer der Zugang zu einer VO-Ressource gewährt. Bei einer nicht erfolgreichen Autorisierung wird der Zugang verweigert. Im konkreten Fall wird bei einer erfolgreichen Autorisierung nicht nur der Zugang des VO Benutzers geregelt, sondern auch festgelegt unter welchen Bedingungen der Zugang bzw. die Nutzung einer VO-Ressource erlaubt ist.

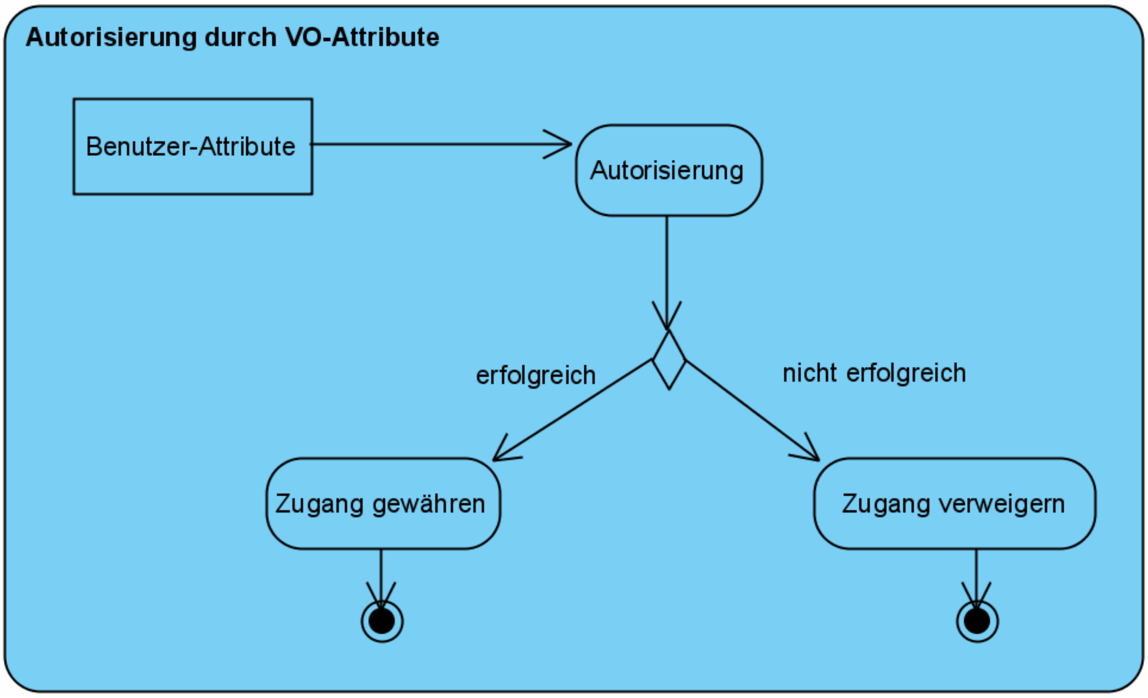

Diagramm 8: Aktivitätsdiagramm für eine VO-Autorisierung

Die Benutzerattribute können in zwei verschiedene Bereiche untergliedert werden:

- **Campusattribute:**
  Campusattribute sind Benutzerattribute, die bei ihren Heimatorganisationen verwaltet werden. Die Attribute identifizieren und beschreiben einen Benutzer, z.B. durch seinem Namen oder seine Telefonnummer. Zusätzlich werden Attribute für die Zugehörigkeit und die Rolle des Benutzers in seinem Unternehmen, Institut oder seiner Universität verwaltet.

- **VO-Attribute:**
  VO-Attribute beschreiben die Mitgliedschaft, Rollen und die Rechte der Benutzer in einer VO.

Für eine Autorisierungsentscheidung können nicht nur die VO-Attribute, sondern auch die Campusattribute eine wichtige Rolle spielen. So kann zum Beispiel auch das Verweigern gewisser Campusattribute (wie das Attribut der Nationalität) für die Zugangsentscheidung relevant sein.
Der Unterschied zwischen VO- und Campusattributen ist auch, dass der Benutzer nur eine VO-Attributfolge, aber mehrere Campusattribute besitzen kann. In der VO-Attributfolge (FQAN) wird die VO, die Rolle und die Fähigkeiten des Benutzers beschrieben. Campusattribute können unabhängig von anderen betrachtet und für die Autorisierung hinzugezogen werden.

Ein Autorisierungsdienst hat, abgesehen vom Treffen der Zugangsentscheidung, noch folgende Aufgaben:

- Die Validierung und Verifizierung der Attribute.
- Das Extrahieren der Attribute aus dem VO-Credential.
- Das Auffinden von VO-Richtlinien.
- Das Vergleichen der Attribute mit den VO-Richtlinien.

Die Validierung und Verifizierung der VO-Credentials des VO-Benutzers ist notwendig, um sicherstellen zu können, dass es sich tatsächlich um gültige VO-Credentials eines VO-Benutzers handelt. Das Extrahieren der VO-Attribute aus dem VO-Credential wird benötigt, um auch einzelne Attribute für die Autorisierungsentscheidung zu berücksichtigen. Für die korrekte Autorisierungsentscheidung ist das vollständige Auffinden von VO-Richtlinien erforderlich. Die VO-Richtlinien wiederum müssen für die attributbasierte Zugangsentscheidung mitberücksichtigt werden.

5.3.3 Kommunikationsstrategien für die verteilte Autorisierung

Um eine VO bilden zu können, die eine weltweit verteilte Ressourcen-Gemeinschaft darstellt, bedarf es einer verteilten Autorisierung. Für eine verteilte Autorisierung der Partner kommen drei Kommunikationsstrategien in Betracht:

- **Agent Sequence:**
  Die Strategie besagt, dass der VO-Benutzer dem Autorisierungsdienst eine Anfrage an eine VO-Ressource stellt. Der Autorisierungsdienst überprüft die Berechtigung anhand der Attribute des Benutzers. Bei einer erfolgreichen Autorisierung dient er als Agent und stellt dem Benutzer die angeforderte VO Ressource bereit.

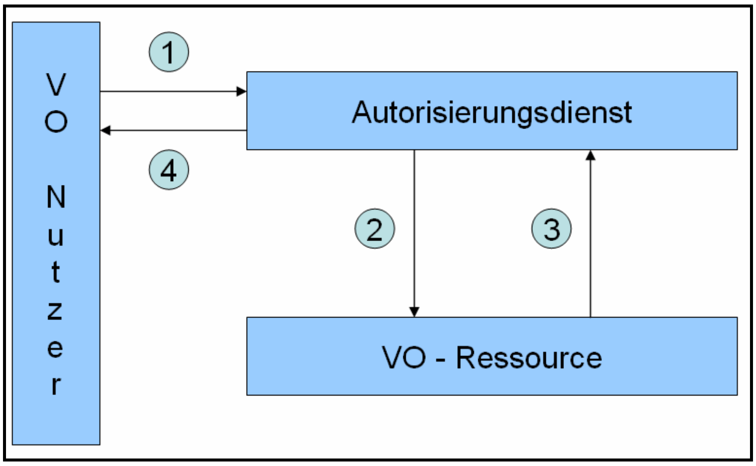

Diagramm 9: Agent Sequence

- **Pull Sequence:**
  Der VO-Benutzer versucht bei dieser Strategie, direkt Zugang zu einer VO-Ressource zu erhalten. Der Zugang wird aber nur bei Sicherstellung einer Benutzer-Autorisierung gewährt. Deswegen wird für diesen Zweck ein Autorisierungsdienst hinzugezogen, der die Berechtigung des Benutzers überprüft. Bei einer erfolgreichen Autorisierung erhält der Benutzer Zugang zur Ressource.

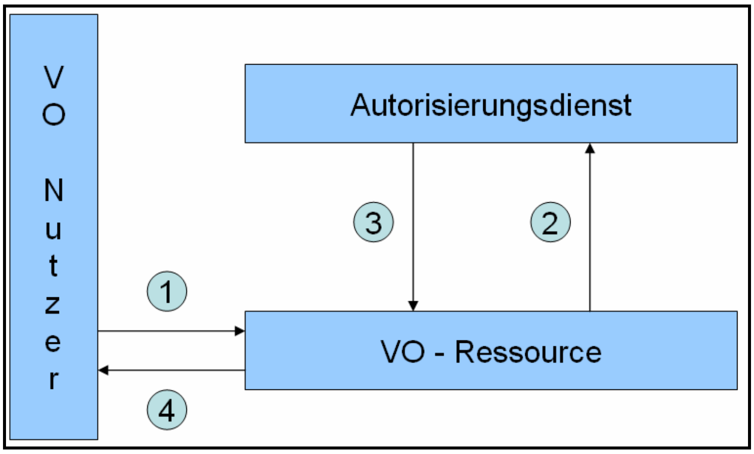

Diagramm 10: Pull Sequence

- **Push Sequence:**
  Bei dieser Strategie muss der VO-Benutzer zunächst einen Nachweis seiner Berechtigung über einen Autorisierungsdienst beschaffen. Diesen Nachweis schickt er zu einer VO-Ressource. Anhand des Nachweises wird überprüft, ob der Benutzer Zugang zu der Ressource erhält.

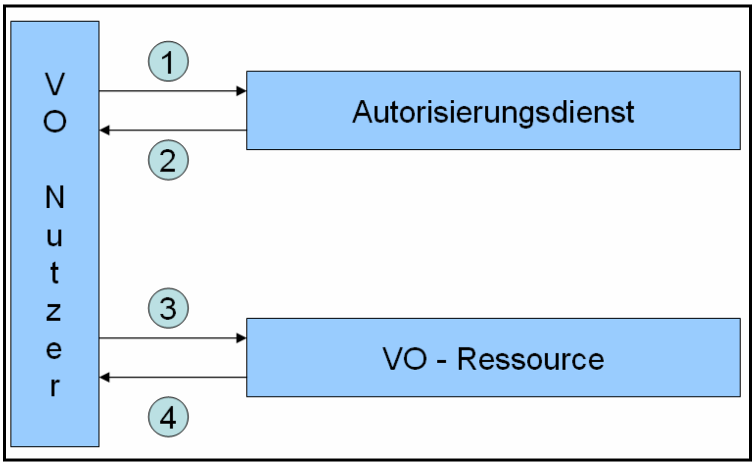

Diagramm 11: Push Sequence

5.3.4 Pull- oder Push-Ansatz der VO-Credentials

Ferner muss überprüft werden, ob die VO-Credentials des VO-Benutzers per Push- oder Pull-Ansatz dem VO-Service Provider übermittelt werden sollen. Bei beiden Ansätzen sendet der VO-Benutzer dem VO-Service Provider einen Nachweis seiner erfolgreichen Authentifizierung zu. Der Nachweis einer erfolgreichen Authentifizierung ist die notwendige Vorraussetzung für das Autorisieren des Benutzers. Die Frage, die sich stellt, ist, ob in dem Authentifizierungsnachweis bereits VO-Attribute vorhanden sind (Attribut-Push). Ist dies nicht der Fall, müssen Funktionen bei einem VO-Service

Provider bereitgestellt werden, die die VO-Attribute des VO-Benutzers abfragen können (Attribut-Pull). Abbildung 25 stellt die beiden Ansätze gegenüber.

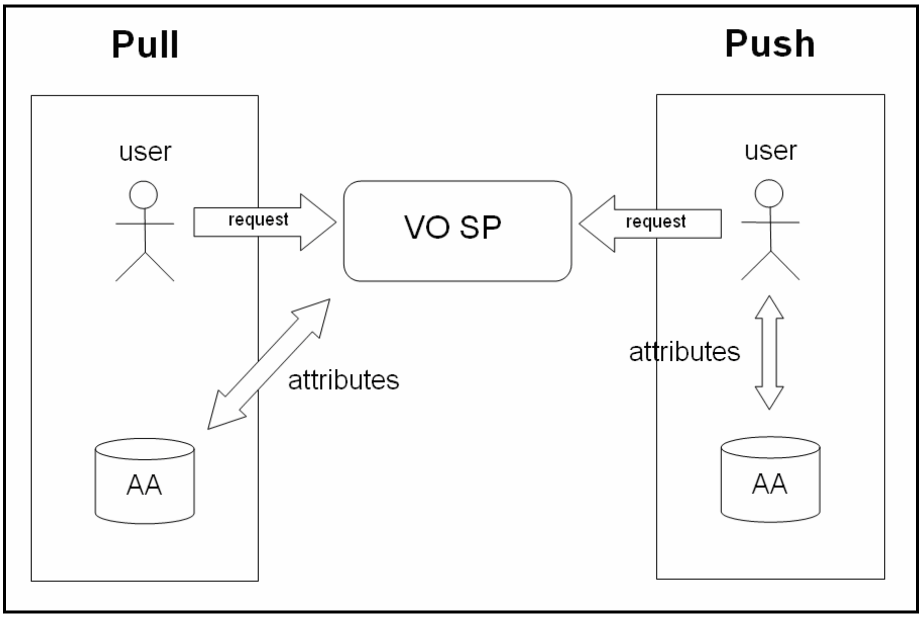

Abb.25: Attribute-Pull vs. Attribute-Push

Shibboleth benutzt einen Attribut-Pull-Ansatz (siehe Abschnitt 3.4.1). Dagegen basiert das VOMS-System auf dem Attribut-Push-Ansatz.

5.3.5 Richtlinien für die verteilte Autorisierung

Die Richtlinien einer VO haben für die Konzeption der Autorisierung eine wesentliche Bedeutung. Die VO-Richtlinien müssen bei einer Autorisierung abgefragt und durchgesetzt werden. Innerhalb der Autorisierungsinfrastruktur lassen sich vier Punkte identifizieren [GRI06]:

- **Policy Retrieval Point:**
  Der *Policy Retrieval Point (PRP)* ermöglicht den Zugriff auf Richtlinien. Zum Beispiel wird ein Datenbank-Zugang zu den Richtlinien zur Verfügung gestellt.

- **Policy Information Point:**
  Der *Policy Information Point (PIP)* ermittelt die für die VO in Frage kommenden Richtlinien.

- **Policy Decision Point:**
  Der *Policy Decision Point (PDP)* trifft unter Berücksichtigung der Richtlinien eine Entscheidung über die Autorisierung des VO-Benutzers.

- **Policy Enforcement Point:**
  Durch den *Policy Enforcement Point (PEP)* wird sichergestellt, dass die Richtlinien durchgesetzt werden.

5.3.6 Die identitätsbasierte UUDB

Die UUDB ist in Java sowie in Shellskripten implementiert worden. Die Java-Realisierung beinhaltet den Kern der UUDB. Die Shellskripte unterstützen das Administrieren der Benutzerlisten.

- **UUDB (Java):**
  Die UUDB wurde modular innerhalb des NJS implementiert. Diese Modularität erlaubt es, die UUDB zu erweitern oder gar neu zu implementieren ohne die Komponente des NJS zu verändern. Die neu implementierte UUDB könnte beispielsweise als *Jar-Format* in den NJS-*Libraries* hinzugefügt werden. Die Konfigurationsumgebung des NJS bietet eine Möglichkeit, die neu implementierte UUDB als Basisklasse anzugeben.

- **UUDB (Shellskripte):**
  Für die Administration der UUDB wurden Shellskripte bereitgestellt. Dadurch ist das Einfügen, Löschen und Auflisten der UNICORE-Benutzer möglich.

Die UUDB wird durch eine *HashMap* realisiert. Die HashMap besteht aus einem Tupel (Schlüssel, Wert). Die Schlüssel der HashMap beinhalten die Zertifikate der Benutzer. Die Werte entsprechen den UNIX-Accounts. Die HashMap wird nach dem Beenden von UNICORE in einer binären Datei abgespeichert. Beim Starten wird eine HashMap initialisiert und durch die binäre Datei mit Werten gefüllt.

Die identitätsbasierte Autorisierung von UNICORE wird in Diagramm 12 dargestellt. Der NJS schickt das Benutzerzertifikat zur UUDB. Die UUDB überprüft das Benutzerzertifikat mit dem in der Hash-Map vorhandenen Zertifikaten. Falls kein identisches Zertifikat gefunden wird, ist die Autorisierung des Benutzers fehlgeschlagen. Falls die Suche erfolgreich ist, wird der UNIX-Account mit dem Hash-Map-Schlüssel entnommen. Durch die Namensstruktur des Accounts wird die Rolle des Benutzers zugewiesen. Die Rolle und der Account werden dann dem NJS zurückgesendet.

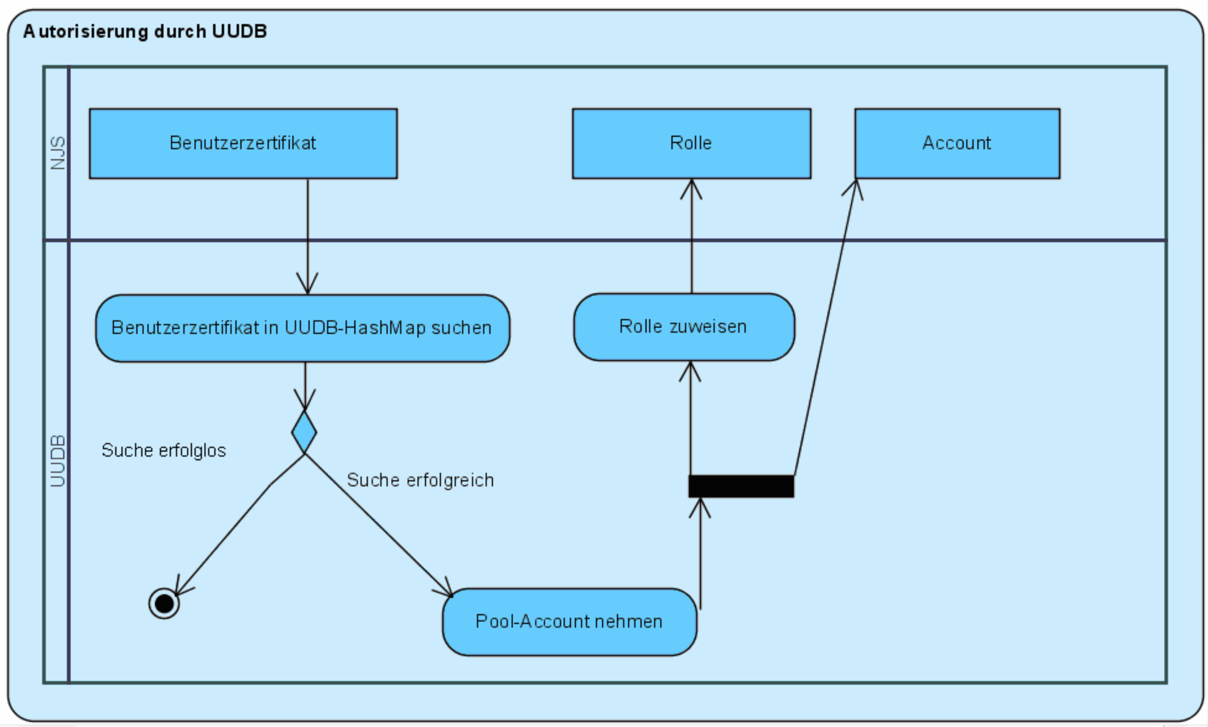

Diagramm 12: Autorisierung durch UUDB

5.3.7 Konzeption der UUDBVO

Um den Autorisierungsdienst UUDB durch VO-basierte Zugangsentscheidungen zu erweitern, müssen notwendige Veränderungen durchgeführt werden. Die neue *UUDBVO (UNICORE User Database for Virtual Organisation)* soll folgende Veränderungen gegenüber der alten UUDB beinhalten:

- Die Schlüssel der HashMap sollen nicht mehr aus Zertifikaten, sondern aus Attributen bestehen.
- Die Administration der Benutzerliste soll durch eine Attributliste ersetzt werden.
- Die UUDBVO soll die Campus- sowie die VO-Attribute der Benutzer aus dem Benutzer-Credential heraus extrahieren.
- Die UUDBVO muss eine PDP bereitstellen, die das Abbilden der UNIX-Logins anhand der Attribute ermöglicht.

Zusätzlich muss noch die Wahl einer Kommunikationsstrategie für die verteilte Autorisierung und die Wahl der Attributübermittlungsstrategie erfolgen. Darüber hinaus müssen Möglichkeiten für das Einbinden von VO-Richtlinien gefunden werden.

Attribute statt Zertifikate

Die UUDBVO soll statt dem identitätsbasierten Ansatz, welcher Benutzerzertifikate auf UNIX-Logins abbildet, einen attributbasierten Ansatz anwenden. So sollen Attribute auf UNIX-Logins bzw. Pool-Accounts gemappt werden. Für die Realisierung sollen die Schlüssel der HashMap die notwendigen Attribute für eine Autorisierung beinhalten.

Administration der Attributliste

Die Attribute, die für die Autorisierung hinzugezogen werden, sollen durch einen Administrator mit den jeweiligen UNIX-Logins in einer Attributliste verwaltet werden können. Der Administrator soll anhand der VO-Policies die Attributliste zusammenstellen können. In der Attributliste sollen Campus- und VO-Attribute durch SAML und durch VOMS verwaltet werden können. Der Administrator kann nicht nur gewünschte, sondern auch *nicht* gewünschte Attribute für die Zugangsentscheidung bezüglich einer Ressource hinzuziehen.

Extrahieren der Attribute und das Bereitstellen einer UNICORE-PDP

Das Extrahieren der Attribute soll gewährleistet werden. Es müssen Attributzertifikate und SAML-Assertions interpretiert werden können. So wären dann eine Validierung, Verifizierung und eine Extrahierung der Benutzer-Attribute möglich.
Nach dem Extrahieren der Attribute muss ein UNICORE-PDP bereitgestellt werden, die die Attribute mit der Attributliste vergleicht. Die PDP entscheidet letztendlich, ob der Benutzer anhand seiner Attribute Zugang zu einer Ressource erhalten soll.

Wahl der Kommunikationsstrategie

Für die Wahl der geeigneten Kommunikationsstrategie wurde die Autorisierung von UNICORE mitberücksichtigt. UNICORE verwendet eine Kombination aus Agent-Sequence- und der Push-Sequence-Strategie. Die Agent-Sequence-Strategie wird in UNICORE verwendet, da der Benutzer keinen direkten Zugriff auf Grid-Ressourcen hat. Die Grid-Ressourcen (wie z.B. Cluster-Systeme) in UNICORE werden hauptsächlich für das Submittieren von Jobs verwendet. In UNICORE erhält der Benutzer auf der Basis seiner Rechte eine Auswahl ihm zur Verfügung stehender Grid-Ressourcen.
Die Form der Autorisierung in UNICORE dagegen entspricht mehr der Push-Sequence-Strategie. Der UNICORE-Benutzer besitzt mit seinem BZ einen Nachweis über seine Berechtigung. Anhand dieses Nachweises wird der Benutzer autorisiert. Erst nach erfolgreicher Autorisierung hat der Benutzer die Möglichkeit Vsites auszuwählen. Die

Pull-Strategie wird in UNICORE nicht verwendet, da dem Benutzer kein direkter Zugriff auf Grid-Ressourcen möglich ist.
Weil ein Ziel der UNICORE-Implementierung darin besteht, den Kern des Systems nicht zu verändern, entspricht die Wahl der Autorisierungsstrategie, die der von UNICORE. Somit wird für das Anbieten der Ressourcen, die Agent-Sequence-, für die Benutzerautorisierung, die Agent-Push-Strategie gewählt.

Im Bereich der UNICORE-Shibboleth-Integration stellt sich noch die Frage nach der Wahl der richtigen Attributübermittlungsstrategie. Der Vorteil des Attribut-Pull-Ansatzes gegenüber dem Push-Ansatz ist, dass der Benutzer nicht seine Attribute sendet. Erst nach Anfrage des Service Providers werden die gewünschten Attribute übermittelt. Die Nachteile des Pull-Ansatzes sind, dass der Ansatz weniger performant ist (Zwei-Weg-Kommunikation), ein IdP-Discovery-Problem[16] existiert und zusätzliche Kommunikationsfunktionen für den Service Provider bereitgestellt werden müssen. Da VOMS einen Attribut-Push-Ansatz verfolgt soll dies bei der UUDBVO unterstützt werden. In Shibboleth sind beide Ansätze möglich. Die Wahl der richtigen Übermittlungsstrategie wird je nach Architektur der UNICORE-Shibboleth-Integration im Unterabschnitt 5.4 bewertet und schließlich ausgewählt.

Notwendige Aktivitäten der UUDBVO

Diagramm 13 beschreibt die Aktivitäten für eine Autorisierung der UUDBVO:

[16] Ein IdP Discovery Problem besagt, dass Probleme bei der Ermittlung des Identity Providers des Benutzers auftreten können.

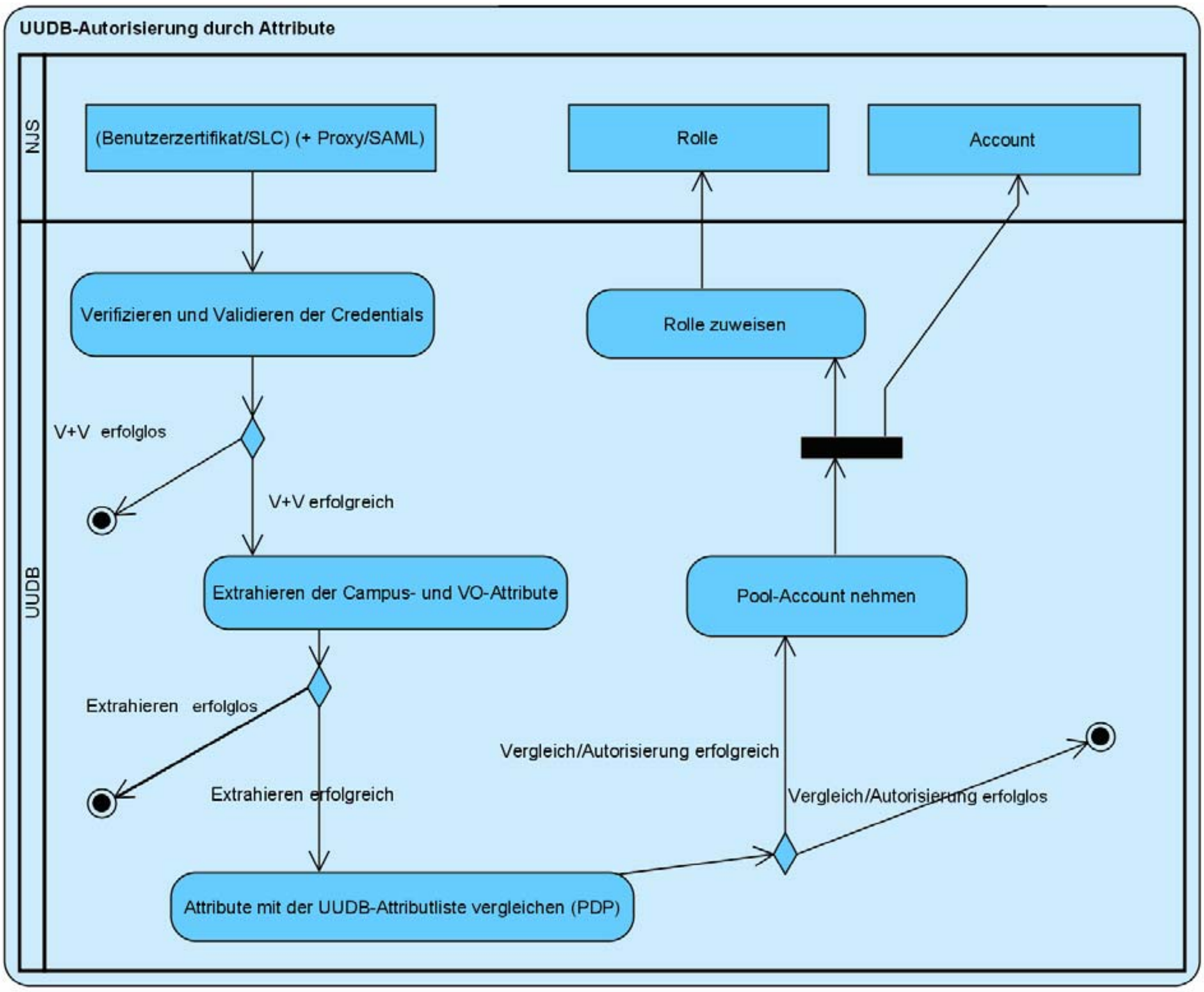

Diagramm 13: UUDB-Autorisierung durch Attribute

Der NJS leitet das Benutzerzertifikat oder das SLC an die UUDBVO weiter. Zusätzlich muss eine Möglichkeit gefunden werden, bei der das Proxy-Zertifikat sowie die SAML Assertion an die UUDBVO weitergeleitet werden. In der UUDBVO werden die Benutzercredentials sowie die VO-Credentials verifiziert und validiert. Bei einem Scheitern der Überprüfung der Gültigkeit und der Vertraulichkeit wird die Autorisierung abgebrochen. Bei einer erfolgreichen Überprüfung werden die Attribute aus der SAML-Assertion oder aus dem Attributzertifikat extrahiert. Die Attribute werden dann mit der Attributliste verglichen. Für ein erfolgreiches Mapping von Attributen zu einem UNIX-Login muss der Benutzer bestimmte Attribute besitzen. Die notwendigen Attributwerte werden in der Attributliste verwaltet und zur Überprüfung der Autorisierungsentscheidung hinzugezogen. Bei einer erfolgreichen Autorisierung wird dem Benutzer ein UNICORE-Login bzw. ein Pool-Account aus der Hash-Map entnommen. Durch die Namensstruktur des Pool-Accounts wird dann die Rolle dem Benutzer zugewiesen. Die Rolle und der Account werden dem NJS weitergegeben.

5.4 Integration von Shibboleth in UNICORE

Für die Architektur der UNICORE-Shibboleth-Integration mussten Möglichkeiten für die Kommunikation der beiden Systeme untereinander gefunden werden. Diese muss trotz verschiedener Benutzeridentitäten und Sprachen für die Authentifizierungs und-Autorisierungsinformationen des Benutzers gelingen. Eine Shibboleth-Identität soll auf eine Grid-Identität abgebildet werden können. Das Austauschen von Authentifizierungs- und Autorisierungsinformationen des Benutzers basiert auf SAML, beim Grid-Umfeld basiert es auf Zertifikaten. Hier stellt sich die Frage, wie die notwendigen Benutzerinformationen, die auf der XML-Sprache SAML beruhen, für die Grid-Welt und deren Aktivitäten zugänglich gemacht werden können. Darüber hinaus muss geklärt werden, welcher Ansatz für das Erhalten der Benutzerattribute am ehesten geeignet ist (Attribute Pull oder Attribute Push).

Es wurden drei Architekturvorschläge bearbeitet, die die Kluft zwischen verschiedenen Identitäten und Sprachen überbrücken sollen. Diese wurden ebenfalls vom Fraunhofer Institut SCAI und vom Forschungsinstitut Jülich bewertet. Schließlich wurde ein Architekturkonzept für die Realisierung der Integration ausgewählt. Diese Arbeit beschreibt und bewertet die drei Architekturvorschläge.

5.4.1 Einbehaltung der Shibboleth-Kommunikation

Der erste Vorschlag der Architektur beruhte stark auf dem Shibboleth-Konzept. Die notwendigen Schritte von Shibboleth sollten bei der Integration mitberücksichtigt werden. Dagegen sollte der UNICORE-Client die Rolle des Service Providers einnehmen. Daher muss der UNICORE Client die SAML-Sprache interpretieren, um die Assertions interpretieren zu können. Die Attribute des Shibboleth-Nutzers werden nach dem erfolgreichem Austausch der Authentifizierungs- und Autorisierungsinformationen in die Extensions eines temporären Zertifikats eingefügt. Durch diesen Mechanismus wird eine Shibboleth-Identität auf eine Grid-Identität abgebildet. Diagramm 14 stellt die Architektur der Integration dar.

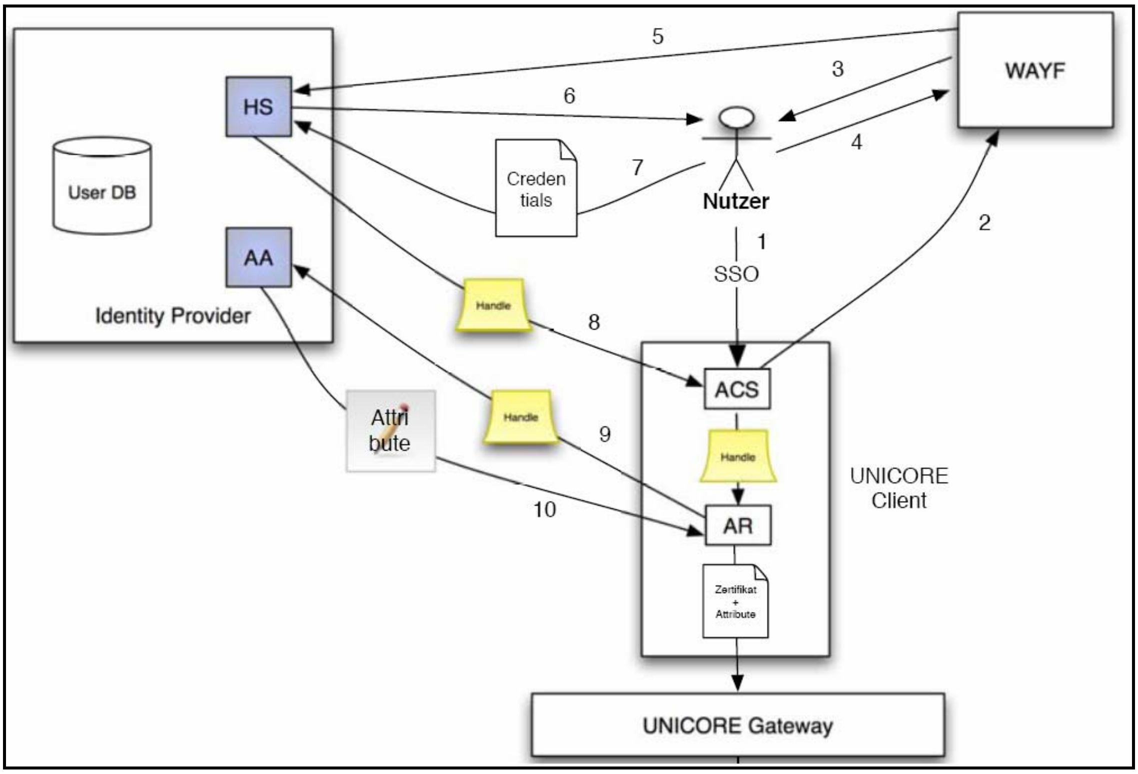

Diagramm 14: Integrationslösung durch Einbehaltung der Shib.-Kommunikation

Für die Anwendung von UNICORE durch einen VO-Benutzer wären dann folgende Schritte notwendig:

1. Der Benutzer startet den UNICORE-Client und stellt eine Anfrage an eine durch Shibboleth geschützte VO-Ressource.

2. Der UNICORE-Client besitzt einen Assertion Consumer Service. Dieser benötigt eine erfolgreiche Authentifizierung des Benutzers (Authentication Assertion) und startet daher das WAYF-Tool.

3. Das WAYF-Tool bietet dem Benutzer eine Liste der Identity Provider, die der VO angehören.

4. Der Benutzer wählt seinen Identity Provider aus.

5. Der WAYF Tool leitet den Benutzer zu seinem Identity Provider (Handle Service) weiter.

6. Der Handle Service fordert den Benutzer auf, sich zu authentifizieren.

7. Der Benutzer authentifiziert sich auf vertrautem Wege gegenüber seinem IdP.

8. Der Handle Service überprüft die Identifizierung des Benutzers. Nach einer erfolgreichen Authentifizierung erstellt er einen eindeutigen Handle und sendet diesen zum UNICORE-Client.

9. Der Attribute Consumer Service des UNICORE-Clients generiert eine Session und übergibt den erhaltenen Handle an den Attribute Requester. Der AR nutzt das Handle, um Attribute des Nutzers bei der Attribute Authority (AA) des IdP abzufragen.

10. Die Attribute Authority sendet unter Berücksichtigung der ARP eine Attribute Assertion mit den Attributen des Benutzers an den AR zurück. Der UNICORE Client erstellt nach Erhalten der Benutzerattribute ein temporäres X.509-Zertifikat und leitet dieses dem Gateway und dem NJS weiter. Der UUDB entscheidet anhand der Attribute, die in das temporäre Zertifikat eingefügt sind, über das Gewähren oder Verweigern des Zugangs des Benutzers zu seiner gewünschten Ressource.

Vorteile dieser Architektur:
Die Vorteile dieser Architektur sind unter anderem, dass die Shibboleth-Kommunikation und die hierfür notwendigen Schritte vollständig in der Integration berücksichtigt wurden. Durch das Verschicken der Handles wird ein zusätzlicher Schutzmechanismus unterstützt. Es werden aufgrund des Attribute-Push-Verfahrens nur die gewünschten Attribute des Benutzers übertragen.

Nachteile dieser Architektur:
Für diese Integrationsform müssten Veränderungen im UNICORE-Kern stattfinden. Somit wäre eine Plugin-Erstellung nicht möglich. Die Architektur wäre für den Benutzer nicht praktisch, da er erstens ein Passwort für das Öffnen seines UNICORE-Keystores benötigt und sich darüber hinaus noch bei seiner Heimatorganisation zu authentifizieren hat. Diese Vorgänge müssten bei jedem Starten des UNICORE-Clients erfolgen.

5.4.2 Integration durch ein externes Modul

Bei diesem Vorschlag soll durch ein externes Modul namens Shibboleth-UNICORE-Module die Kommunikation zwischen beiden Systemen ermöglichen. Das Modul soll nach einer erfolgreichen Authentifizierung des Benutzers bei seinem IdP eine SLC erstellen und diese UNICORE-Client zukommen lassen. Das Erstellen des SLC wird mit Hilfe von MyProxy realisiert. MyProxy ist ein Online Credential Repository, welches nach einer Authentifizierung kurzlebige Credentials (wie zum Beispiel SLC) generiert.

Das Shibboleth-UNICORE-Modul soll die Brücken der Identität und der Sprache schließen, ohne den UNICORE-Client erheblich verändern zu müssen. Die SLCs werden in den UNICORE-Keystore des Benutzers eingefügt und wie gewöhnliche Benutzerzertifikate behandelt, mit dem Unterschied, dass ihre Extensions Attributinformationen beinhalten.

Diagramm 15 beschreibt die Schritte dieses Architekturvorschlags.

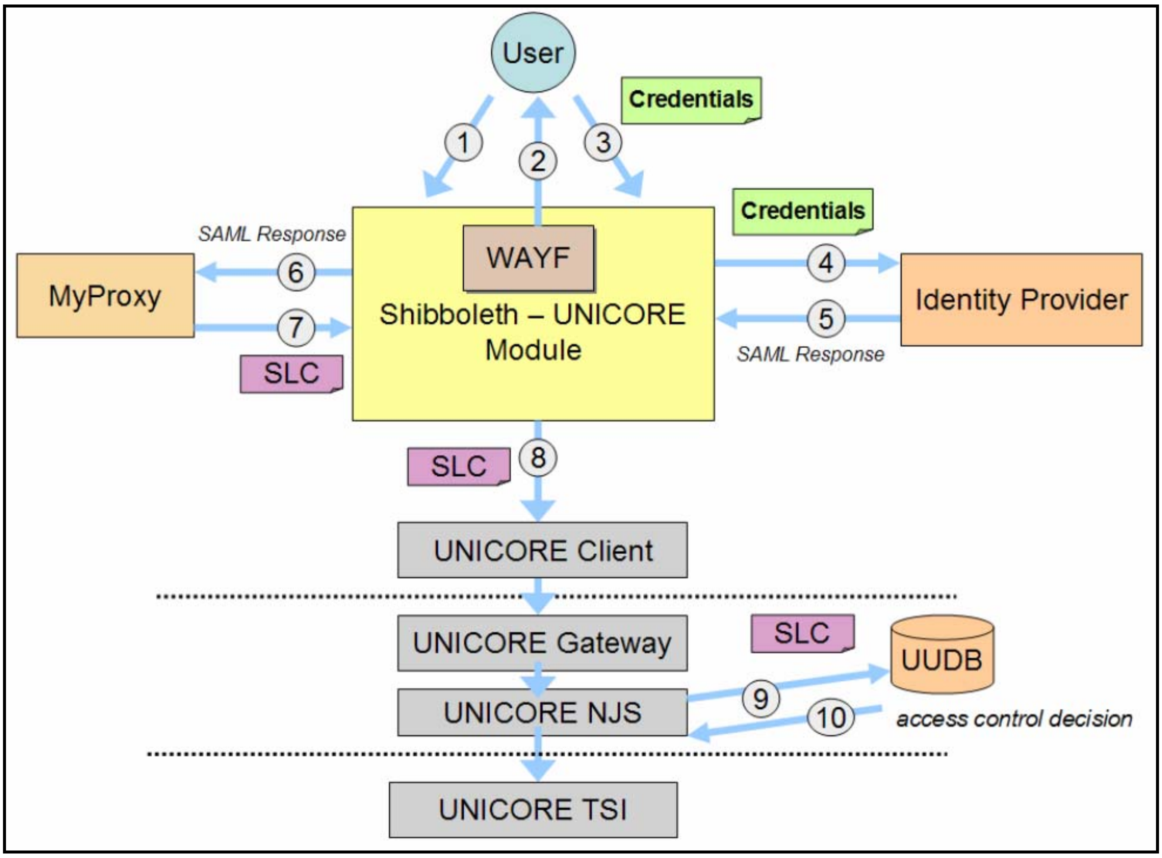

Diagramm 15: Integrationslösung durch ein externes Modul

Die einzelnen Schritte sind:

1. Der Benutzer startet das Shibboleth-UNICORE-Modul.
2. Das Modul überprüft, ob der Benutzer einen gültigen SLC besitzt. Falls ja, wird Schritt 8 ausgeführt. Wenn nicht, präsentiert es dem Benutzer eine Liste von IdPs, die der VO angehören.
3. Der Benutzer selektiert seinen IdP und gibt seine Credentials ein. Das Eingeben seiner Credentials kann durch Wissen oder durch einen Token erfolgen. Wissen kann z.B. die Eingabe eines Passwortes bedeuten. Ein Token wäre zum Beispiel ein X.509-Zertifikat.
4. Das Shibboleth-UNICORE-Modul authentifiziert den Benutzer mit Hilfe des Benutzer-Credentials an dessen IdP.
5. Nach einer erfolgreichen Authentifizierung sendet der IdP dem Shibboleth-UNICORE-Modul eine SAML-Responce-Nachricht. Die Response-Nachricht beinhaltet eine Authentifizierung -und eine Attributassertion.
6. Das Shibboleth-UNICORE-Modul schickt die Nachricht an MyProxy weiter.
7. Durch MyProxy wird ein kurzlebiges X.509-Zertifikat (SLC) generiert, welches die Attribute Assertion in den Extensions des Zertifikats einfügt. Das SLC wird dann dem Shibboleth-UNICORE-Modul zurück gesendet.
8. Das Modul fügt das erhaltene SLC in den UNICORE-Keystore des Benutzers ein. Das Modul startet den UNICORE-Client und benutzt das SLC für die UNICORE-Benutzer-Authentifizierung.
9. Nachdem das SLC vom UNICORE-Client zum Gateway und zum NJS weitergeleitet worden ist, sendet der NJS das SLC zu der UUDB.
10. Die UUDB extrahiert aus der SLC die Attribute des Benutzers und autorisiert den Benutzer anhand der extrahierten Attribute.

Vorteile dieser Architektur:

Die Vorteile der Architektur sind zum einen, dass der UNICORE-Client nicht wesentlich verändert werden muss. Es muss lediglich der Keystore des Benutzers neu erstellt werden. Zum anderen kann ein SLC eine Lebensdauer von ungefähr 11 Tagen besitzen. In dieser Spanne könnte eine Authentifizierung gegenüber einem Identity Provider entfallen.

Nachteile dieser Architektur:
Bei der Weitergabe der Benutzercredentials an den Identity Provider entsteht ein zusätzlicher sicherheitskritischer Bereich. Der Benutzer soll seine Credentials z.B. in Form eines Passwortes dem UNICORE-Shibboleth-Modul weitergeben. Das Modul soll dann den Benutzer mit seiner Heimatorganisation authentifizieren. Dieser Mechanismus erfordert eine sichere Weitergabe der Credentials. Er gibt aber auch einen weiteren Angriffspunkt der UNICORE-Shibboleth-Integration preis.
Beide Architekturvorschläge bieten keine Möglichkeit der Interoperabilität mit anderen Integrationen von Shibboleth in Grid-Middleware-Systeme an.

5.4.3 Integration durch GridShib-CA und UNICORE-Plugin

Ein anderer Vorschlag sollte erarbeitet werden, um eine Interoperabilität zu anderen Integrationen von Shibboleth in Grid-Middleware zu ermöglichen. Da die DFN als Online Service die GridShib-CA nutzt und diese für die Erstellung von SLCs verwenden will, bietet es sich an, eine Möglichkeit zu finden, die GridShib-CA für die Bearbeitung eines Architekturvorschlags ebenfalls zu integrieren.
Die GridShib-CA ist für das Abbilden einer Shibboleth-Identität auf eine Grid-Identität in Form eines SLC zuständig. Der UNICORE-Client soll das SLC aufnehmen können, dieses in ein Benutzerzertifikat transformieren und in den Keystore des Benutzers einfügen. Zusätzlich muss die UUDB die SAML Assertions des Benutzers extrahieren können und diese in ein SAML-Objekts transformieren, um Funktionalitäten wie Validierung, Verifizierung und das Entnehmen der Benutzerattribute zu ermöglichen.

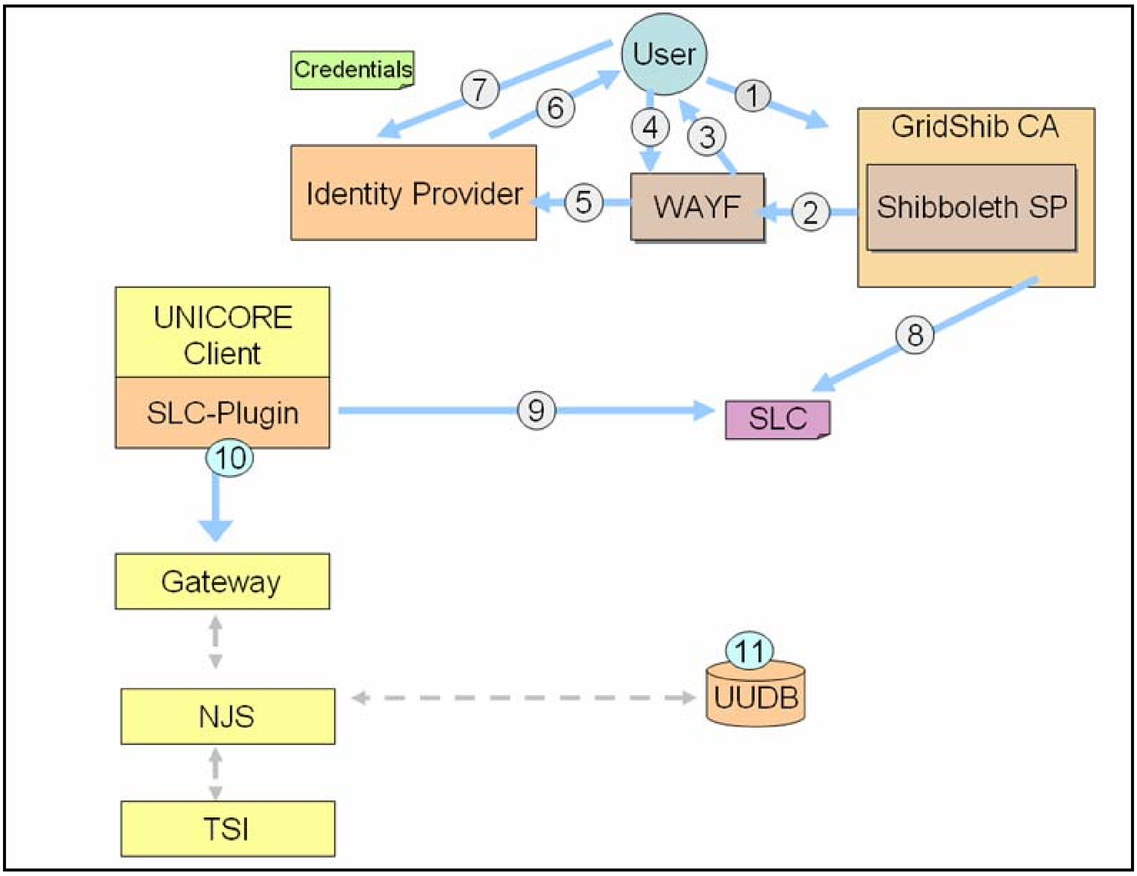

Diagramm 16: Integrationslösung durch GridShib-CA und UNICORE-Plugin

Die Schritte des Vorschlages sind:

1. Der Benutzer greift auf den Online-Dienst GridShib-CA zu.
2. Der Dienst GridShib-CA fungiert als Service Provider und leitet den Benutzer an einen WAYF-Service weiter.
3. Der WAYF-Service gibt dem Benutzer die Möglichkeit, seine Heimatorganisation auszuwählen.
4. Der Benutzer wählt seinen IdP aus.
5. Der WAYF-Service leitet den Benutzer per Redirect zu seinem IdP weiter.
6. Der Benutzer gibt dem IdP seine Credentials weiter.
7. Der IdP authentifiziert den Benutzer anhand seiner Credentials und schickt die erhaltenen Assertions der GridShib-CA weiter.

8. Nach einer erfolgreichen Authentifizierung erstellt die GridShib-CA ein SLC. Die erhaltenen Assertions werden in die Extension des SLCs eingebettet.

9. Das zu implementierende SLC-Plugin soll das SLC des Benutzers entgegennehmen und dieses als Benutzerzertifikat für UNICORE verwenden.

10. Das SLC, welches nun als Benutzerzertifikat dient, wird an das Gateway und den NJS weitergeleitet. Dies setzt aber voraus, dass das Gateway und der NJS den erstellten SLC vom GridShib-CA vertrauen. Der NJS sendet dann das SLC an die UUDB weiter.

11. Die UUDB extrahiert die Assertions aus der Zertifikatsextension. Sie entnimmt die Benutzerattribute aus den Assertions und trifft anhand dieser eine Autorisierungsentscheidung.

Vorteile dieser Architektur:
Durch die Verwendung vom GridShib-CA kann eine Interoperabilität zur Middleware Globus ermöglicht werden, da Globus die GridShib-CA nutzt. Die Middleware gLite verwendet eine modifizierte GridShib-CA [MAR06]. Für das Realisieren der Interoperabilität zwischen den verschiedenen Grid-Middleware bei der Verwendung von Shibboleth muss lediglich eine einheitliche Darstellung der SLC erarbeitet werden. Das UNICORE-SLC-Plugin kann leicht erweitert und für andere VO–Management-Systeme, die SLC generieren, benutzt werden. Der Vorteil dieser Architektur ist, dass man UNICORE mit einem SLC-Plugin erweitert. Dafür wäre kein externer Modul mehr notwendig.

Nachteile der Architektur:
Der Nachteil dieser Architektur ist, dass sie keinen Attribute-Pull-Ansatz unterstützt. Da aber in Zukunft eher das Attribute-Push angewendet wird [GRO06], kann auf diese Funktionalität verzichtet werden.

Diese Architektur wurde für die Implementierung von Shibboleth in UNICORE gewählt, da sie eine Interoperabilität zu VO-Technologien bietet. Die Architekturen VOMS-UNICORE und Shibboleth-UNICORE wurden bewusst so gewählt, dass die Kombination der beiden Systeme möglich ist.

5.5 Integration von VOMS in UNICORE

Der Abschnitt ist unterteilt in zwei Bereiche: Zunächst wird beschrieben, welche Sicherheitsschritte durchgeführt werden müssen, um aus den VOMS-Credentials die VOMS-Attribute zu extrahieren. Anschließend werden verschiedene Architekturvorschläge für die Integration von VOMS in UNICORE vorgestellt und bewertet.

5.5.1 Extrahieren der VOMS-Attribute

VOMS verwendet X.509-Zertifikate, um folgende Aufgaben zu lösen:

- Authentisieren des Benutzers
- Authentisieren des VOMS-Servers
- Signieren des Attributzertifikats
- Signieren des Proxy-Zertifikats

Verschiedene Zertifikatsarten werden für das VOMS-System eingesetzt:

- Benutzerzertifikate (z.B. `usercert.pem`, `userkey.pem`)
- Server-Zertifikate (z.B. `hostcert.pem`, `hostkey.pem`)
- CA-Zertifikate (z.B. `cacert.pem`)
- Proxy- und Attributzertifikate (z.B. `X.509_up_{UID}`)

VOMS basiert auf dem Sicherheitsmodell der *Grid Security Infrastructure*. GSI erwartet ein /`etc/gridsecurity`-Verzeichnis, in dem zum Beispiel die Server- und CA-Zertifikate abgelegt werden. Nach CA-Zertifikaten wird im Verzeichnis `/etc/grid-security/certificates` gesucht. Dagegen werden Server-Zertifikate in das Verzeichnis `/etc/gridsecurity` eingefügt. Der Benutzer hat die Aufgabe, in seinem Home-Verzeichnis ein Unter-Verzeichnis mit dem Namen `.globus` zu erstellen und sein Benutzerzertifikat sowie seinen privaten Schlüssel in diesem Verzeichnis abzulegen. Damit das VOMS-System weiß, bei welchem Server-Zertifikat es sich um das VOMS-Server-Zertifikat handelt, wird die GSI mit dem Verzeichnis `/etc/gridsecurity/vomsdir` erweitert. Das VOMS-Server-Zertifikat muss in dieses Verzeichnis eingefügt werden.

Das Attributzertifikat wird vom VOMS-Server signiert und in das Proxy-Zertifikat als Extension eingefügt. Das Proxy-Zertifikat hingegen wird vom Benutzer signiert. Wie bereits im Abschnitt 3.4.5 beschrieben, geschieht sowohl die Authentifizierung als auch die Signierung durch das Ausführen des Kommandos `voms-proxy-init`. Das Proxy-Zertifikat (PZ), der private Schlüssel des PZs sowie das Benutzerzertifikat werden in einer Credential-Datei im `/tmp`-Verzeichnis mit folgender Namenssyntax gespeichert:

`x509_up_u{UID}` z.B. `x509_up_u1004`

Bevor UNICORE die VOMS-Attribute aus dem Attributzertifikat extrahieren kann, müssen die Schritte des folgenden Sicherheitsmodells ausgeführt werden:

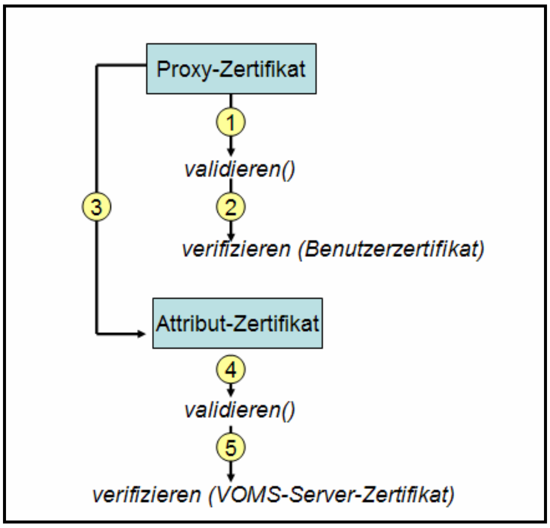

Diagramm 17: Sicherheitsmodell für VOMS

1. UNICORE muss das erhaltene Proxy-Zertifikat validieren, d.h. überprüfen, ob das Proxy-Zertifikat noch gültig ist.

2. Das Proxy-Zertifikat wird mit dem Benutzerzertifikat verifiziert. So wird festgestellt, ob der Benutzer das Proxy-Zertifikat signiert hat. Dadurch kann UNICORE feststellen, ob das Proxy-Zertifikat tatsächlich dem Benutzer gehört.

3. Nachdem Validierung und Verifizierung durchgeführt wurden und dadurch UNICORE dem Proxy-Zertifikat vertraut, darf das Attributzertifikat aus der Proxy-Zertifikats-Extension extrahiert werden.

4. Die Gültigkeit des Attributzertifikats wird überprüft.

5. Der letzte Schritt für die Überprüfung der Sicherheit bezieht sich auf die Verifizierung des Attributzertifikates. Dadurch kann UNICORE sicherstellen, dass die VO-Attribute von einer Instanz kommen, der UNICORE vertraut.

Falls die obigen Sicherheitsüberprüfungen erfolgreich waren, d.h. falls das PZ über das AZ verifiziert und validiert wurden, können die VO-Attribute aus dem AZ extrahiert und für die Autorisierung verwendet werden.

5.5.2 Allgemeine Integrationsarchitektur von VOMS und UNICORE

Um eine VO-basierte Autorisierung in UNICORE zu ermöglichen, muss ein VO-Modul ins UNICORE-System integriert werden, das Benutzer-Credentials mit VO-Autorisierungs-Informationen erstellt. Für die Integration von VOMS in UNICORE müssen zwei Komponenten implementiert werden:

- VOMSPlugin für den UNICORE Client
- UUDB-Erweiterung für VOMS-VO-FQANs

VOMSPlugin:

Im Allgemeinen muss das VOMSPlugin die Übertragung des VO-Credentials zum NJS ermöglichen. In VOMS ist das VO-Credential ein Attributzertifikat, das in ein Proxy-Zertifikat integriert ist. Deswegen muss entweder das Attribut- oder das Proxy-Zertifikat weitergeschickt werden.

Erweiterte UUDB:

Die UUDB-Implementierung soll UNICORE durch einen VO-basierten Autorisierungsmechanismus erweitern. Konkreter sollen RBAC- und ABAC-Autorisierungen eingesetzt werden können. Damit UUDB das Mapping anhand von VO-Attributen durchführt, muss in der Implementierung sichergestellt werden, dass das UNICORE-System den Attributen vertraut. Dem UNICORE-Administrator soll die Möglichkeit gegeben werden, zwischen identitätsbasierter und rollenbasierter Autorisierung (RBAC) zu wechseln.

Die Schwierigkeit der Integration liegt in der Fragestellung, wie das VOMS-Credential, ob in Form eines Proxy- oder Attributzertifikats, an die NJS-Komponente weitergereicht werden kann. Die Integrationslösung muss die Anforderungen, die in Abschnitt 4.1 beschrieben wurden, erfüllen.

Diese Problematik hat dafür gesorgt, dass für die Integration von VOMS in UNICORE drei Architekturvorschläge vom Fraunhofer Institut SCAI und dem Forschungsinstitut Jülich unterbreitet und besprochen wurden. Aufgrund der beschriebenen Ziele und nach der Abwägung von Vor- und Nachteilen der Vorschläge entschied man sich für eine Integrationsarchitektur. Im Folgenden werden die drei Vorschläge und deren Bewertungen beschrieben.

5.5.3 Architekturvorschlag durch AJO-Integration

Das Proxy-Zertifikat wird durch das VOMS-Plugin als *site-specific-object* in das AJO eingefügt. Da das AJO-Objekt zu den UNICORE-Komponenten weitergereicht wird, wenn ein Job ausgeführt wird, kann das Proxy-Zertifikat so zu der NJS-Komponente gelangen.

Das untere Diagramm und die darauf folgende Liste beschreiben die Integrationsarchitektur:

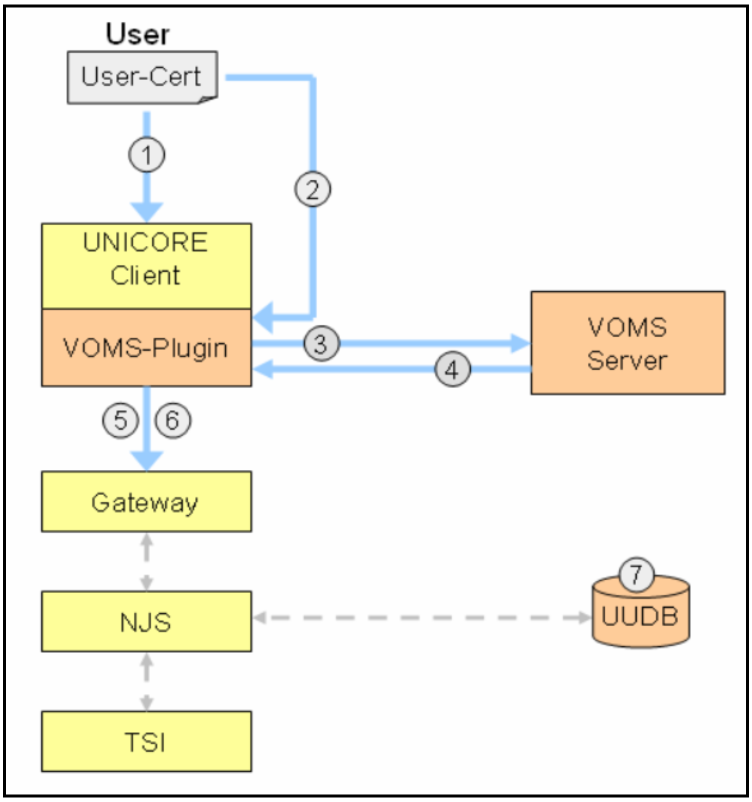

Diagramm 18: Integrationsarchitektur der AJO-Lösung

1. Der Benutzer authentisiert sich gegenüber dem UNICORE-Client mit seinem permanenten Benutzerzertifikat.

2. Der Benutzer spezifiziert seinen VOMS-Request, indem er das VOMS-Plugin verwendet.

3. Das VOMS-Plugin überprüft die Syntax des Requests, validiert die Identität des Benutzers und sendet den Request zum VOMS-Server.

4. Der VOMS Server validiert den Request und sendet die benötigten Autorisierungsinformationen in Form eines Attributzertifikats an das VOMS-Plugin zurück.

5. Das VOMS-Plugin erstellt ein Proxy-Zertifikat und erweitert es um das empfangene Attributzertifikat.

6. Der Benutzer submittiert einen Job mit den gewöhnlichen UNICORE-Mechanismen. Das VOMS-Plugin fügt das Proxy-Zertifikat im AJO als SSO-Objekt. Der UNICORE-Client sendet das AJO mit Hilfe des UNICORE-Protokolls durch die Gateway-Komponente zum NJS.

7. Die erweiterte UUDB bildet die VO-FQANs auf lokale Accounts ab.

Vorteile der Architektur:

Der wesentliche Vorteil dieser Architektur ist, dass die UNICORE 5-Authentifizierung nicht verändert wird und die Benutzer-Identität (P12-keystore) nicht um VO-Credentials erweitert werden muss. Die VO-Autorisierungsinformationen werden unabhängig davon zum NJS geschickt, wenn ein Job ausgeführt wird. Das Einfügen eines Proxy-Zertifikats in ein SSO-Objekt und die daraus resultierende Möglichkeit dieses durch die UNICORE-Komponente zu verschicken, wurde bereits beim *GRIP-Projekt* (Grid Interoperability Project) [RAM02] im Rahmen eines Europäischen Projekts realisiert, welches das Ziel hatte, eine Interoperabilität zwischen den Grid-Middleware UNICORE und GLOBUS zu erreichen.

Nachteile der Architektur:

Ursprünglich war die SSO-Lösung der festgelegte Architekturvorschlag. Nach näherer Beobachtung der NJS-Implementierung wurde festgestellt, dass es keine Schnittstelle gibt, die es ermöglicht, *vor* der Autorisierung an das AJO-Objekt zu gelangen. Das wiederum bedeutet, dass diese Lösung gegen die Anforderung, den UNICORE-Kern nicht zu verändern, verstößt. Die NJS-Kern-Implementierung müsste dadurch geändert werden. Die Communities, die bereits UNICORE nutzen, müssten die NJS-Komponente komplett neu installieren, um eine VO-basierte Autorisierung durchführen

zu können. Die Lösung konzentriert sich zu sehr auf VOMS und bietet keine Interoperabilität zu anderen VO-Management-Systemen.
Aus den genannten Gründen wurde entschieden, dass dieser Vorschlag keine geeignete Lösung für die Integration ist.

5.5.4 Architekturvorschlag durch einen SLC-Service

Ein SLC-Service wird verwendet, um ein SLC für den Benutzer zu erstellen. Der Service signiert das SLC und fügt in ihm das VOMS-Proxy-Zertifikat des Benutzers hinzu. Wenn das UNICORE-System dem SLC-Service vertraut, kann der Benutzer das SLC als Benutzer-Credential verwenden, um Zugang zu UNICORE zu erhalten.
Das untere Diagramm beschreibt die Integrationsarchitektur:

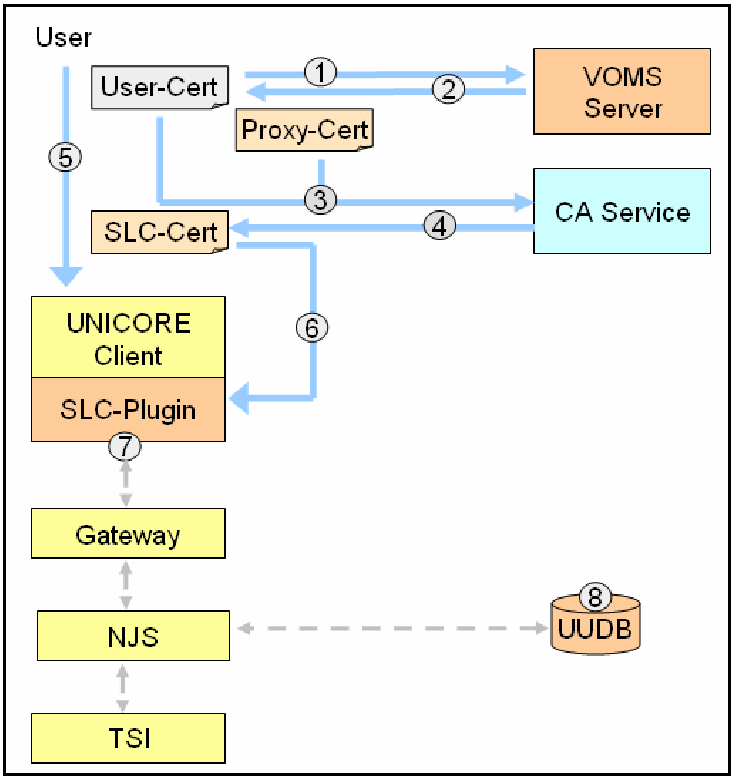

Diagramm 19: Integrationslösung durch ein SLC-Service

1. Der Benutzer führt *voms-proxy-init* aus und spezifiziert sein VOMS-Request. Dadurch wird sein VOMS-Request an den VOMS-Server gesendet.

2. Der VOMS-Server validiert den Request und sendet die benötigten Autorisierungsinformationen in Form eines Attributzertifikats an den Benutzer zurück. Der *voms-proxy-init*–Mechanismus erstellt anschließend ein Proxy-

Zertifikat und fügt das Attributzertifikat in ihm hinzu. Der Benutzer signiert das Proxy-Zertifikat mit seinem privaten Schlüssel.

3. Der Benutzer authentisiert sich mit seinem Benutzerzertifikat an einem SLC-Service und sendet sein Proxy-Zertifikat dem Service zu.

4. Der SLC-Service erstellt ein SLC und fügt das Proxy-Zertifikat als SLC-Extension ein. Der SLC-Credential-Service signiert das SLC und sendet es an den Benutzer zurück.

5. Der Benutzer startet den UNICORE-Client, ohne sich mit einem Zertifikat zu authentisieren.

6. Der Benutzer führt das SLC-Plugin aus und gibt sein SLC an das Plugin weiter.
7. Das SLC-Plugin validiert und verifiziert die Credentials und erstellt für den Benutzer einen UNICORE-Keystore auf Basis des SLCs. Anhand des erstellten Keystores kann der Benutzer beim Gateway authentisiert werden.

8. Die erweiterte UUDB extrahiert die VO-FQANs aus dem SLC und bildet die VO-FQANs auf lokale Accounts ab.

Vorteile der Architektur:

Im Gegensatz zur vorherigen Architektur bleibt die NJS-Kern-Implementierung unberührt. Durch den SLC-Service wird das Proxy-Zertifikat in das SLC eingefügt und muss nicht separat zu der NJS-Komponente gesendet werden. Auch der UNICORE-Authentisierungsmechanismus wird durch diese Realisierung nicht verändert. Durch den Einsatz eines SLC-Services können weitere Attribute bzw. Credentials in das SLC integriert werden. Diese Lösung kann für verschiedene VO-Management-Systeme verwendet werden.

Nachteile der Architektur:

Der wesentliche Nachteil ist, dass das D-Grid momentan nicht über solch einen SLC-Service verfügt. Der Service muss allen im D-Grid verwendeten VO-Servern und VO-Benutzern vertrauen. Die Implementierung soll Integrations-Lösungen für die D-Grid-Communities bereitstellen, die sofort verwendet werden können. Diese Anforderung würde nicht erfüllt.

Obwohl es sich beim Benutzer-Credential um ein Benutzerzertifikat handelt, muss ein weiteres erstellt werden.
Aus den genannten Gründen kam auch dieser Vorschlag nicht in Frage.

5.5.5 Architekturvorschlag durch eine Keystore-Erweiterung

Hierbei verwendet der Benutzer sein Benutzerzertifikat, um Zugang zum UNICORE und dem VOMS-System zu erhalten. Sein Benutzerzertifikat und sein privater Schlüssel werden in UNICORE, wie schon im Abschnitt 5.2.2 beschrieben, in Form eines P12-Keystores abgespeichert. Der UNICORE-Client wird um einen *Proxy-Plugin* erweitert, welches das VOMS-Proxy-Zertifikat in die Certificate-Chain des Benutzer-Keystores einfügt.
Das folgende Diagramm und die nachfolgende Auflistung veranschaulichen den Architektur-Vorschlag:

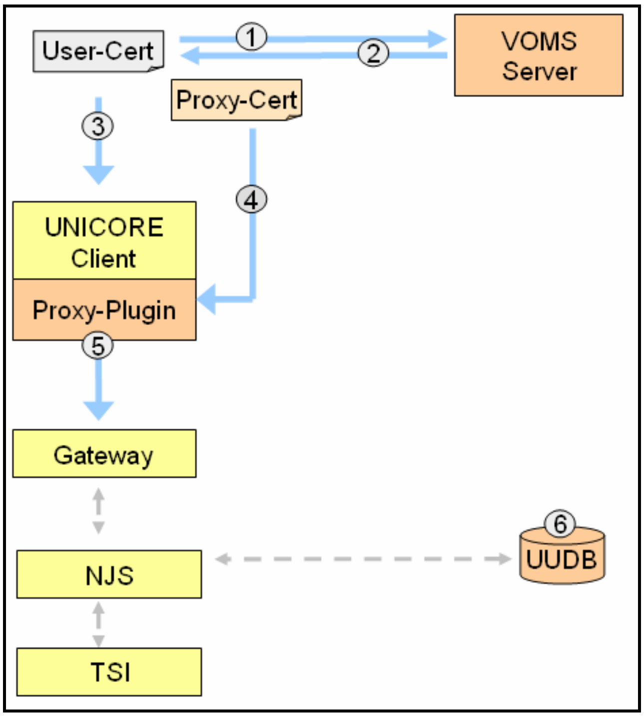

Diagramm 20: Integrationslösung durch eine Keystore-Erweiterung

1. Der Benutzer führt `voms-proxy-init` aus und spezifiziert seinen VOMS-Request. Dieser wird dann dem VOMS-Server zugesendet.

2. Der VOMS-Server validiert den Request und sendet die benötigten Autorisierungsinformationen in Form eines Attributzertifikats an den Benutzer zurück. Der `voms-proxy-init`–Mechanismus erstellt anschließend ein Proxy-

Zertifikat und fügt das Attributzertifikat in ihm hinzu. Der Benutzer signiert das Proxy-Zertifikat mit seinem privaten Schlüssel.

3. Der Benutzer startet den UNICORE-Client und authentisiert sich standardmäßig mit seinem Benutzerzertifikat.

4. Der Benutzer führt das Proxy-Plugin aus und gibt sein Proxy-Zertifikat an das Plugin weiter.

5. Das Proxy-Plugin validiert und verifziert das Proxy-Zertifkat. Anschließend erweitert das Plugin die Certificate-Chain des Benutzer-Keystores um das Proxy-Zertifikat.

6. Die erweiterte UUDB extrahiert die VO-FQANs aus dem Attributzertifikat und bildet die VO-FQANs auf lokale Accounts ab.

Vorteile der Architektur:

Die NJS-Kern-Implementierung und der UNICORE-Authentisierungs-Mechanismus werden mit diesem Realisierungsvorschlag nicht verändert. Interoperabilität mit anderen VO-Management-Systemen ist gegeben. Es muss kein zusätzliches Benutzer-Credential erstellt werden. Diese Integrationslösung kann von den D-Grid-Communities sofort genutzt werden.

Nachteile der Architektur:

Um das Proxy-Zertifikat in den Benutzer-Keystore zu integrieren, muss der Benutzer sein Keystore-Passwort eingeben, um den Keystore zu entsperren. Dies würde jedoch gegen das SSO-Prinzip von UNICORE verstoßen.

Diese Architektur wurde für die Implementierung der UNICORE und VOMS-Integration gewählt, da sie eine Interoperabilität zu anderen VO-Technologien gewährleistet. Trotz des SSO-Nachteils, erfüllt dieser Vorschlag die allgemeinen Integrations-Anforderungen, die im Unterabschnitt 4.1.1 beschrieben wurden. Durch diesen Vorschlag ist die Möglichkeit einer Kombination mit dem Shibboleth-System gegeben.

5.6 Integration der Shibboleth- und VOMS-Architektur

Die Architekturen von UNICORE-Shibboleth und UNICORE-VOMS wurden unter der Maßgabe, die beiden Systeme auch kombiniert verwenden zu können, konzipiert. Der SLC-Plugin ist für die Erstellung eines Benutzer-Credentials zuständig. Der Proxy-Plugin erweitert die Benutzeridentität mit Proxy-Zertifikaten. Durch die Plugins und der neuen UUDBVO sind Autorisierungen anhand von Campus- und VO-Attributen möglich. Die folgende Architektur (siehe Diagramm 21) beschreibt die neuen UNICORE-Prozesse.

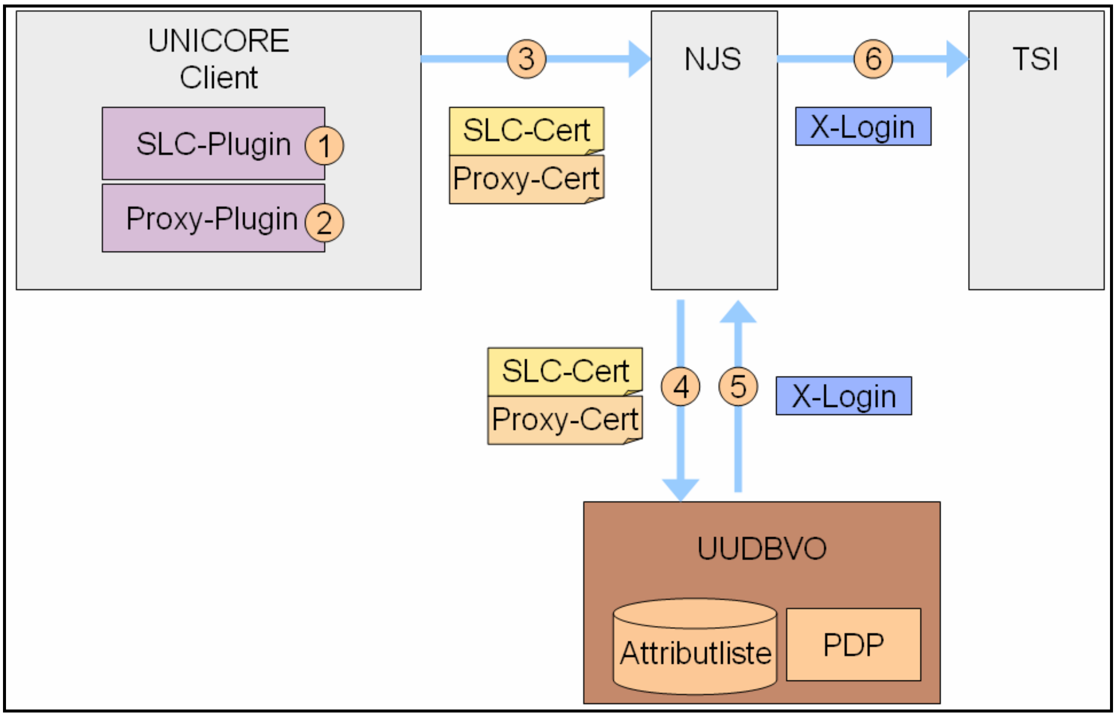

Diagramm 21: Integration der Shibboleth und VOMS-Architektur

Die Schritte für die Anwendung von Shibboleth und VOMS sind:

1. Der Benutzer startet den SLC-Plugin. Das Plugin nützt ihm nur, wenn er bereits Besitzer einer SLC ist. Das Plugin entnimmt den SLC des Benutzers und verwendet ihn als Benutzeridentität für das UNICORE-System.

2. Wenn der Benutzer Mitglied einer VO bei dem System VOMS ist, kann er durch das Proxy-Plugin sein Proxy-Zertifikat dem UNICORE-System für die Weitergabe zur Verfügung stellen.

3. UNICORE sendet den SLC und das Proxy-Zertifikat zum Gateway und NJS.

4. Der NJS sendet die Credentials des Benutzers an die UUDBVO weiter.

5. In dem Benutzer-SLC sind die Campusattribute, in dem Proxy-Zertifikat die VO-Attribute des Benutzers abgespeichert. Die UUDBVO extrahiert die Attribute des Benutzers. Sie entscheidet mit der PDP-Komponente, ob der Benutzer Zugang zu einer VO-UNICORE-Ressource erhält. Die PDP nimmt die Attribute des Benutzers entgegen und vergleicht sie mit der Attributliste. Wenn der Vergleich erfolgreich war, wird dem Benutzer ein XLogin zugestellt. Die UUDBVO sendet das X-Login an den NJS weiter. Der NJS nimmt das X-Login und gibt dem Benutzer anhand diesem eine Rolle.

6. Der NJS sendet das X-Login dem TSI weiter und der Benutzer hat die Möglichkeit, auf UNICORE-Ressourcen zuzugreifen.

5.7 Interoperabilität der UNICORE-Konzeption

Die Erweiterung von UNICORE soll auch eine Interoperabilität zu verschiedenen VO-Management-Technologien ermöglichen. Die Architektur wurde so gewählt, dass flexible Lösungen bereitstehen, die von verschiedenen VO-Technologien angewendet werden können.
Im Bereich der *Authentisierung* muss der UNICORE-Client um zwei Funktionen erweitert werden:

- **SLC-Plugin:**
  Der Benutzer verwendet das *SLC-Plugin*, wenn es sich bei seinem Benutzer-Credential nicht um ein Benutzerzertifikat handelt. Das SLC-Plugin hat die Aufgabe, eine Credential-Datei, die das SLC-Zertifikat und den korrespondierenden privaten Schlüssel beinhaltet, in eine UNICORE-Benutzeridentität umzuwandeln (P12-Keystore). Da ein SLC-Service gleichzeitig als VO-Credential-Service fungieren kann, ist es möglich, dass sich VO-Credentials im SLC-Zertifikat befinden.

- **Proxy-Plugin:**
  Der Benutzer verwendet das *Proxy-Plugin*, um seine UNICORE-Benutzeridentität um VO-Credentials zu erweitern. Diese müssen in einem Proxy zusammengefasst sein, damit UNICORE sie verifizieren kann. Dadurch kann UNICORE sicherstellen, dass die VO-Credentials eindeutig dem Benutzer zuzuordnen sind. Es können beliebig viele Proxy-Zertifikate in den Keystore des Benutzers eingefügt werden.

Die attributbasierte VO-*Autorisierung* muss gegenüber der identitätsbasierten Lösung um zwei zusätzliche Komponenten erweitert werden:

- **Attribute-Issuer:** vertrauenswürdige Instanzen, die die VO-Attribute signieren.

- **Policy-Decision-Point (PDP):** Autorisierungsentscheidungs-Mechanismen, die bei den verschiedenen *Attribute Consumern* zum Einsatz kommen.

Attributbasierte Autorisierungen werden schon in einigen Grid-Umgebungen verwendet. Daher werden im Folgenden einige Attribute-Issuers und Attribute-Consumers vorgestellt. Von besonderer Wichtigkeit ist, welche Credentials von den Attribute-Issuers signiert werden. Die Attribute-Consumers müssen PDP-Lösungen anbieten, die

die VO-Attribute aus den Credentials extrahieren und die Benutzer anhand der Attribute autorisieren.
Zurzeit werden folgende Attribute-Issuer und Attribute-Consumer in Grids verwendet:

Attribute-Issuer:

- VOMS: Im VOMS-System werden die Attribute in Attributzertifikaten zusammengefasst. In der nächsten VOMS-Version sollen anstatt von Attributzertifikaten SAML-Assertions eingesetzt werden.
- myVocs: Das myVocs-System speichert die Attribute in SAML-Assertions.
- GridShib-CA: Das Attribut-Credential vom GridShib-CA-Service entspricht SAML-Assertions

Attribute-Consumer:

- Globus-Toolkit 4.0 (WS Komponenten): optionale PDPs für SAML sowie Attributzertifikate. Diese sollen für die Version GT4.2 fest eingesetzt werden.
- gLite: Attributzertifikate, nur anhand von FQANs

UNICORE 5 und 6 bieten momentan nur eine identitätsbasierte Autorisierung an. Daher sollen die UNICORE-Erweiterungen der vorliegenden Arbeit ermöglichen, dass die Attribute-Credentials, die von den Attribute-Issuern signiert werden, im UNICORE-System für die Autorisierung angewendet werden können. Die UNICORE-PDP verarbeitet deswegen SAML-Assertions sowie Attributzertifikate. gLite ist bis jetzt noch die einzige Middleware, in der die attributbasierte VO-Autorisierung fest verankert ist. Die Autorisierungsentscheidung ist bei gLite jedoch auf das hierarchische Modell der FQANs beschränkt [WP3-07].
Es gibt zurzeit keine Möglichkeit, Campusattribute zusätzlich für die Autorisierungsentscheidung zu verwenden.
Die zu implementierende UNICORE-PDP soll nicht nur FQAN-Attribute verarbeiten können, sondern auch die Möglichkeit bieten, anhand von Campusattributen zu autorisieren. Zusätzlich sollen Campus- sowie VO-Attribute für die Autorisierungsentscheidung miteinander kombiniert werden können. UNICORE wurde im Rahmen dieser Masterarbeit um eine flexible PDP erweitert, die diese Anforderungen erfüllt.

5.7.1 Kombination von VO- und Campusattributen

In diesem Unterabschnitt werden zwei verschiedene Ansätze vorgestellt, wie man VO- und Campusattribute in Grids kombinieren kann. Anschließend wird analysiert, ob die UNICORE-Erweiterungen diese Ansätze unterstützen.

myVocs mit Trust Proxying

myVocs (siehe Abschnitt 3.4.4) kann durch die Kombination des unten beschriebenen Workflows VO-Credentials kreieren, die Campus- sowie VO-Attribute beinhalten. Die Abbildung 27 zeigt die nachfolgenden Schritte [WP3-07]:

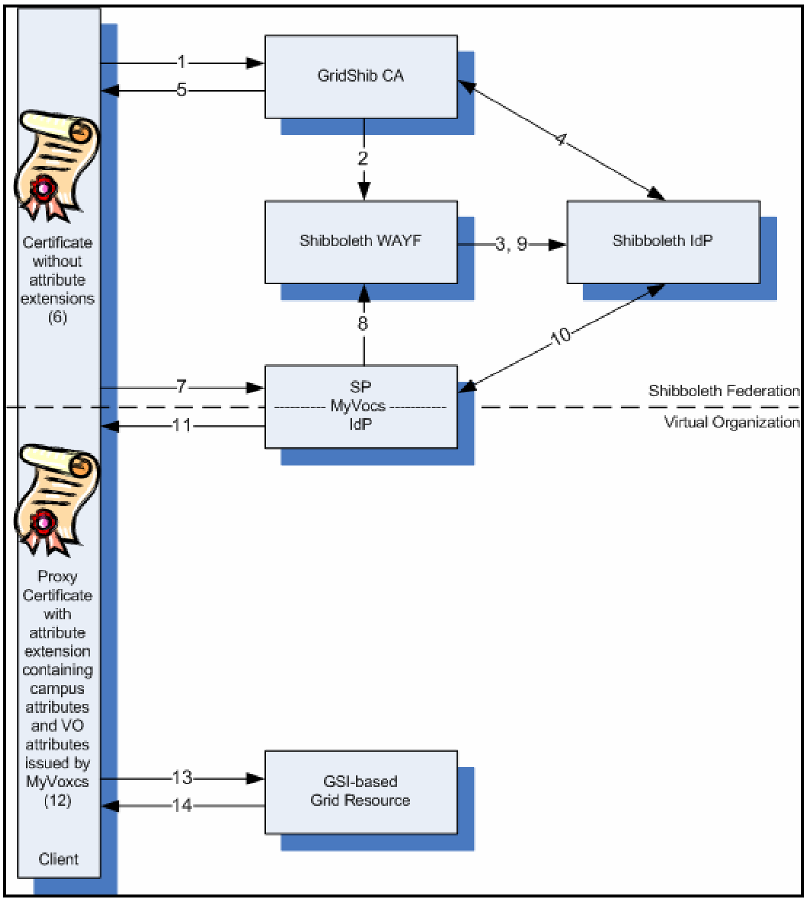

Abbildung 27: Trust-Proxy-Workflow für myVocs [WP3-07]

Falls es sich beim Benutzer-Credential um ein Benutzerzertifikat handelt, beginnt der Workflow bei Schritt 7. Ansonsten gelten folgende Schritte:

1. Der Benutzer führt ein SLCS aus, um ein SLC zu erhalten.

2. Der Benutzer wird zum WAYF-Service der Förderation weitergeleitet.

3. Der Benutzer wählt die IdP seiner Heimatorganisation aus und wird zu dieser weitergeleitet. Der Benutzer authentisiert sich mit seinem Benutzer-Credential am IdP. Bei erfolgreicher Authentifizierung wird er wieder zum SLCS weitergeleitet.

4. Der SLCS autorisiert dem Benutzer anhand der Attribute, die er vom IdP erhält. Der Service erstellt für den Benutzer ein SLC.

5. Der SLCS schickt den Benutzer das erstellte SLC.

6. Der Benutzer ist somit im Besitz eines Zertifikates, das noch keine Campus- oder VO-Attribute beinhaltet.

7. Der Benutzer ruft den myVocs-Service auf, um seine VO und Campusattribute zu erhalten.

8. Für die Authentisierung wird der Benutzer wieder zum WAYF-Service, …

9. …zu seinem IdP…

10. … und schließlich zurück zum myVocs-Service geleitet.

11. Der IdP fügt die Attribute einer SAML-Assertion hinzu und signiert sie. Die Assertion wird als Extension einem Proxy-Zertifikats-Request hinzugefügt und zum Benutzer gesendet. Der Benutzer signiert den Request mit seinem Benutzerzertifikat.

12. Der Benutzer ist nun in Besitz eines Proxy-Zertifikats. Die VO- und Campusattribute sind in Form einer SAML-Assertion im Zertifikat gespeichert.

13. + 14. Der Benutzer kann nun zu jeder GSI-basierten Grid-Ressource Zugang erhalten, indem er sein Proxy-Zertifikat für die Authentisierung verwendet und über seine Attribute autorisiert wird.

Durch die Implementierung der vorliegenden Arbeit kann der myVocs-Benutzer auch UNICORE verwenden, um Zugang auf Grid-Ressourcen zu bekommen. Dazu muss er nach dem 12. Schritt folgendes ausführen:

13 . Der Benutzer startet den UNICORE-Client und führt das SLC-Plugin aus. Der Benutzer übergibt dem Plugin seine SLC-Datei.

14. Das SLC-Plugin erstellt aus der Datei für den Benutzer eine UNICORE-Identität. Damit wird der Benutzer beim Gateway authentisiert.

15. Der Benutzer führt nun das Proxy-Plugin aus, um sein Proxy-Zertifikat dem Plugin weiterzugeben.

16. Das Proxy-Plugin erweitert seine UNICORE-Identität um das Proxy-Zertifikat.

17. Anhand der Attribute im Proxy-Zertifikat autorisiert die neue UUDBVO den Benutzer.

VOMS/VOMRS mit Trust Proxying (GridShib-CA)

Ein GridShib-CA wird verwendet, um die Benutzer-Attribute, die in seiner Heimatorganisation verwaltet werden, in einem SLC zusammenzufassen. Das SLC wird vom GridShib-CA signiert. Wenn das SLC als Benutzer-Credential verwendet wird, kann anschließend VOMS/VOMRS ausgeführt werden, um die VO-Attribute zu erhalten.

Dieses Prinzip wird durch die folgende Abbildung dargestellt [WP3-07]:

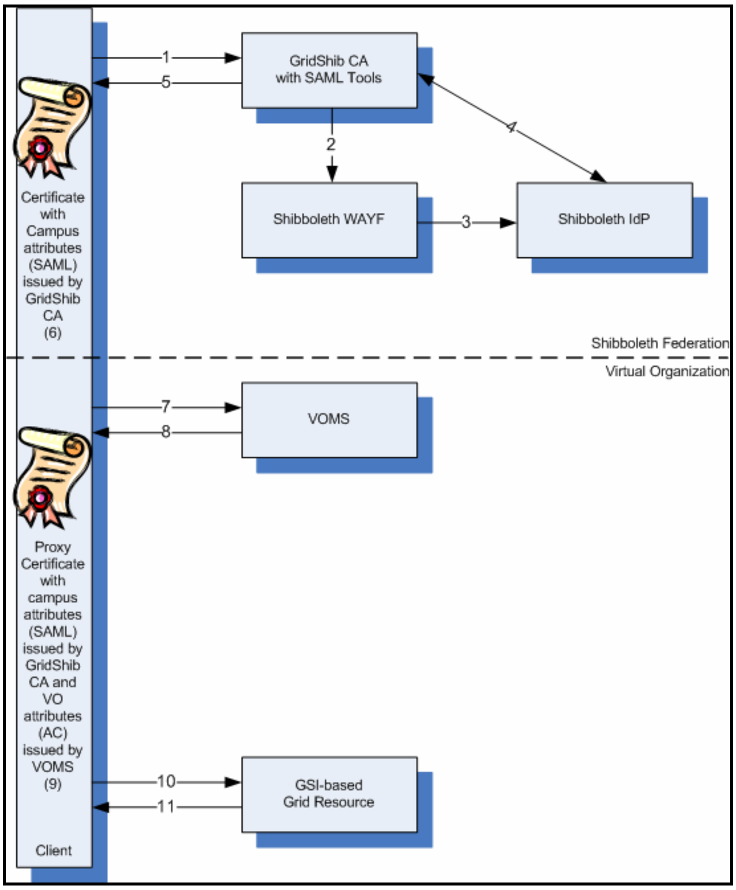

Abbildung 28: VOMS/VOMS mit GridShib-CA [WP3-07]

1. Der Benutzer führt ein SLCS aus, um ein SLC zu erhalten.

2. Der Benutzer wird zum WAYF-Service der Förderation weitergeleitet.

3. Der Benutzer wählt die IdP seiner Heimatorganisation aus und wird zu dieser weitergeleitet. Der Benutzer authentisiert sich mit seinem Benutzer-Credential beim IdP. Bei erfolgreicher Authentifizierung wird er wieder zum SLCS weitergeleitet.

4. Der SLCS fügt die Campusattribute des Benutzers per Attribute-Push als SAML Assertions im SLC hinzu.

5. Der SLCS schickt dem Benutzer das erstellte SLC.

6. Der Benutzer ist im Besitz eines SLC, das die Campusattribute beinhaltet.

7. Der Benutzer erstellt ein Proxy-Zertifikat durch das Ausführen von zum Beispiel `voms-proxy-init`.

8. Die VO-Attribute, die beim VOMS/VOMRS-Server gespeichert sind, werden in

einem Attributzertifikat zusammengefasst und vom Server signiert. Das Attributzertifikat wird dem Benutzer zurückgeschickt, um in einem Proxy-Zertifikat als Extension abgelegt zu werden.

9. Der Benutzer ist nun im Besitz eines Proxy-Zertifikats, das die VO- sowie Campusattribute beinhaltet.

10. +11. Der Benutzer kann nun zu jeder GSI-basierten Grid-Ressource Zugang erhalten, indem er sein Proxy-Zertifikat für die Authentisierung verwendet und über seine Attribute autorisiert wird.

Dieser Ansatz wird von der Implementierung dieser Arbeit unterstützt. Da UNICORE keine Proxy-Delegation anbietet, muss das Proxy-Zertifikat nicht die Campusattribute beinhalten, vielmehr können diese aus dem SLC extrahiert werden. Nach Schritt 9 müssen folgende Schritte ausgeführt werden:

10. Der Benutzer startet den UNICORE-Client und führt das SLC-Plugin aus. Der Benutzer übergibt dem Plugin seine SLC-Datei.

11. Das SLC-Plugin erstellt aus der Datei für den Benutzer eine UNICORE-Identität. Damit wird der Benutzer beim Gateway authentisiert. Im SLC befinden sich die Campusattribute. Das SLC wird später auch für die Autorisierung benötigt.

12. Der Benutzer führt nun das Proxy-Plugin aus, um sein Proxy-Zertifikat an das Plugin weiterzugeben.

13. Das Proxy-Plugin erweitert seine UNICORE-Identität mit dem Proxy-Zertifikat.

14. Anhand der VO-Attribute im Proxy-Zertifikat und der Campusattribute im SLC autorisiert die neue UUDBVO den Benutzer.

Die beiden Ansätze verdeutlichen, dass sich die VO-Credentials im SLC sowie im Proxy-Zertifikat befinden können und dass die Autorisierung anhand von SAML Assertions und Attributzertifikaten durchgeführt werden muss.

6. Implementierung

Für die UNICORE-Erweiterung wurden zwei Plugins (VOMS-Plugin und SLC-Plugin) und die attributbasierte UUDBVO implementiert. Der Unterabschnitt 6.1 beschäftigt sich mit der Implementierung der Plugins. Unterabschnitt 6.2 geht auf die Realisierung der UUDBVO ein.

6.1 Implementierung der Plugins

Durch die Plugins soll gewährleistet werden, dass der Benutzer seine Attribut-Credentials dem UNICORE-System weitergeben kann. Die Implementierung der Plugins wird in folgende Bereiche unterteilt:

- Verarbeitung der Credential-Dateien
- Erstellung sowie Erweiterung einer UNICORE-Benutzeridentität

6.1.1 Verarbeiten der Credential-Dateien

Wie in der Konzeption beschrieben, wird vorausgesetzt, dass sich der Benutzer bevor er UNICORE ausführt seine Attribut-Credentials bei einem VO-Credential-Service bzw. SLC-Service abholt. Die Credential-Services fassen diese Credentials in einer Credential-Datei zusammen. Nachdem der Benutzer diese Datei von einem Credential-Service erhalten hat, muss er sie dem UNICORE-System weitergeben. Dadurch können die Plugins entweder eine Identität für den Benutzer erstellen oder eine bereits bestehende UNICORE-Benutzeridentität mit den Credentials erweitern.
Damit UNICORE eine Credential-Datei verarbeiten kann, muss der Inhalt der Datei in seine ursprüngliche Credential-Form konvertiert werden. Dies wird durch die folgende abstrakte Klasse beschrieben:

```
public abstract class ParseCredentialFile {

      public abstract void setCredentialFile(File file);
      public abstract File getCredentialFile();
      public abstract void parseCredentialFile();
      public abstract Object[] getCredentials();

}
```

Listing 2: ParseCredentialFile.java

Im Rahmen der vorliegenden Masterarbeit werden zwei verschiedene Credential-Dateien im UNICORE-System verarbeitet:

- SLC-File
- Proxy-File

SLC-Files beinhalten das SLC-Zertifikat sowie den korrespondierenden privaten Schlüssel. Der private Schlüssel wird dabei nicht durch eine Passphrase geschützt. Schlüssel sowie Zertifikat liegen im PEM-Format vor. Der GridShibSAML-Service speichert z.B. die SLC-Datei (`x509_up_u{UID}`) im `/tmp`-Verzeichnis ab. Das untere Listing veranschaulicht den Inhalt eines SLC-Files:

```
-----BEGIN CERTIFICATE -----
.......                                  SLC-Cert
-----END CERTIFICATE -----

----- BEGIN RSA PRIVATE KEY ------
.........                                SLC-Priv.
----- END RSA PRIVATE KEY -----
```

Listing 3: Inhalt der SLC-File

Proxy-Files fassen das Proxy-Zertifikat, den privaten Proxy-Schlüssel und das Benutzerzertifkat zusammen. Diese werden in der folgenden Reihenfolge und Struktur gespeichert:

```
-----BEGIN CERTIFICATE -----
.......                                  Proxy – Cert
-----END CERTIFICATE -----

----- BEGIN RSA PRIVATE KEY ------
.........                                Proxy – Priv.
----- END RSA PRIVATE KEY -----

-----BEGIN CERTIFICATE -----
.......                                  User - Cert
-----END CERTIFICATE -----
```

Listing 4: Inhalt der Proxy-File

Die Klassen `ParseSLCFile` und `ParseProxyFile` wurden implementiert, um die SLC- und Proxy-Credentials aus den jeweiligen Credential-Dateien zu extrahieren.
Beide Klassen wandeln den Inhalt der Credential-Datei in ein String-Objekt ab, um es einfacher zu parsen. Dies wird durch das folgende Listing beschrieben:

```
DataInputStream dis = new DataInputStream(new
FileInputStream(file));
byte[] Bytes = new byte[(int)file.length()];
dis.read(Bytes);
dis.close();

stringFile = new String(Bytes);
```

Listing 5: String-Konvertierung des Dateiinhalts

Wie zu erkennen ist, sind die Credentials umschlossen von einem festgelegten Anfangs- sowie End-String. Dadurch lassen sich die Credentials leicht parsen, um sie in ihre ursprüngliche Credentialform umzuwandeln. Folgender Programm-Code veranschaulicht dies am Beispiel eines X.509-Zertifikats:

```
ByteArrayOutputStream bosCert = new ByteArrayOutputStream();
bosCert.write(userCert.getBytes());
bosCert.close();
ByteArrayInputStream bisCert = new
ByteArrayInputStream(bosCert.toByteArray());
uCert[0] = (X509Certificate)certificatefactory.
generateCertificate(bisCert);
```

Listing 6: Parsen eines X.509-Zertifikats

Im Unterschied zur abstrakten Klasse speichern die beiden konkreten Klassen die privaten Schlüssel in einem separaten Array ab. Diese Methode wurde wie folgt definiert:

```
# getPrivateKeys():PrivateKey[]
```

Die Verarbeitung der Credentials wird im nächsten Unterabschnitt beschrieben.

6.1.2 Erstellung sowie Erweiterung einer UNICORE-Benutzeridentität

Der UNICORE Client wurde im Rahmen der Masterarbeit, um ein SLC- sowie Proxy-Plugin erweitert:

Das SLC-Plugin ist für das Erstellen einer UNICORE-Benutzeridentität zuständig.

Der Benutzer verwendet das Plugin, wenn es sich bei seinem Benutzer-Credential nicht um ein Benutzerzertifikat handelt. Das SLC-Plugin wandelt die SLC-Credentials, die er von der `ParseSLCFile`-Klasse erhält, in einen UNICORE-Keystore um.
Beim Ausführen des SLCPlugin, wird der Benutzer zunächst aufgefordert, die Lokalität zu seiner SLCFile anzugeben. Diese wird im Textfeld-Objekt `SLCDirTextField` abgespeichert. Der folgende Programm-Code veranschaulicht die Anweisungen, die ausgeführt werden, um aus den Credentials, die sich im SLCFile befinden, einen UNICORE-Keystore zu erstellen:

```
ParseSLCFile cred = new ParseSLCFile();
cred.setCredentialFile(new File(SLCDirTextField.getText()));
cred.parseCredentialFile();
X509Certificate [] chain = (X509Certificate)
cred.getCredentials();
PrivateKey [] keys = cred.getPrivateKeys();
String alias = "SLC-UNICORE-IDENTITY";


ResourceManager.getKeystoreManager().addKeyEntry(keys[0],
chain,alias, false);
```

Listing 7: Erstellung eines UNICORE-Keystores

Standardmäßig speichert der UNICORE-Client die Keystore-Instanzen persistent in einer `keystore` Datei und legt sie im Home-Verzeichnis des Benutzers unter `.unicore` ab. Diese wird beim nächsten Neustarten des UNICORE-Clients geladen. Um die persistente Abspeicherung auch für den vom Plugin erstellten UNICORE-Keystore zu gewährleisten, muss folgender Programm-Code durchgeführt werden.

```
ResourceManager.getUserDefaults().setDefaultIdentity(alias);
ResourceManager.getUserDefaults().writeToFile();
```

Listing 8: Persistente Speicherung des Keystore

Im UNICORE-Keystore können mehrere Benutzer-Identitäten gespeichert werden. Deswegen muss die Methode `setDefaultIdentity()` ausgeführt werden, damit die neu erstellte Keystore als Standard-Identität gesetzt wird. Die Methode `writeToFile()` speichert die Keystore-Instanz in die oben beschriebene Datei ab.

Durch die bisher beschriebenen Programm-Codes wird noch nicht gewährleistet, dass der neu erstellte Keystore für die aktuelle Sitzung geladen wird. Dafür muss der Keystore zunächst lokal in einer Datei abgespeichert werden, um anschließend mit der

Methode `ResourceManager.loadKeystore(keystorefile, password)` für die aktuelle Sitzung geladen zu werden.

Der Benutzer verwendet das Proxy-Plugin, um seine UNICORE-Benutzeridentität um Attribut-Credentials zu erweitern. Diese müssen in einem Proxy zusammengefasst sein, damit UNICORE sie verifizieren kann. Dadurch kann UNICORE sicherstellen, dass die Attribut-Credentials eindeutig dem Benutzer zuzuordnen sind.
Das Proxy-Plugin erhält das Benutzerzertifikat und das Proxy-Zertifikat über die Methode `getCredentials()` von der `ParseProxyFile`-Klasse. Diese speichert er im Zertifiktas-Array chain ab. Das Benutzerzertifikat, das an der Stelle `chain[0]` abgelegt ist, muss mit seinem Benutzerzertifikat, das für die Erstellung seiner UNICORE-Identität verwendet wurde, übereinstimmen. Wenn dies der Fall ist, erhält das Proxy-Plugin durch den folgenden Programm-Code das Benutzerzertifikat:

```
ParseProxyFile cred = new ParseProxyFile();
cred.setCredentialFile(new File(ProxyDirTextField.getText()));
cred.parseCredentialFile();
X509Certificate [] chain = (X509Certificate)
cred.getCredentials();

KeyStore keyStore =
ResourceManager.getKeystoreManager().getSSLKeystore(chain[0]);

String alias =
ResourceManager.getKeystoreManager().getKeyAliasFromCertificate
(chain[0]);
```

Listing 9: Erhalten des Benutzer-Keystores

An der Methode `getSSLKeystore()` wird das Benutzerzertifikat, das sich im `chain[0]` befindet, als Parameter weitergegeben. Falls der Wert der Variable keyStore nicht null entspricht, erhält man dadurch den UNICORE-Benutzer-Keystore und stellt sicher, dass das Benutzerzertifikat auch für die UNICORE-Identitätserstellung verwendet wurde. Den Alias-Namen des Benutzer-Keystores erhält man durch die Methode `getKeyAliasFromCertificate`().
Bevor die Benutzer-Identität mit dem Proxy-Zertifikat erweitert wird, muss die Signatur des Proxy-Zertifikats mit dem öffentlichen Schlüssel des Benutzerzertifikats verifiziert werden. Anschließend kann erst die *certificate chain* (Zertifizierungskette) des Benutzer-Keystores erhalten und mit dem Proxy-Zertifikat erweitert werden.
Die `RessouceManager`-Klasse bietet keine Methode an, um die Zertifizierungskette eines bestehenden Keystores zu erweitern. Dadurch muss man den privaten Schlüssel

des Benutzer-Keystores entnehmen, um denselben Keystore erweitert mit dem Proxy-Zertifikat neu erstellen zu können. Dies wird mit dem unteren Listing verdeutlicht:

```
JFrame frame = new JFrame();
PasswordDialog passwordDialog = new PasswordDialog(
      (JFrame) frame, res.getString("PASSWORD_TITLE"));

passwordDialog.setStatusMessage(res.getString("ENTER_PASSWORD
"));

passwordDialog.show();
char[] passwd = passwordDialog.getPassword();

PrivateKey userkey = (PrivateKey) keyStore.getKey(alias,
passwd);

ResourceManager.getKeystoreManager().removeKeyEntry(alias);

ResourceManager.getKeystoreManager().addKeyEntry(userkey,
chain, alias, false);
```

Listing 10: Erhalten des Benutzer-Keystores

Das Entnehmen des privaten Schlüssels erfordert, dass die Eingabe eines Keystore-Passwortes. Hierfür wird das Dialog-Fenster `PasswordDialog` geöffnet. Das vom Benutzer erhaltene Keystore-Passwort wird in einem Char-Array gespeichert. Das Char-Array und der Alias-String werden als Parameter an die Methode `getKey()` weitergegeben, um den privaten Schlüssel des Benutzerzertifikats zu erhalten. Mit der Methode `removeKeyEntry()` wird die alte UNICORE-Benutzeridentität gelöscht und mit der Methode `addKeyEntry()` wird eine neue mit der Proxy-Zertifkatserweiterung erstellt.

Der erstellte Keystore muss zum Schluß wie beim SLC-Plugin persistent abgespeichert sowie neu geladen werden.

6.2 Implementierung der UUDBVO

Die UUDBVO ist für die Autorisierung der VO-Benutzer zuständig. Die Implementierung der UUDBVO besteht aus fünf Bereichen:

- Bereitstellung einer Autorisierung durch Attribute
- Administrieren der Attributliste
- Verifizieren und Validieren der Credentials
- Extrahieren der Attribute
- UNICORE-PDP

Die UUDBVO muss eine Autorisierung durch Attribute ermöglichen können. Dieses Vorhaben soll durch eine HashMap realisiert werden. Es müssen Klassen implementiert werden, die das Verwalten der HashMap verwirklichen. Ein Administrator muss durch bereitgestellte Skripte eine Attributliste verwalten können, in der er die notwendigen Attribute für eine Autorisierung bestimmen kann. Darüber hinaus müssen die Credentials der Benutzer validiert und verifiziert werden, um die Attribute der Benutzer für vertrauenswürdig erklären zu können. Es muss möglich sein, die Campus- und/oder VO-Attribute des Benutzers aus den Credentials zu extrahieren und zu interpretieren. Schließlich muss noch ein PDP implementiert werden, der anhand der Benutzer-Attribute eine Autorisierungsentscheidung trifft. Die folgenden Unterabschnitte beschreiben die Implementierung der fünf UUDBVO-Bereiche.

6.2.1 Bereitstellung einer Autorisierung durch Attribute

Die Autorisierung der alten UUDB erfolgte durch das Benutzerzertifikat. Bei einer attributbasierten Autorisierung spielt das Benutzerzertifikat aber keine Rolle mehr. Die Struktur der UUDBVO muss das Abbilden von Attributen auf Accounts gewährleisten. Sie muss das Administrieren einer Attributliste ermöglichen. Anhand dieser soll dann von der UUDBVO eine Autorisierungsentscheidung getroffen werden. Die Attributliste wird durch eine HashMap realisiert. Es müssen wiederum Funktionalitäten bereitgestellt werden, die die HashMap-Einträge persistent speichern und laden können. Die nächsten Unterabschnitte beschreiben die Klasse UUDBVO, in der eine HashMap implementiert wurde. Später gehen sie auf die Speicherung und das Laden der HashMap-Einträge ein. Schließlich wird noch auf Java-Klassen eingegangen, die für das Verwalten, die Auflistung und Erstellung bzw. Aktualisierung der HashMap-Einträge zuständig sind.

Die UUDBVO-HashMap

Die Attributliste soll durch eine HashMap realisiert werden. Die HashMap kann Objekte im Arbeitsspeicher verwalten. Die Eigenschaften eines HashMap-Objektes sind:

- Der Eintrag einer HashMap besteht aus dem Tupel (Objektschlüssel, Objektwert).
- Jeder Objektschlüssel kann nur einmal in der HashMap vorhanden sein.
- Jeder Objektschlüssel kann nur einem Wertobjekt zugeordnet sein.

Die HashMap-Einträge werden unsortiert gespeichert, sind aber über den Schlüssel schnell verfügbar. Ein weiterer Grund, eine HashMap zu nutzen ist, dass die alte identitätsbasierte UUDB sie ebenfalls verwendet. Es wäre daher möglich, dass UNICORE eine identitäts- sowie attributbasierte Autorisierung unterstützt.
Die HashMap wurde in der Klasse `UUDBVO` implementiert. Die wichtigen Eigenschaften für das Verwalten der HashMap werden in Listing 11 dargestellt.

Die Klasse der UUDBVO erstellt in ihrem Konstruktor eine neue HashMap. In dem Listing werden die bekannten HashMap-Methoden angesprochen. Die Methode `setEntry()` speichert neue HashMap-Einträge. Ein neuer HashMap-Eintrag benutzt Attribute als Schlüssel und XLogins als Werte. Die Methode `getAttributeSet()` gibt die Menge der Attribute in Form eines Set-Objektes wieder. Die Methode `getXlogin()` gibt aufgrund des Attributschlüssels das davon assoziierte XLogin wieder. Die Methode `isInUUDB()` testet, wie auch die vorherige Methode `getXlogin()`, das Vorhandensein eines Schlüssels. Sie sendet durch die HashMap-Methode `containskey()` die boolesche Variable „true" bei einem erfolgreichen Auffinden des Attributschlüssels. Die Methode `getUUDBVOsize()` liefert die Anzahl der Schlüssel/Wert-Paare.

```
	public void setEntry(String attributes, String value){
		database.put(attributes, value);
	}

	public Set getAttributeSet(){
		return database.keySet();
	}

	public String getXlogin(String attribute){
		return (String) database.get(attribute);
	}


	public boolean isInUUDB(String attributes){
		return database.containsKey(attributes);
	}

	public int getUUDBVOsize(){
		return database.size();
	}
```

Listing 11: Eigenschaften der VO-HashMap

Speichern und Laden der HashMap

Weiterhin muss eine persistente Speicherung der HashMap gewährleistet werden. Dafür wird noch in der Klasse `UUDBVO` die Möglichkeit des Ladens und des Speicherns der HashMap realisiert.

Die Methode für das Laden der HashMap sieht wie folgt aus:

```
public void load(ObjectInputStream in) throws ClassNotFoundException,
IOException {
			Map new_database = new HashMap();
			int size = in.readInt();

			for(int i = 0; i < size; i++) {
				String xlogin = (String) in.readObject();
				String attributes = (String) in.readObject();
				new_database.put(attributes,xlogin);
			}
			database = new_database;
	}
```

Listing 12: Laden der HashMap

Durch die Methode `load()` kann ein serialisiertes Objekt ausgelesen werden. Die Deserialisation ermöglicht, Attribut- und XLogin-Einträge wieder in die HashMap einzufügen. Für das Laden muss die HashMap vorher in eine Datei in serialisierter Form gespeichert werden. Das persistente Speichern der HashMap in einer Datei wird

durch das Weiterleiten von HashMap-Objekten in einen Objekt-Ausgabe-Strom (ObjectOutputStream) und dann in einen Datei-Ausgabe-Strom (FileOutputStream) realisiert. Schließlich werden die Objekte in einem File gespeichert.
Listing 13 beschreibt die Möglichkeit des Auslesens aus einer Datei.
Die Vorgänge des Einlesens können als eine Spiegelung des Speichers in einer Datei gesehen werden.
Die Objekte werden aus einer Datei geladen, dann werden sie von einem Datei-Eingabe-Strom (FileInputStream) in einen Objekt-Eingabe-Strom (ObjectInputStream) weitergeleitet. Nach der Instanzierung der `UUDBVO` wird die Methode `load()` mit dem Objekt-Eingabe-Strom als Parameter angesprochen.

```
fis = new FileInputStream(uudb_file);
ois = new ObjectInputStream(fis);
uudb_table = new com.fujitsu.arcon.uudb.UUDB_VO();
uudb_table.load(ois);
```

Listing 13: Ausführen der Methode load() mit einem ObjectInputStream

Auflistung und Aktualisierung der UUDBVO

Für das Verwalten der HashMap wurden noch zwei weitere Klassen implementiert. Für die Auflistung ist die Klasse `ListUUDBVO` zuständig. Für die Aktualisierung der HashMap wird die Klasse `MakeUUDBVO` angesprochen. Diese Klassen können dann für den Administrator mit Shell-Skripten aufrufbar gemacht werden.

Bei der Auflistung der Attributliste kann der Administrator mit dem Shellskript „list_attributes“ die Javaklasse `ListUUDBVO` ansprechen. In dem Skript ist zusätzlich als Argument noch die serialisierte Datei, in der die HashMap-Paare eingetragen sind, aufgeführt. Die Klasse führt bei einem erfolgreichen Entnehmen der serialisierten Datei die Methode `load()` der UUDBVO, wie in dem Abschnitt des persistenten Speichers der HashMap beschrieben aus. Nach der Methode existiert so eine HashMap, die die HashMap-Paare der Attributliste besitzt. Mit der `AttributeSet()`-Methode können die Attributschlüssel der HashMap entnommen werden und diese werden dann iterativ mit ihrem Wert ausgegeben.

Die Klasse `MakeUUDBVO` kann vom Administrator durch das Shell-Skript „`makedb_attributes`“ gestartet werden. Die Klasse entnimmt alle Attribute und füllt die Attribute und XLogins / Rules in die HashMap. Danach wird die HashMap serialisiert und in einer Datei gespeichert.

6.2.2 Administration der Attributliste

Das Administrieren der Attributliste ist für den UUDBVO-PDP entscheidend, denn durch sie werden die benötigten Attribute der Benutzer für einen Zugang zu UNICORE-Ressourcen festgelegt. Erst nach einer erfolgreichen Autorisierung ist eine Submittierung der Jobs möglich. Der Administrator ist für das Verwalten der Attributliste zuständig. Dafür wurden vier Shell-Skripte implementiert und dem Administrator zur Verfügung gestellt. Die Skripte realisieren das Erstellen, das Löschen, das Auflisten und das Aktualisieren der Attributliste. Die Skripte „`makedb_attributes`" und „`list_attributes`" wurden schon im vorherigen Unterabschnitte beschrieben. Das wichtigste Skript für den Administrator ist „`add_attributes`". In diesem Skript kann er Attribute und Regeln definieren, die ein Benutzer für eine Autorisierung aufweisen muss. Das Shell-Skript bietet die Möglichkeit sofort ein Attribut und XLogin beim Aufruf des Skriptes einzugeben. Auch ist es möglich, dem Administrator beim Einfügen eines Attributes inklusiver der Regeln zu unterstützen. Die Attributliste soll VO- und Campusattribute verwalten können. Zusätzlich muss noch das Abbilden von benötigten Attributen auf ein XLogin (Unix-Login) gewährleistet werden. Bei einer erfolgreichen Autorisierung werden die XLogins dem Benutzer zugewiesen. Unter dem UNIX-Account kann der Benutzer UNICORE-Ressourcen ausführen. In der Attributliste können

1. benötigte VO-Attribute,
2. benötigte Campusattribute,
3. VO-Attribute und zusätzlich noch Campusattribute

definiert werden.

Definieren von benötigten VO-Attribute

Die UUDBVO erlaubt es, VO-Attribute wie z.B. FQAN in der Attributliste verwalten zu können. FQANs besitzen eine 3-Tupel -Form:

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

In FQANs werden die VOs, die Gruppen, Rollen und die Fähigkeiten der VO-Benutzer definiert. In der Attributliste kann dann ein bestimmter FQAN als eine Bedingung für den Zugang zu einer Ressource definiert werden. Ein Eintrag in der Attributliste könnte dann folgende Struktur aufweisen:

```
/VO-IVOM/Role=Student/Capability=NULL      student01
```

Falls der VO-Benutzer einen solchen FQAN besitzt, wird dem Benutzer der UNIX-Account `student01` zugewiesen und er ist erfolgreich autorisiert. Diese Form der Autorisierung regelt den Zugriff auf UNICORE-Ressourcen nur anhand von VO-Attributen.

Momentan existieren nur zentrale VO-Management-Systeme wie VOMS und MyVocs. Die zentralen VO-Systeme erweitern die Benutzer-Credentials um VO-Attribute. Denkbar wäre aber auch ein dezentraler Ansatz, bei dem VO-Attribute vom Identity Provider des Benutzers verwaltet werden. Somit werden Campus- und VO-Attribute von einer Instanz erstellt und der UUDBVO weitergegeben. Für die Implementierung der UUDBVO wurde auch ein solcher Ansatz mitberücksichtigt und stellt für das Verwalten der Attributliste kein Problem dar. In diesem Fall würde das Erstellen der benötigten VO-Attribute in der Attributliste äquivalent zur Erstellung der Campusattribute sein.

Definieren von benötigten Campusattributen

Campusattribute werden vom Identity Provider des Benutzers ausgegeben. Die Attribute (z.B. Name, Telefonnummer, E-Mail) identifizieren und beschreiben einen Benutzer. Als Campusattribute werden aber die Rolle oder Zugehörigkeit des Benutzers in seinem Unternehmen, Institut oder seiner Universität verstanden. Bei der attributbasierten Autorisierung mit UNICORE kann es auch Benutzer geben, die keine VO-Attribute besitzen. Benutzer, die die Infrastruktur von Shibboleth benutzen, müssen nicht notwendigerweise über ein VO-Management-System verfügen. Die UUDBVO kann auch für solche Benutzer den Zugriff auf UNICORE-Ressourcen gewährleisten. Ihre Autorisierung geschieht ausschließlich über deren Campusattribute. Das Verwalten der benötigten Campusattribute kann eine derartige Struktur aufweisen:

```
urn:mace:dir:attribute-
def:eduPersonScopedAffiliation=Student@fh-koeln.de stud01
```

Ein Campusattribut ist der `ScopedAffiliation` vom EduPerson-Schema. In dem obigen Beispiel werden nur Studenten für die Autorisierung berücksichtigt. Das Attribut `eduPersonScopedAffiliation` bietet sich für eine Autorisierung an, da sie eine Gruppe von Identitäten in sich tragen könnte. Mit der Bezeichnung „`Scope`" kann noch das Umfeld des Attributs definiert werden. In dem Beispiel wird die Fachhochschule Köln als Scope definiert. Eine attributbasierte Autorisierung anhand von Telefonnummer oder Name macht weniger Sinn, da sie demnach der identitätsbasierten Autorisierung mitsamt ihren Problemen entsprechen würde.
Für eine Autorisierung kann es Szenarien geben, die mehr als ein Attribut für die Zugangentscheidung benötigen. Ein Beispiel dafür wäre, dass nur die Studenten der FH Köln des 1. Semesters, die aber nicht Maschinenbau studieren, Zugang zu UNICORE-Ressourcen haben dürfen. Die Implementierung der UUDBVO bietet auch solch eine Autorisierung an. Für die Form kann der Administrator zu einem Hauptattribut Regeln definieren. In der Attributliste müsste der Administrator dann folgende Zeile definieren:

```
urn:mace:dir:attribute-def:eduPersonScopedAffiliation=
Student@fh-koeln.de @@-Studiengang=Maschinen-Bau@@+Semester=1
%studs1
```

Nachdem das Hauptattribut definiert wurde, ist eine Regel definiert worden. Regeln können zusätzlich durch benötigte, aber auch nicht gestattete Attribute definiert werden. Die nicht gestatteten Attribute werden mit dem Vorzeichen „-", die benötigten Attribute mit dem Vorzeichen „+" definiert. Nachdem Definieren von zusätzlichen Regeln wird der XLogin eingegeben. Die Trennung von der Regel zu einem XLogin wird durch das Zeichen „%" realisiert. Regeln beginnen mit einem „@@" Zeichen und sind so von XLogins unterscheidbar.

Definieren von VO-Attributen mit Campusattributen

Wenn VO-Attribute der Benutzer für eine Autorisierung nicht genügen, können in der UUDBVO-Implementierung zusätzlich Campusattribute hinzugezogen werden. Ein Beispiel für solch eine Definition wäre:

```
/VO-IVOM/Role=Student/Capability=NULL       @@-
urn:mace:dir:attribute-def:eduPersonPrincipalName=Max
Mustermann%student01
```

Das Beispiel beschreibt, dass dem Studenten Max Mustermann der Zugriff auf die UNICORE-Ressource verweigert wird. Allen anderen Studenten der VO „IVOM“ wird der Zugang gewährt.

6.2.3 Verifizieren und Validieren der Credentials

Bevor anhand der Attributte des Benutzers eine Autorisierungsentscheidung durchgeführt werden kann, müssen die verschiedenen Credentialformen verifiziert und validiert werden. Diese Überprüfung wird für die folgenden Credentials angewendet:

Benutzerzertifikat:

Im Benutzerzertifikat können sich Autorisierungsinformationen befinden. Deswegen muss das Benutzerzertifikat außer der Verifizierung sowie Validierung auch nach Attribut-Credential-Erweiterung überprüft werden.

Für die Verifizierung benötigt die UUDBVO das CA-Zertifikat, welches das Benutzerzertifikat signiert hat. In der `njs.properties`-Datei müssen die Lokalitäten der notwendigen CA-Zertifikate eingetragen werden. Dies geschieht durch die folgende Struktur:

```
njs.properties
…
#
# Signing authority certificates
# ":" separated list of file names
#
njs.unicore_ca_loc=/home/user/UNICORE/cert/CA_Root_cert.pem:/home/u
ser/UNICORE/cert/VOMS-AttributeIssuerCA.pem
```

Listing 14: njs.properties

Um an diese CA-Zertifikaten zu gelangen, die im `njs-properties`-File stehen, wurde in der Klasse `DetailsUUDB_VO` eine Methode implementiert, die wie folgt definiert ist.

```
getCertificateIssuer(String issuer):X509Certificate
```

Die Methode benötigt das DN-Subjekt des gesuchten Zertifikats als String. Wenn eines der eingetragenen Zertifikate dieselbe DN besitzt, wird dieses Zertifikat als Rückgabewert zurückgegeben. Das untere Listing beschreibt die Verifizierung und die Validierung des Benutzerzertifikats:

```
String issuer = usercert.getIssuerDN().toString();
X509Certificate issuerCertificate = getCertificateIssuer(issuer);
PublicKey puKey = issuerCertificate.getPublicKey();
verifyCert(usercert, puKey);
validateCert(usercert);

public void verifyCert(X509Certificate cert , PublicKey puKey){
   try{
      cert.verify(puKey);
      }catch(CertificateException e1){
      …
       }
}
public void validateCert(X509Certificate cert){
   try{
      cert.checkValidity();
      }catch(CertificateException e2){
      …
       }
}
```

Listing 15: Verifizierung und Valdierung des Benutzerzertifikats

Falls das CA-Zertifikat des Benuzers durch die oben beschriebene Methode gefunden wurde, muss daraus der öffentliche Schlüssel entnommen werden, um die Signatur des Benutzerzertifikats damit zu verifizieren. Die Validierungsmethode überprüft die Gültigkeitsdauer des Benutzerzertifikats.

Proxy-Zertifikat

Der Benutzer ist selbst der Aussteller des Proxy-Zertifikats. Dadurch erübrigt sich die Methode `getCertificateIssuer()` ausgeführt werden. Der öffentliche Schlüssel wird aus dem Benutzerzertifikat extrtahiert und für das Verifizieren des Proxy-Zertifikats verwendet. Die in Listing 15 beschriebenen Methoden werden auch für das Verifizieren und Validieren des Proxy-Zertifikats eingesetzt.

Attribute-Credentials

Die Attribut-Credentials können sich im Benutzerzertifikat und/oder in den Proxy-Zertifikaten befinden. Falls Attribute-Credentials im Benutzerzertifikat abgelegt sind, ist das Verifizieren und Validieren dieser nicht zwingend erforderlich. Da der Aussteller des Benutzerzertifikats in der Regel diese Überprüfung durchführt, bevor er sie als X509-Extension im Zertifikat speichert.

Wenn sich die Attribut-Credentials in einem Proxy-Zertifikat befinden, ist das Verifizieren und Validieren dieser Credentials notwendig. Das UNICORE-System muss sicherstellen, dass die Attribute-Credentials von einer vertrauenswürdigen Instanz signiert wurden. Die CA-Zertifikate der Attribute-Credentials können, wie bei den Benutzerzertifikaten beschrieben, über einen Eintrag in der `njs.properties`-Datei dem UNICORE-System bekannt gemacht werden und über die Methode `getCertificateIssuer()` für das Verifizieren verwendet werden.

6.2.4 Extrahieren der Attribut-Credentials

Ein Attribut-Credential fasst eine Menge von Benutzerattributen zusammen. Attribute können anhand ihrer Typen unterteilt werden in:

- Campusattribute
- VO-Attribute

Der Unterschied zwischen VO- und Attribut-Credentials liegt darin, dass die Attribute der VO-Credentials nur VO-spezifische Autorisierungsinformationen beinhalten. Attribut-Credential können dagegen Campusattribute besitzen, die keine Angaben über Autorisierung oder eine VO liefern. Da Autorisierungsentscheidungen auch durch die Kombination von Campus- und VO-Attributen getroffen werden können, muss es möglich sein, Attribute aus den Attribute-Credentials extrahieren zu können.

Obwohl für Attribut-Credentials unterschiedliche Credential-Typen zum Einsatz kommen können, lassen sich für das UNICORE-System einige Eigenschaften und Methoden verallgemeinern, die bei jedem Credntial-Typ benötigt werden. Dies wird mit der abstrakten Klasse `AttributeExtraction` beschrieben (siehe Listing 16).

Um Attribut-Credentials von anderen unterscheiden zu können, müssen folgende Eigenschaften beschrieben werden:

- **Service-Name:**
  Jedes Attribute-Credential wird von einem bestimmten Attribute-Credential-Service erstellt. Ein Beispiel hierfür ist der GridShibSAML.

- **Credential-Typ:**
  Die Credentials können unterschiedlichen Typs sein. Sowie zum Beispiel Attributzertifikate und SAML-Assertions.

- **Attribut-Typ:**
  Für die PDP-Implementierung ist der Attribut-Typ von besonderer Wichtigkeit. Wie oben beschrieben, gibt es zwei verschiedene Attribut-Typen: Campus- und VO-Attribute.

```
public abstract class AttributeExtraction {

  String serviceName;
  String credentialType
  String attributeType;
  X509Certificate certificate;
  String OID;
  Object ac;                  // for attribute credential
  Object issuerCredential;
  List attributeList;



  public abstract void setCertificate(X509Certificate certificate);

  public abstract boolean checkForExtension();

  public abstract void setAttributeIssuerCredential(Object
issuerCredential);

  public abstract X509Certificate getCertificate();


  public abstract void extractAttributeCredential();

  public abstract Object getAttributeCredential();

  public abstract void validateAttributeCredential();

  public abstract void verifyAttributeCredential();

  public abstract String getAttributeIssuerName();

  public abstract List getAttributes();
```

Listing 16: AttributeExtraction.java

Durch das UNICORE-System können die Attribut-Credentials nur in Zertifikaten als Extensions gespeichert werden. Daher wird für die Attribut-Extraktion Folgendes benötigt:

- **Das X509-Zertifikat**:
  Das Zertifikat, in dem sich die Attribut-Credentials befinden können.

- **Object Identifier (OID):**
  Eine eindeutige OID sorgt dafür, dass man die Attribut-Credentials einen Attribute-Credential-Service zuordnen kann. Die Attribut-Credentials werden mit diesem OID in der Zertifikats-Extension gespeichert.

Um zu gewährleisten, dass UNICORE den Attributen im Attribut-Credential vertraut, benötigt UNICORE das Issuer-Credential, das das Attribut-Credential signiert hat. Die Attribute sollen in einer Liste gespeichert werden, da die Anzahl der Attribute nicht bekannt ist und sich diese ändern kann.

Im Allgemeinen sind folgende Methoden für eine Attribut-Extraktion notwendig:

- **Zertifikat nach Attribut-Credential-Erweiterungen überprüfen:**
  Es muss eine Methode implementiert werden, die überprüft, ob ein bestimmtes Attribute-Credential in die Zertifikats-Extension eingefügt wurde.

- **Extrahieren des Attribut-Credentials:**
  Wenn im X509-Zertifikat eine Attribut-Credential-Erweiterung gefunden wurde, kann das Attribut-Credential aus dem Zertifikat extrahiert werden.

- **Validieren und Verifizieren des Attribut-Credentials:**
  Es ist notwendig, dass nicht nur das Zertifikat validiert und verifiziert wird, sondern das extrahierte Attribute-Credential muss auch mit den beiden Methoden überprüft werden.

- **Erhalten der Attribute:**
  Nachdem durch die Validierung und Verifizierung des Attribut-Credentials festgestellt wurde, dass man dem Credential vertrauen kann, können die Attribute des Credentials extrahiert werden.

Durch die Konzeption wird deutlich, dass UNICORE zwei Attribut-Credential-Typen interpretieren muss:

- Attributzertifikate

- SAML-Assertions

Die VOMS-Attribute werden in Attributzertifikaten gespeichert. GridShib-SAML Tools-Service verwendet SAML-Assertions als Attribute-Credential-Typ. Dadurch wurden bei dieser Arbeit für die attributbasierte Autorisierung die Klassen `VOMSAttributeExtraction` und `GridShibSAMLAttributeExtraction` implementiert, die die abstrakte Klasse `AttributeExtraction` konkretisieren.

Extrahieren der VOMS-Attribute

Das Attributzertifikat ist in VOMS gespeichert und zwar im Proxy-Zertifikat des Benutzers. Das untere Listing veranschaulicht das Prinzip eines VOMS-Proxys:

```
> voms-proxy-info -all

Subject:    /C=DE/O=FHG/OU=PCert/L=KOELN/CN=Mustermann/CN=proxy
Issuer:     /C=DE/O=FHG/OU=PCert/L=KOELN/CN=Mustermann
identity:   /C=DE/O=FHG/OU=PCert/L=KOELN/CN=Mustermann
…
```

Listing 17: VOMS-Proxy-Prinzip

Das VOMS-Proxy-Zertifikat besitzt das DN-Subjekt des Benutzerzertifikats sowie den Eintrag „`/CN=proxy`" als sein eigenes DN-Subjekt. Das Benutzerzertifikat ist gleichzeitig der Herausgeber und die Identität des Proxy-Zertifikats. VOMS hat eine weltweit eindeutige Nummer (OID) "`1.3.6.1.5.3004.100.100`". Alle weiteren Spezialisierungen, wie zum Beispiel "`1.3.6.1.5.3004.100.100.`**`1`**", dürfen nur vom VOMS-System verwendet werden.

In ein Proxy-Zertifikat werden die VOMS-Informationen als Extensions eingebettet, die durch die OID „**`1.3.6.1.5.3004.100.100.5`**" gekennzeichnet werden. Durch diese OID können das VOMS-System und auch andere Grid-Systeme feststellen, ob das Credential über VOMS-Attribute verfügt. Ein VOMS-Attributzertifikat wurde mit der Nummer „`1.3.6.1.4.1.8005.100.100.4`" eindeutig gekennzeichnet. Die untere Abbildung veranschaulicht ein Proxy-Zertifikat, das mit VOMS-Informationen erweitert wurde:

```
Certificate:
   Data:
      Version: 3 (0x2)
      Serial Number: 14 (0xe)
      Signature Algorithm: md5WithRSAEncryption
      Issuer:
      Validity
      Subject:
Exponent: 65537 (0x10001)
      X509v3 extensions:
         1.3.6.1.4.1.8005.100.100.5:
            0...0...0...0......0X.V0Q.O0M1.0...U....DE1.0
..U....NRW1.0...U.
0...U....user....h0f.d0b1.0...U....DE1.0
..U....NRW1.0...U.
.......L:.0"..20070705092825Z..20070705212825Z0..0...
+.....Edd.1..0.....vo://kappes:150010h.
      /vo/Role=Rechnend/Capability=NULL../vo/Role=NULL/Capability=NULL.%/vo/IVOM/Role=NULL/Capability=
      NULL0,0...U.8....0.............o....p.YJ..4O.&..[......_.&N..5..*........h?5.....3Y..bnRtz..T.2.t
.s.....N).GX..|.....}..5..R.M{m.....f....0..C.._...q......._......0..s.3.>....:9l......1.#.....B.z....y..SR.[......a?..].y..X............-
      |......r..2X.+....*.........9....0....ihY....|
....
```

Abb.26: Proxy-Zertifikat mit VOMS-Extension

Wie zu erkennen ist, sind die VOMS-Informationen nicht als String-Erweiterungen im Proxy-Zertifikat eingefügt. Der Extension-Eintrag muss dadurch wieder zu seinem ursprünglichen Credential-Typ verarbeitet werden. Durch das Bouncy Castle[17]-Paket kann man in der Programmiersprache Java von X.509-Zertifikaten (`X509Certificate.java`) mit der Methode

```
getExtensionValue(java.lang.String OID):byte []
```

die Extensions erhalten. Da VOMS Attributzertifikate als VO-Credentials verwendet, muss das dadurch erhaltene Byte-Array in ein Attributs-Zertifikats-Objekt umgewandelt werden.

Im RFC 3281 [24] wurde ein Profil für Attributzertifikate definiert, die für Sicherheitsanwendungen und Umgebungen zum Einsatz kommen können. Dieses Profil wurde vom Bouncy Castle's (BC) Krypto-Paket verwendet, um X.509-Attributzertifikatsklassen mit deren Unterklassen zu definieren.

Ein Attributs-Zertifikat besteht außer den Attributen aus Feldern, wie Version, Besitzer, Herausgeber, Signatur, Serial-Nummer, Gültigkeitsdauer, ID des Herausgebers sowie Erweiterungen. Das folgende Listing veranschaulicht die zwei wesentlichen Felder:

[17] Bouncy Castle (vollständig Bouncy Castle Crypto API) ist ein für JAVA-Plattformen entwickeltes Open-Source Kryptographie-Paket

```
AttributeCertificate ::= SEQUENCE {
      acinfo                  AttributeCertificateInfo,
      signatureAlogrithm      AlgorithmIdentifie,
      signatureValue          BIT STRING
}

AttributeCertificateInfo ::= SEQUENCE {
      version                 AttCertVersion -version is v2,
      holder                  Holder,
      issuer                  AttCertIssuer,
      serialNumber            CertificateSerialNumber,
      attrCertValidityPeriod  AttCertValidityPeriod,
      attributes              SEQUENCE OF Attribute,
      issuerUniqueID          UniqueIdentifier OPTIONAL,
      extensions              Extensions OPTIONAL
}
```

Listing 18: AC-Profil nach RFC 3281

In Bouncy-Castle entsprechen die mit Grossbuchstaben beginnenden Begriffe den Java-Klassen. Die Bouncy-Castle-Implementierung für X.509-Attributzertifikate sind nach bisherigen Erfahrungen (BC-Version 1.37) noch nicht vollständig funktionsfähig. Die VOMS-Entwickler haben eine *Shadow-Implementierung* der BCs-Attributzertifikatsklassen entwickelt, um Attributzertifikate für das VOMS-System einzusetzen. Diese entsprechen zum großen Teil dem Profil des im RFC beschriebenen Attributzertifikats. Geringe Modifikationen wurden in den Feldern Holder, AttCertIssuer, V2Form, Attributes und Extensions durchgeführt [CIA04]. Die Attributs-Zertifikats-Implementierung wurde im *VOMS AC-JAVA API* (org.glite.security.voms.ac) hinzugefügt.

Einige Grid-Projekte, wie zum Beispiel GridSite vom GridPP [MCN05] hatten Probleme, die VOMS AC APIs für eigene Implementierungen zu verwenden. Im Rahmen dieser Masterarbeit wurde das BC-Sicherheits-Paket-Version 1.37 und VOMS AC API für die Implementierung verwendet. Beide Pakete sind nicht kompatibel zueinander und erforderten eine Modifizierung, welche im folgenden Listing dargestellt wird:

```
// Ursprünglicher Code
DERConstructedSet set = (DERConstructedSet)
attribute.getObjectAt(1);

// Neuer Code
ASN1Set set = (ASN1Set) attribute.getObjectAt(1);
```

Listing 19: Modifizierung der AttributeCertificateInfo-Klasse

Die Java-Klasse AttributeCertificateInfo.java des VOMS AC APIs greift auf eine veraltete BC-Klasse zu, die in der Version 1.37 nicht mehr verwendet wird. In der

Dokumentation des BC-Packets wird darauf hingewiesen, anstatt von `DERConstructedSet` die Klasse `DERSet` zu verwenden. Der Hinweis hat nicht dazu geführt, dass die Attributzertifikat-Klassen verwendet werden kann. `ASN1Set` ist eine abstrakte Klasse, die die DERObject-Klasse erweitert. Durch die Verwendung dieser Klasse konnten Attributzertifikate aus den Zertifikats-Extensions erstellt werden.

Die konkrete Klasse `VOMSAttributeExtraction` ist für das Extrahieren der VOMS-Attribute zuständig. Die Eigenschaften wurden wie folgt definiert:

```
public class VOMSAttributeExtraction extends AttributeExtraction{


      public String serviceName = "VOMS";
      public String credentialType = "Attribute Certificate";
      public String attributeType = "VO";
      public X509Certificate certificate = null;
      public static final String OID = "1.3.6.1.4.1.8005.100.100.5";
      public X509Certificate issuerCertificate = null;
      AttributeCertificate ac = null;
      List attributeList = new ArrayList();
…
}
```

Listing 20: Eigenschaften der VOMS-Klasse

Durch die OID-Information kann überprüft werden, ob das Zertifikat mit einer VOMS-Extension erweitert wurde. Damit man an die VOMS-Attribute gelangt, muss zunächst das Attribut-Credential aus dem Zertifikat extrahiert werden. Die Methode `extractAttributeCredential()` ist für diese Extrahierung zuständig.

Eine Extension besteht aus einer OID und einer `ASN.1`[18]-Struktur. Um Daten in einer X509-Extension zu speichern, findet eine `ASN.1 DER`-Kodierung statt. Jede Extension hat einen eindeutigen Bezeichner (OID), ein Flag, das darauf hinweist, ob die Extension kritisch ist und den eigentlichen Wert der Extension (`extnValue`) [HOL05]. VOMS-Extensions sind als unkritisch definiert. Der `extnValue` ist als `OCTET STRING` definiert. Folgendes Listing beschreibt, welche Schritte notwendig sind, um an das Attributzertifikat zu gelangen:

[18] Die Abstract Syntax Notation One (ASN.1) ist eine abstrakte Beschreibungssprache. Sie ist standardisiert von ITU-T und definiert in X.680ff.

```
public void extractAttributeCredential(){
 …
 }else{

       byte[] payload = this.certificate.getExtensionValue(OID);
       // Octet String encapsulation - see RFC 3280 section 4.1
       payload = ((ASN1OctetString) new ASN1InputStream(
                    new ByteArrayInputStream(payload)).readObject())
                    .getOctets();


       ASN1Sequence acSequence = (ASN1Sequence) new ASN1InputStream(
                                  new ByteArrayInputStream(payload))
                                   .readObject();

       for (Enumeration e1 = acSequence.getObjects(); e1
            .hasMoreElements();) {

         ASN1Sequence seq2 = (ASN1Sequence) e1.nextElement();
            for (Enumeration e2 = seq2.getObjects(); e2
                .hasMoreElements();) {

               this.ac = new AttributeCertificate((ASN1Sequence)
                         e2.nextElement());
                        }
            }

        }
    …
}
```

Listing 21: extractAttributeCredential() für VOMS-Attribute

Für das Erhalten der VOMS-Attribute hat das VOMS AC API die Attributzertifikats-Klasse um die Methode `getFullyQualifiedAttributes()` erweitert. Die FQAN-Attribute werden in einer Liste zusammengefasst.

Extrahieren der GridShibSAMLAttribute

Der GridSHibSAML-Service erstellt Attribut-Credentials in Form von SAML Assertions. Die Klasse `GridShibSAMLAttributeExtraction` wurde implementiert, um diese Attribut-Credentials sowie die darin befindlichen Attribute zu extrahieren. Die Eigenschaften vom GridShibSAML-Credential wurden wie folgt definiert:

```
public class GridShibSAMLAttributeExtraction  extends
AttributeExtraction{

      public String serviceName = "GridShib-SAML";
      public String credentialType = "SAML Assertion"
      public String attributeType = "Campus";
      X509Certificate certificate = null;
      private static final String OID = "1.3.6.1.4.1.3536.1.1.1.10";
      SAMLAssertion ac = null;
      List attributeList = new ArrayList();
…
}
```

Listing 22: Eigenschaften der GridShibSAML-Klasse

Der Attribut-Typ ist im Gegensatz zu den VOMS-Attributen als Campus definiert. GridshibSAML verwendet die eindeutige OID „`1.3.6.1.4.1.3536.1.1.1.10`“. Falls eine GridShibSAML-Extension in ein Zertifikat eingebettet wurde, kann die OID verwendet werden, um an diese Erweiterung zu gelangen. Der Extension-Eintrag muss dann in seinem ursprünglichen Credential-Typ verarbeitet werden.

Für die Implementierung wurde die `Globus-OpenSAML-Bibliothek Verison 1.1` verwendet. Diese soll mindestens mit der `Sun JAVA Version 1.5` kompiliert werden, da bei `Java 1.4` Fehler beim Parsen entstehen können.

Für das Extrahieren des Attribute-Credential muss zunächst, wie bei den VOMS-Attributen schon beschrieben, der `extnValue` des Extensions entnommen werden. Diese wird dann als ByteArray gespeichert. Das folgende Listing beschreibt, welche Schritte notwendig sind, um daraus den ursprünglichen Attribut-Typ zu extrahieren:

```
try {
      ByteArrayInputStream in = new ByteArrayInputStream(payload);

        if (in != null)
          ac = new SAMLAssertion(in);

     } catch(SAMLException exp) {
        exp.printStackTrace();
     }
```

Listing 23: Eigenschaften der GridShibSAML-Klasse

Verglichen mit den VOMS-Attributen ist das Extrahieren der GridShibSAMLException ein wenig komplexer. Deswegen wird die SAML-Struktur vom GridShibSAML kurz beschrieben:

- Ein SAML-Attribut besteht immer aus einem Attributnamen und einem Attribut-Wert.
- SAML-Attribute sind gespeichert in SAMLAttributeStatements.
- SAMLAttributeStatements sind gespeichert in SAML-Assertions.
- SAML-Assertions müssen nicht zwingend SAML-AttributeStatements besitzen.
- In einer SAML-Assertion können verschiedene SAML-Assertions gespeichert sein.
- Die folgende Methode extrahiert die Campus-Attribute aus einer GridShibSAML-Assertion:

```
public List getAttributes(){
try{
if(this.ac == null){
..
}else{

    final List SAMLAttributesName = new ArrayList();
    final List SAMLAttributesValue = new ArrayList();

       for (final Iterator iter = ac.getAdvice(); iter.hasNext();)
       {
        final SAMLAssertion statement = (SAMLAssertion) iter.next();
        final SAMLAttributeStatement state =
(SAMLAttributeStatement)statement.getStatements().next();

          for (final Iterator iter2 = state.getAttributes();
          iter2.hasNext();) {
            SAMLAttribute SAMLatti = (SAMLAttribute) iter2.next();
            SAMLAttributesName.add(SAMLatti.getName().toString());
            SAMLAttributesValue.add(SAMLatti.getValues().next().
            toString());
           }
       }

      for (int i = 0; i < SAMLAttributesName.size(); i++){
       this.attributeList.add(SAMLAttributesName.get(i) +"=" +
            SAMLAttributesValue.get(i));
       }
    }
    }catch (GridShibSAMLException exception){...}
return this.attributeList;
}
```

Listing 24: getAttributes() für GridSHibSAML-Attribute

Aus einer SAMLAssertion werden zunächst die verschiedenen Assertions entnommen. In diesen Assertions wird überprüft, ob SAMLAttributeStatements vorhanden sind. Falls dies der Fall ist, werden die Attributnamen sowie die Attribut-Werte jeweils in einer ArrayListe gespeichert. Zum Schluss werden diese Listen zu einer zusammengeführt.

6.2.5 UNICORE-PDP

Der UNIOCRE-PDP soll anhand von Attributen eine Autorisierungsentscheidung treffen können. Eine Autorisierung erfolgt durch eine erfolgreiche Weitergabe des XLogins.
Von der Klasse `DetailsUUDBVO` bekommt die Klasse `PDP` die Attribute des Benutzers. Der Benutzer kann VO- und / oder Campusattribute besitzen. Ein VO-Benutzer autorisiert sich mit seinen VO-Attributen. Bei Benutzern ohne VO-Attribute soll eine Autorisierung anhand seiner Campusattribute erfolgen. Die Aufgabe der Klasse `PDP` ist es, die HashMap der Attributliste zu laden, und diese mit den Attributen des Benutzers zu vergleichen. Erst wenn der Benutzer die notwendigen Attribute, die in der Attributliste verwaltet werden, besitzt, steht einer erfolgreichen Autorisierung nichts mehr im Wege. In den Einträgen der Attributlisten können aber auch Regeln definiert werden, die eine differenziertere Autorisierungsentscheidung ermöglichen.

Listing 25 stellt den Vergleich der Benutzer-Attribute mit der Attributliste dar. Ein Benutzer, der keine VO-Attribute besitzt oder sie für die Autorisierung nicht weitergibt, wird nicht als Mitglied einer VO angesehen und besitzt daher auch keine Berechtigung auf VO-Ressourcen zuzugreifen.
Zunächst überprüft die Methode `mapAttributeToLogin()` in der Klasse `PDP`, ob der Benutzer VO-Attribute besitzt. VO-Attribute werden in der String-Variable `VOAttributes`, die Campus Attribute werden in Form einer Array-Liste in der Variable `CampusAttributes` aufbewahrt. Falls ein Benutzer VO-Attribute besitzt, so will er anhand dieser eine erfolgreiche Autorisierung zu VO-Ressourcen erhalten. Die UNICORE-PDP vergleicht das VO-Attribut des Benutzers mit der Attributliste. Der Vergleich wird durch die Methode `isInUUDBVO()` realisiert. Das VO-Attribut des Benutzers muss in der Attributliste vorhanden sein. Erst dann ist die VO und die Fähigkeit des VO-Benutzers dem UNICORE bekannt. Wenn der Vergleich scheitert, dann wird der `XLogin` mit dem Wert „`null`“ zurückgegeben und die Methode wird beendet. Falls die VO-Attribute des Benutzers bekannt sind erhält die boolesche Variable `PDPUser` den Wert „`true`“. Anschließend wird die Methode `getXlogin(VO-Attributes)`ausgeführt. Das VO-Attribut des Benutzers dient als Schlüssel der Attributliste und wird als Parameter in der Methode eingefügt. Dadurch wird in der

HashMap der Wert des Tupels (Attribut, Wert) entnommen und in Form einer String-Variablen namens „`value`“ initialisiert.
Der Value kann durch nur einen XLogin gekennzeichnet werden, er kann aber auch zusätzlich noch durch Regeln definiert sein. Ein Beispiel für einen VO-Attribut-Eintrag ohne Regel wäre:

```
FQAN      XLogin
```

Nach einem FQAN wird ein XLogin entnommen. In diesem Fall würde der Wert „`Value`“ den XLogin in Form eines Strings besitzen. In einem VO-Attribut-Eintrag könnten Campusattribute in Form Regeln beschrieben worden sein. Ein Beispiel dafür ist:

```
FQAN      @@-CampusAttribut@@+CampusAttribut%XLogin
```

Für die Methode `mapAttributeToLogin()` spielen somit die Zeichen „`@@`“, „`-`“, „`+`“ und „`%`“ eine wichtige Rolle. Durch die Methode `indexOf()` wird die Variable `Value` mit dem Zeichen „`@@`“ überprüft. Falls der Rückgabewert der Methode größer als -1 ist, wurde die Zeichenkombination und daher auch das Vorhandensein von Regeln gefunden. Mit dem Zeichen „`%`“ kann die Trennung von Regeln und XLogin erfolgen. Die Regeln werden dann mit Hilfe der Methode `getRules()` in eine ArrayListe initialisiert. Die Liste besteht aus zwei Arrays, in der nicht erlaubte sowie benötigte Attribute enthalten sind. Zuerst wird überprüft, ob der Benutzer Campusattribute besitzt, die für die Autorisierung nicht erlaubt sind. Falls also ein Attribut des Benutzers mit dem `DenyAttributes` der Regel übereinstimmt, dann wird der XLogin mit dem Wert „`null`“ zurückgegeben und die Autorisierung des Benutzers ist fehlgeschlagen. Falls benötigte Attribute (`requireAttributes`) in der Regel vorhanden sind, muss der Benutzer diese in seinen Campusattributen besitzen.

Wenn der Benutzer keine VO-Attribute besitzt, werden seine Campusattribute für die Autorisierung entnommen. Ein Attribut, welches von einer Menge Identitäten aufgewiesen werden kann, dient als Schlüssel und andere Attribute können durch Regeln noch definiert werden. Die PDP wendet für die Campus-Autorisierung den gleichen Algorithmus wie die des VO-Benutzers an. Sie sucht nur nicht nach VO-Attribut-Schlüsseln, die ein VO-Attribut verwenden. Nach einer erfolgreichen Autorisierung des Benutzers wird dem Benutzer eine Rolle zugeordnet. Die Rolle wird hierbei zunächst den Wert USER besitzen, sie kann aber auch durch Rollen z.B. Administrator erweitert werden.

```
if(VOAttributes != null){
  boolean PDPUser = uudbVO.isInUUDB(VOAttributes);
    if(PDPUser = true){
      String value= uudbVO.getXlogin(VOAttributes);
      int marker_rules=value.indexOf(marker1);
        if(marker_rules >=0){
          int marker_Xlogin=value.indexOf(marker2);
          String rules = value.substring(0, marker_Xlogin);
           List [] ruleAttributes = getRules(rules);
          List denyAttributes = ruleAttributes[0];
          List requireAttributes = ruleAttributes[1];

           for(int x=0; x<denyAttributes.size(); x++){
              for(int y=0; y <CampusAttributes.size();y++){

if(CampusAttributes.get(y).equals(denyAttributes.get(x))){
                  return Xlogin;
                  }
               }
            }

     for(int i=0; i<requireAttributes.size(); i++){

      matchAttributes = false;

      for(int y=0; y <CampusAttributes.size();y++){
       if(CampusAttributes.get(y).equals(requireAttributes.get(i))){
        matchAttributes = true;
        break;}
            }
       if(matchAttributes == false){
            break;
        }
      }

      if(requireAttributes.size() > 0 && matchAttributes == false){
        return Xlogin;
      }

      Xlogin=value.substring(marker_Xlogin +1, value.length());
  }else{
    Xlogin=value;
}
```

Listing 25: UNICORE-PDP für VO-Benutzer

7. Bewertung der UNICORE-Erweiterung

Das Kapitel bewertet die UNICORE-Erweiterung der vorliegenden Arbeit mit den Anforderungen, die in Abschnitt 4.1 aufgelistet wurden. Bei diesen handelt es sich um allgemeine Anforderungen sowie solche, die vom Benutzer, dem Service Provider und dem Administrator gestellt werden.

7.1 Bewertung der allgemeinen Anforderungen

Eine allgemeine Anforderung ist, dass *ein VO-Management-System die Kommunikation mit dem Grid-Middleware-System UNICORE gewährleisten muss*. Für das Gewähren der Kommunikation zwischen dem UNICORE-System und einer VO-Technologie wurden die zurzeit verwendeten Formen für die Vermittlung der Attributinformationen des Benutzers berücksichtigt. Die UNICORE-Erweiterung kann dadurch Attributinformationen in Form von X.509.v3-Zertifikaten wie Attributzertifikaten, Proxyzertifikaten, Benutzerzertifikaten und in Form von SAML-Assertions interpretieren und für die Autorisierung auswerten. Zusätzlich wurde eine Implementierungsstruktur realisiert, die eine einfache Erweiterung von zukünftigen Attributformen erlaubt.
Durch das Anbieten von Plugins und einer neuen UUDBVO wurde die Anforderung, den *UNICORE-Kern nicht zu verändern,* realisiert. Die neue UUDBVO kann eine *Autorisierung von UNICORE anhand von Benutzerattributen nach dem ABAC-Modell ermöglichen.* Die Benutzer-Credentials entsprechen in der UNICORE-Erweiterung auch X509.v3-Zertifikaten, daher war eine Veränderung der UNICORE-Authentifizierung nicht notwendig.
Eine andere Anforderung ist, dass ein VO-Management-System möglichst mit mehreren Grid-Middleware zusammenarbeiten soll. Die VO-Management-Systeme VOMS und MyVocs arbeiten mit Proxy-Zertifikaten. Die Grid-Middleware-Systeme gLite und GT 4 verwenden ebenfalls Proxy-Zertifikate und erlauben die Zusammenarbeit mit diesen VO-Management-Systemen. Durch die Implementierung der vorliegenden Arbeit kann nun auch UNICORE mit VO-Systemen, wie VOMS und MyVocs, kollaborieren. Somit ist es für den VO-Benutzer oder Communities möglich, die Grid-Middleware anhand ihrer Kriterien auszuwählen. Falls zukünftig eine dezentrale VO-Struktur durch Shibboleth verwendet wird, kann die UNICORE-Erweiterung auch diese unterstützen.
Die Anforderung *eine direkte Verwendung der UNICORE-Erweiterung in D-Grid zu ermöglichen*, wird gewährleistet. Bei der Verwendung von Shibboleth-Benutzer in UNICORE muss aber eine Instanz wie GridSHib-CA oder die DFN-AAI den Shibboleth-

Benutzern SLCs erstellen und zuweisen können. Erst nach der Erstellung von SLC kann ein Shibboleth-Benutzer das UNICORE-System nutzen.

7.2 Bewertung der Anforderungen der Benutzer

Eine Anforderung ist, dass die *Benutzung des VO-Management-Systems möglichst einfach sein soll*. Die Implementierung der Arbeit stellt dem VO-Benutzer in dem UNICORE-Client zwei Plugins bereit, indem er seine Credentials in Form eines SLC und/oder eines Proxyzertifikats eingeben kann. Die Anforderung, dass der Benutzer *nur einmal aufgefordert werden soll, sein Passwort einzugeben*, um Zugang zu allen erlaubten Ressourcen bei seinem VO zu erhalten, wurde zum Teil realisiert: Für die Nutzung von Shibboleth als VO-Management-System ist die Verwendung nur eines einzigen Passworts möglich. In dem Fall wurde ein SSO realisiert. Bei der Verwendung von Proxys wurde keine Möglichkeit gefunden, ein SSO anzubieten.

7.3 Bewertung der Anforderungen des Administrators

Durch die attributbasierte Autorisierung ist das *Verwalten der UUDBVO für den Administrator weniger aufwendig*, als dies bei der identitätsbasierten der Fall wäre. Für das Verwalten der UUDBVO müssen nur noch die Attribute des Benutzers bekannt sein. Die neue Implementierung kann *einfach in ein bestehendes UNICORE-System eingefügt werden*. Dabei wird vorausgesetzt, dass die UNICORE-Umgebung die JAVA SUN JDK1.5 unterstützt. Die Implementierung bietet dem Administrator die *Möglichkeit zwischen einer identitätsbasierten und einer attributbasierten Autrorisierungsform zu wählen*. Alle *System-Zustände, die bei der Implementierung für die Erweiterung eine Rolle spielen werden in Log-Dateien protokolliert*. Die UUDBVO unterstützt keine dynamische Überprüfung der VO-Policies. Der Administrator kann durch *das Verwalten der Attributliste*, die benötigten Attribute für die Autorisierung des Benutzers definieren. Das Problem der UUDBVO sowie der identitätsbasierten UUDB ist, dass es nicht möglich ist, sie von entfernten Rechnern zu administrieren. Ein verteilter Autorisierungsdienst wäre gerade für VOs von Vorteil. Die UUDBVO bietet eine *Möglichkeit an, VO- sowie Campusattribute für die Autorisierung des Benutzers zu kombinieren*. Anhand der Kombination von Attributformen ist eine flexible Autorisierung der VO-Benutzer möglich.

8. Zusammenfassung und Ausblick

UNICORE ist von den gängigen Middleware-Systemen das einzige, welches noch keine Integration von Shibboleth oder VOMS anbietet. Aufgrund dessen wurde UNICORE in der vorliegenden Arbeit um VOMS und Shibboleth erweitert. Dadurch können die VO-Management-Systeme auch unabhängig von der jeweiligen verwendeten Grid-Middleware benutzt werden.

Die rein identitätsbasierte Autorisierung von UNICORE wurde durch eine attributbasierte Autorisierung ersetzt, welche die Probleme der Skalierbarkeit und dem hohen administrativen Aufwand reduziert. Die attributbasierte Autorisierung kann das Verwalten der UUDB von der Last befreien, Zugangsentscheidungen auf der Basis der Identität eines Benutzers treffen zu müssen. Die Entscheidung erfolgt nun aufgrund der Benutzerattribute. Somit müssen in der im Rahmen dieser Arbeit implementierten UUDBVO nur noch die Attribute der Benutzer bekannt sein.

Im Gegensatz zu anderen Grid-Middleware-Systemen, die bisher nur eine attributbasierte Autorisierung durch Campus- oder VO-Attribute erteilen, ist UNICORE durch diese Arbeit das erste Grid-Middleware-System, das Autorisierungs-entscheidungen durch die Kombination *beider* Attributformen trifft.

Darüber hinaus hat sich die Integration nicht nur auf die Technologien VOMS und Shibboleth beschränkt, sondern bietet eine Interoperabilität zu anderen VO-Management-Systemen. Eine Erweiterung von UNICORE um VO-Management-Systeme wie MyVocs oder VOMRS ist ohne großen Aufwand möglich.

Die Integration in UNICORE wurde durch Plugins und veränderbare Komponenten erweitert und stellt somit eine problemlose Einfügung in bereits bestehende UNICORE-Systeme dar.

Die UNICORE Version 6 basiert im Gegensatz zu Version 5 auf der Web-Service-Technologie und bietet eine OGSA-Konformität. Sie erfüllt auch die OASIS WSRF 1.2 und OGF JSDL 1.0-Standards. Das Sicherheitsmodell sowie die AAI werden mit X.509-Zertifikaten realisiert. Die in dieser Masterarbeit entwickelten Plugins und die UUDBVO-Implementierung sollen auch an die Version 6 angepasst werden.

Im Rahmen des EU-Projektes OMII-Europe wird ein VOMS-SAML-Service entwickelt [25], der SAML-Assertions anstatt von Attributzertifikaten als Attribut-Credentials verwendet [26]. Die VOMS-Attribute sind hierbei nicht nur auf FQANS beschränkt und entsprechen eher dem ABAC-Modell. Für UNICORE 5 könnte eine weitere AttributeExtraction-Klasse implementiert werden, die diese SAML-basierten Attribute aus den Proxy-Zertifkaten extrahieren kann.

Das OMII-Projekt hat zusätzlich die Aufgabe eine Integration des SAML-basierten VOMS in UNICORE 6 zu gewährleisten. Hierbei ist eine Kollaboration zwischen OMII-Europe und dem IVOM-Projekt notwendig, um eine einheitliche Integrations-implementierung zu realisieren.

XACML (eXtensible Access Control Markup Language) ist ein XML-Schema, das vom OASIS-Konsortium standardisiert wurde. Es wird verwendet, um Autorisierungs-informationen darzustellen und Autorisierungs-Policies zu definieren. Für verteilte und interoperable Autorisierungsframeworks erweist sich die XACML als eine wichtige Komponente [WP3-07]. Durch die in dieser Arbeit implementierte UNICORE-PDP, könnten XACML-Autorisierungsinformationen in Form von SAML-Assertions für die Autorisierungsentscheidung verwendet werden. Es wäre wünschenswert, dass die UUDBVO auch XACML-basierte Regeln sowie Policies aufstellen und interpretieren kann.

Literaturverzeichnis

[ALF05] R. Alfieri et al. (2005), „*From gridmap-file to VOMS: managing authorization in a Grid environment*", INFN Parma and University of Parma

[BPB04] BPB: Bundeszentrale für politische Bildung (2004), „*Ausländische Direktinvestitionen (ADI) pro Jahr*", Bonn

[CIA04] V. Ciaschini (2004), „*A VOMS Attribute Certificate Profile for Authorization*", Istituto Nazionale di Fisica Nucleare, Parma

[COJ07] N. Cojocaru (2007), „*Realisierung Virtueller Organisationen als Grid-Ressourcen*", Technische Universität München

[FOS98] I. Foster and C. Kesselman (1998), „*The Grid: Blueprint for a new computing infrastructure*," Morgan Kaufmann, Tech. Rep.

[FOS01] I. Foster et al. (2001), *"The Anatomy of the Grid"*, University of Chicago

[FOS04] I. Foster and C. Kesselman (2004), „*The Grid 2: Blueprint for a New Computer Infrastructure"*, Morgan Kaufmann, San Francisco

[FTL07] P. Flury et al (2007), „*Shibboleth Interoperability with Attribute Retrieval through VOMS*", EGEE-II

[GRI06] C. Grimm et. Al. (2006). „*Analyse von AA-Infrastrukturen in Grid-Middleware"*, D-Grid, Fachgebiet 3-4 – Aufbau einer AA-Infrastruktur für das D-Grid

[GRO06] R. Gröper et al. (2006), „*Aktuelle Entwicklungen zu GridShib*", Leibniz Universität Hannover

[HEL76] W. Diffie and M. E. Hellman (1976), „*New Directions inCryptography",* IEEE Transactions on Information Theory

[HOL05] A. Hollosi (2005): „*X.509 Zertifikatserweiterungen für die Verwaltung*“, Bundeskanzleramt, Österreich

[JIN05] H. Jin et al. (2005), „*A RBAC based grid access control architecture*”, Huazhong University of Science and Technology, Wuhan, China

[KAR06] S. Karsch (2006), „*IT – Sicherheit“,* Folien, Fachhochschule Köln

[LEH02] P. Lehmann (2002), „*Konzeption und Implementierung eines Credential Managers*”, Universität des Saarlandes

[LIN03] P. Lindner et al. (2003), „*Gridwelten: User Requirements and Environments for GRID-Computing*”, DFN-Projekt, Stuttgart

[MAK06] S. Makedanz et al (2006), „*Virtuelle Organisationen und Geschäfts-prozesse im Grid*“, AWI, Bremerhaven

[MAR06] S. Marienfeld (2006), *„Aufbau und Analyse einer Shibboleth/GridShib-Infrastruktur”*, Leibniz Universität Hannover

[MCN05] A. McNab(2005): „GridSite status“, University of Manchester

[MIN03] M. Mink (2003), „*Sicheres Single-Sign-On für Webdienste*“, Diplomarbeit, Technische Universität Darmstadt

[MUE97] T. Müller (1997), „*Virtuelle Organisationen*“, Universität Konstanz, Konstanz

[NEU07] H. Neuroth et al. (2007), „*Die D-Grid Initiative*”, Universitätsverlag Göttingen

[PIO05] P. Piochacz (2005), „*Grid Computing*“, TUM Informatik, München

[POL03] Polze et al. (2003), „*Grid Computing*”, Universität Potsdam

[RAM02] M. Rambath (2002), „*Interoperabilität von Grid-Systemen am Beispiel von UNICORE und Globus”*, FHZ Jülich

[SCH07] M. Schiffers (2007), „*Management dynamischer Virtueller Organisationen in Grids*”, Ludwig-Maximimilians-Universität München

[TAN07] D. Tanriverdi (2007), „*Grid-Computing*”, Technische Universität München

[VAC04] J. R. Vacca (2004) „*Public Key Infrastructure”,* Auerbach Publishers Inc.

[VED06] T. Vedder (2006), „*Federated Identity Management mit Shibboleth*“, Seminararbeit, Fachhochschule Bonn-Rhein-Sieg

[WP2-07] P. Gietz et al. (2007), “*IVOM Work Package 2: VO-Management Requirements from a Community Perspective*”, IVOM D-Grid, Tübingen

[WP3-07] P. Gietz et al. (2007), “*IVOM Work Package 3: A Concept for Authorization on D-Grid Resources*”, IVOM D-Grid, Tübingen

Internetlinks

[1] http://www.d-grid.de/, zuletzt besucht am: 03.12.07

[2] http://sourceforge.net/, zuletzt besucht am: 03.12.07

[3] http://www.unicore.eu/, zuletzt besucht am: 03.12.07

[4] http://setiathome.berkeley.edu/, zuletzt besucht am: 03.12.07

[5] http://dgi.d-grid.de/index.php?id=86, zuletzt besucht am: 03.12.07

[6] http://www.ietf.org/rfc/rfc2459.txt, zuletzt besucht am: 03.12.07

[7] http://www.artikel5.de/archiv/gesetze/iukdg_3.html, zuletzt besucht am: 03.12.07

[8] http://www.linux-magazin.de/heft_abo/ausgaben/2004/06/safer_grid, zuletzt besucht am: 03.12.07

[9] https://www.aai.dfn.de/glossar/, zuletzt besucht am: 03.12.07

[10] http://www.oasis-open.org/home/index.php, zuletzt besucht am: 03.12.07

[11] http://www.globus.org/alliance/, zuletzt besucht am: 03.12.07

[12] http://www.globus.org/toolkit/about.html, zuletzt besucht am: 03.12.07

[13] http://www.lrz-muenchen.de/projekte/dgridmab/, zuletzt besucht am: 03.12.07

[14] www.d-grid.de/fileadmin/dgrid_document/Dokumente/VOMS-Thesenpapier.pdf, zuletzt besucht am: 03.12.07

[15] http://www.internet2.edu/, zuletzt besucht am: 03.12.07

[16] http://de.wikipedia.org/wiki/Schibboleth, zuletzt besucht am: 03.12.07

[17] http://www.spiritproject.de/orakel/magie/lyrik/bibel/richter.htm, zuletzt besucht am: 03.12.07

[18] https://spaces.internet2.edu/display/GS/MyVocs, zuletzt besucht am: 03.12.07

[19] http://gridshib.globus.org/docs/gridshib-ca-0.4.0/process-flow.html, zuletzt besucht am: 03.12.07

[20] https://computer.ncsa.uiuc.edu/gridshib-ca-0.4.0/, zuletzt besucht am: 03.12.07

[21] http://www.openidp.com/, zuletzt besucht am: 03.12.07

[22] http://www.protectnetwork.org/, zuletzt besucht am: 03.12.07

[23] http://inqueue.internet2.edu/, zuletzt besucht am: 03.12.07

[24] www.ietf.org/rfc/rfc3281.txt, zuletzt besucht am: 03.12.07

[25] http://repository.omiieurope.org/downloads/project.jsp?projectid=7, zuletzt besucht am: 03.12.07

[26] http://www.unicore.eu/summit/2007/presentations/11_Riedel_VOMS-SAML.pdf, , zuletzt besucht am: 03.12.07

Abbildungsverzeichnis

Abb.1: Das Schichtenmodell nach Foster et al 9
Abb.2: OGSA-Konzept 12
Abb.3: Darstellung eines Systems 16
Abb.4: Betrachtungsebenen von Virtuellen Organisationen 17
Abb.5: VO-Struktur 19
Abb.6: PKI Entities 21
Abb.7: asymmetrische Verschlüsselung 23
Abb.8: Zwei-Wege-Authentifizierung 27
Abb.9: Autorisierung 32
Abb.10: Zugriffswegdarstellung 33
Abb.11: Föderation von Einrichtungen und Anbietern 37
Abb.12: Die Architektur von UNICORE 42
Abb.13: Der UNICORE Client 43
Abb.14: Proxy-Delegation 45
Abb.15: Die AAI von UNICORE 48
Abb.16: identitätsbasiertes Mapping 49
Abb.17: ABAC-Bindungen 50
Abb.18: Shibboleth-Framework 56
Abb.19: Shibboleth mit zentralem VO Management 59
Abb.20: VO-Management durch MyVocs 60
Abb.21: Funktionsweise der GridShib-CA 62
Abb.22: Das VOMS System 65
Abb.23: VOMS Zertifikatserstellung 66
Abb.24: Konzeptionsmöglichkeit der Integration 74
Abb.25: Attribute-Pull vs. Attribute-Push 89
Abb.26: Proxy-Zertifikat mit VOMS-Extension 140

Diagramme

Diagramm 1: Sequenzdiagramm für VO-Credential-Erstellung....................................77
Diagramm 2: Use Case-Diagramm für VO-Credential-Service 78
Diagramm 3: zusammengefasste VO-Credentials 79
Diagramm 4: UNICORE 5-Zugangsschritte für den Benutzer 80
Diagramm 5: Benutzer-Zugang zu UNICORE & VO-CS durch ein BZ..........................81
Diagramm 6: Benutzer-Zugang zu UNICORE & VO-CS durch SLC 82
Diagramm 7: Nutzungs-Reihenfolge vom VO- sowie UNICORE-System 83
Diagramm 8: Aktivitätsdiagramm für eine VO-Autorisierung 85
Diagramm 9: Agent Sequence 87
Diagramm 10: Pull Sequence...88
Diagramm 11: Push Sequence..88
Diagramm 12: Autorisierung durch UUDB..91
Diagramm 13: UUDB-Autorisierung durch Attribute 94
Diagramm 14: Integrationslösung durch Einbehaltung der Shib.-Kommunikation 96
Diagramm 15: Integrationslösung durch ein externes Modul 98
Diagramm 16: Integrationslösung durch GridShib-CA und UNICORE-Plugin.............101
Diagramm 17: Sicherheitsmodell für VOMS 104
Diagramm 18: Integrationsarchitektur der AJO-Lösung 106
Diagramm 19: Integrationslösung durch ein SLC-Service...108
Diagramm 20: Integrationslösung durch eine Keystore-Erweiterung 110
Diagramm 21: Integration der Shibboleth und VOMS-Architektur 112

Listings

Listing 1: SAML AttributeStatement-Assertion 39
Listing 2: ParseCredentialFile.java 121
Listing 3: Inhalt der SLC-File 122
Listing 4: Inhalt der Proxy-File 122
Listing 5: String-Konvertierung des Dateiinhalts..123
Listing 6: Parsen eines X.509-Zertifikats 123
Listing 7: Erstellung eines UNICORE-Keystores..124
Listing 8: Persistente Speicherung des Keystore 124
Listing 9: Erhalten des Benutzer-Keystores 125
Listing 10: Erhalten des Benutzer-Keystores 126
Listing 11: Eigenschaften der VO-HashMap 129
Listing 12: Laden der HashMap...129
Listing 13: Ausführen der Methode load() mit einem ObjectInputStream....................130
Listing 14: njs.properties...134
Listing 15: Verifizierung und Valdierung des Benutzerzertifikats................................135
Listing 16: AttributeExtraction.java 137
Listing 17: VOMS-Proxy-Prinzip 139
Listing 18: AC-Profil nach RFC 3281...141
Listing 19: Modifizierung der AttributeCertificateInfo-Klasse 141
Listing 20: Eigenschaften der VOMS-Klasse...142
Listing 21: extractAttributeCredential() für VOMS-Attribute..143
Listing 22: Eigenschaften der GridShibSAML-Klasse 144
Listing 23: Eigenschaften der GridShibSAML-Klasse 144
Listing 24: getAttributes() für GridSHibSAML-Attribute 145
Listing 25: UNICORE-PDP für VO-Benutzer 148

Aufgabenverteilung:

Da es sich bei diesem Werk um eine Gemeinschaftsarbeit handelt, nehmen die beiden Autoren im Folgenden eine Aufführung der Arbeitsaufteilung vor:

- Arash Faroughi bearbeitete in dieser Arbeit selbständig die Kapitel:
  2.1, 2.2, 2.3, 3.2, 3.4.6, 3.4.7, 5.2, 5.5, 6.1, 6.2.1und 6.2.3

- Roozbeh Faroughi bearbeitete in dieser Arbeit selbständig die Kapitel:
  2.4, 2.5, 2.6, 3.1, 3.4 - 3.4.5, 5.3, 5.4 und 6.2

Alle anderen Abschnitte wurden gemeinsam bearbeitet.

Jeder Autor bearbeitete ein vom Umfang her gleiches Pensum.

Selbstständigkeitserklärung

Wir erklären hiermit, dass wir diese Masterarbeit selbstständig angefertigt haben und alle Teile, die wörtlich oder inhaltlich anderen Quellen entstammen, als solche kenntlich gemacht und in das Literaturverzeichnis aufgenommen haben. Diese Arbeit wurde weder in dieser noch einer ähnlichen Form einer anderen Prüfungsbehörde vorgelegt.

Gummersbach, den 06 Dezember 2007

--------------------------------	--------------------------------
(Arash Faroughi)	(Roozbeh Faroughi)