FORSCHUNGSZENTRUM JÜLICH GmbH
Zentralinstitut für Angewandte Mathematik
D-52425 Jülich, Tel. (02461) 61-6402

Interner Bericht

Ein Web-Service-basierter Autorisierungsdienst für Grid-Umgebungen am Beispiel von UNICORE

André Latour

FZJ-ZAM-IB-2006-02

Februar 2006

(letzte Änderung: 6.2.2006)

Fachhochschule Aachen

Standort Jülich

Fachbereich: Angewandte Naturwissenschaften und Technik

Studienrichtung: Technomathematik

Ein Web-Service-basierter Autorisierungsdienst für Grid-Umgebungen am Beispiel von UNICORE

Diplomarbeit von André Latour

Jülich, Februar 2006

Die vorliegende Diplomarbeit wurde in Zusammenarbeit mit dem Forschungszentrum Jülich GmbH, Zentralinstitut für Angewandte Mathematik, Abteilung Verteilte Systeme und Grid Computing angefertigt.

Diese Diplomarbeit wurde betreut von:

Referent: Herr Prof. Dr. Volker Sander

Korreferent: Herr Dr. Bernd Schuller

Diese Arbeit ist von mir selbständig angefertigt und verfasst. Es sind keine anderen, als die angegebenen Quellen und Hilfsmittel benutzt worden.

Jülich, Februar 2006

Inhaltsverzeichnis

Abkürzungsverzeichnis

ACL	Access Control List
AJO	Abstract Job Object
ASCII	American Standard Code for Information Interchange
BMBF	Bundesministerium für Bildung und Forschung
BSS	Batch Subsystem
CGI	Common Gateway Interface
CORBA	Common Object Request Broker Architecture
CPMD	Car-Parrinello Molecular Dynamics
DBMS	Database Management System
DCOM	Distributed Component Object Model
GPL	Gnu Public License
IDB	Incarnation Database
ITU	International Telecommunication Union
JSDL	Job Service Description Language
JWS	Java Web Service
GGF	Global Grid Forum
HTTP	Hypertext Transfer Protocol
HTTPS	HTTP via SSL
NJS	Network Job Supervisor
OGSA	Open Grid Service Architecture

OSI	Open System Interconnection
PEM	Privacy Enhanced Mail
PKCS12	Personal Information Exchange Syntax Standard
RMI	Remote Method Invocation
RPC	Remote Procedure Call
SAML	Security Assertion Markup Language
SMTP	Simple Mail Transfer Protocol
SOAP	Simple Object Access Protocol
SSL	Secure Socket Layer
TCP	Transmission Control Protocol
TSI	Target System Interface
UAS	UNICORE Authorization Service
UNICORE	Uniform Interface to Computing Resources
UniGrids	Uniform Interface to Grid Services
Usite	UNICORE Site
UUDB	UNICORE User Database
URL	Uniform Resource Locator
Vsite	Virtual Site
WS	Web Service
WSDD	Web Service Deployment Descriptor
WS-RF	Web Service Resource Framework

WSDL	Web Service Description Language
XML	Extensible Markup Language

Abbildungsverzeichnis

Tabellenverzeichnis

1 Einleitung

Grid Computing ist eine zunehmend wichtiger werdende Technologie zur Nutzung verteilter Speicher- und Rechenressourcen mit dem Ziel der effizienten Nutzung von Ressourcen. Neben Anwendungen im wissenschaftlichen Umfeld wird diese Technologie auch zunehmend in kommerziellen Bereichen eingesetzt. Einen Zugang zu diesen verteilten Kapazitäten bieten Grid-Middleware[1] Lösungen. Neben UNICORE, welches in Abschnitt 1.2 und Kapitel 2 detaillierter vorgestellt wird, existieren weitere bedeutende Grid-Middleware Lösungen, wie z.B. Globus [1], gLite [2] und Condor [3].
In dieser Arbeit wird der Entwurf und die Entwicklung eines Autorisierungsdienstes für UNICORE beschrieben, welcher im Rahmen des europäischen Projekts UniGrids [4] entstanden ist. Ziel der Erstellung des Dienstes ist es, eine Web-Service basierte virtuelle Organisation zu erschaffen, die alle Autorisierungsanforderungen für Grid-Komponenten verwaltet und zusätzlich ein sicheres administratives Interface zur Verfügung stellt, das die Pflege und Wartung der Datenhaltung ermöglicht.
Die Arbeit ist wie folgt strukturiert: Zunächst wird innerhalb des ersten Kapitels eine kurze Einführung in die Begrifflichkeit der zugrunde liegenden Technologien gegeben. In Kapitel 2 wird die Grid-Middleware UNICORE, welche die Basis für die hier entwickelte Dienstkomponente darstellt, in Bezug auf ihre Architektur, das Sicherheitsmodell und die bisherige, nur rudimentär umgesetzte, Autorisierungsmöglichkeit vorgestellt. Kapitel 3 gibt einen kurzen Überblick über das Thema Web-Services und die Gestaltungsmöglichkeit ihrer konkreten Realisierung.
Der Schwerpunkt der Arbeit liegt in einer detaillierten Beschreibung des Designs und der Implementierung des UNICORE Authorization Services (UAS), welche in Kapitel 4 vorgenommen wird. In Kapitel 4.5 wird der aktuelle Umsetzungsstatus der Softwarekomponente beschrieben und ein Ausblick auf mögliche Erweiterungen gegeben.

1.1 Grid Computing

Grid Computing ist das Schlüsselwort für eine neue Technologie, die sich innerhalb des letzten Jahrzehnts manifestiert hat und eine neue Form der Nutzung von Ressourcen definiert. In seinem Wortursprung leitet es sich vom englischen Begriff für das Strom-

[1] Middleware ist eine anwendungsunabhängige Technologie die Softwareschnittstellen und/oder Dienste entkoppelter Softwarekomponenten bereitstellt.

netz (Power Grid) ab. Dem Wort und seiner Ableitung immanent ist die Vorstellung von der Einfachheit der Nutzung dieses verteilten Rechennetzes, analog zur einfachen Nutzung des Stromnetzes.

Innerhalb der Natur-, Ingenieurwissenschaften und kommerziellen Simulationen treten heutzutage Problemstellungen[2] auf, deren Bedarf an Rechenkapazität nicht von einzelnen Ressourcen gestillt werden kann. Ziel muss es somit sein, verteilte Ressourcen zu einer virtuellen Einheit zu verbinden um so die gestiegenen Anforderungen der Anwendungen zu bedienen.

Notwendige Voraussetzung für die effiziente Nutzung von verteilten Rechenkapazitäten und deren Kommunikation untereinander, ist eine hoch verfügbare Infrastruktur mit einer hohen Ausfallsicherheit.[3] Zusätzlich soll der Zugang zum Grid einfach möglich sein, ohne Charakteristika der einzelnen verwendeten Rechenressource, wie z.B. Architektur, Betriebssystem, Rechenkapazitäten, Speicherkapazitäten, Batch-Systeme usw. im Detail zu kennen. Eine weitere Anforderung ist es flexibel Rechenkapazitäten innerhalb des Grids abzurufen oder zu reservieren. Im Projekt GRIDWELTEN [5] werden eine Vielzahl von Nutzungsszenarien von Grids charakterisiert, die durch unterschiedliche Anforderungen differenziert werden können.

Reinefeld und Schintke [6] fassen Anforderungen an das Grid-Computing, basierend auf dem OGSA-Standardisierungsvorschlag [7], wie folgt zusammen:

- Unterstützung von heterogenen Ressourcen (Dynamisches Finden von Ressourcen im Grid und Verwaltung der verteilten Dienste)
- Handhabung der Verfahrensregeln lokaler Verwaltungseinheiten (Sicherheits- und Abrechnungsvorschriften auf lokaler Ebene)
- Nutzung der Ressourcen (Allokation und Reservierung der Rechenkapazitäten)
- Job-Ausführung und Qualitätsüberwachung (Überwachung und Kontrolle der Arbeitsabläufe und einhalten von Dienstgütevereinbarungen)
- Sicherheit (Benutzer des Grids müssen authentifiziert und autorisiert werden können)

[2] Proteinfaltungen (Biochemie), Finite Elemente Berechnungen, Monte-Carlo-Simulatoin für Geld-Derivate

[3] Vgl. X-WiN, Wissenschaftsnetz des DFN mit einem Multi-Gigabit-Kernnetz zwischen 43 Netzstandorten

- Skalierbarkeit (das Grid kann dynamisch an wechselnde Anforderungen angepasst werden)
- Verfügbarkeit (das Grid soll flexibel auf Ausfälle reagieren können)

Auf Basis dieser Spezifizierungen sind unterschiedliche Grid-Middleware-Entwicklungen im Einsatz deren Ziel es ist, ein Grid so einfach wie möglich zu nutzen. Neben dem Globus Toolkit [1] ist UNICORE [8] eine bedeutende Grid-Middleware Lösung.

1.2 UNICORE

Das Projekt UNICORE und das Nachfolgeprojekt UNICORE PLUS wurden durch das BMBF gefördert und hatten als Zielsetzung einen nahtlosen, sicheren und intuitiven Zugang zu verteilten Ressourcen innerhalb der deutschen Hochleistungsrechenzentren zur Verfügung zu stellen. Innerhalb dieser Projekte entstand eine Grid-Middelware die ebenfalls den Namen UNICORE trägt. Im weiteren Verlauf dieser Arbeit wird mit UNICORE ausschließlich die Grid-Software bezeichnet. In verschiedenen europäischen Projekten wird die Grid-Software ständig weiterentwickelt und steht auf SourceForge [9] als GNU/GPL zur Verfügung.
Die Hauptmerkmale von UNICORE sind folgende:

- UNICORE verbirgt die Charakteristika verschiedener Rechnerarchitekturen, herstellerspezifischen Betriebssystemen, Batch-Systemen, Anwendungsumgebungen, Dateisystemen, Sicherheitsregelungen usw. vor dem Benutzer.
- Die Sicherheit von UNICORE und seinen Serverkomponenten bzw. der Kommunikation der Komponenten basiert auf dem X.509 Standard.
- Eine graphische Benutzerschnittstelle unterstützt den Anwender in der Erstellung und Verwaltung von Arbeitsabläufen und deren Überwachung
- Ein für Benutzeranforderungen dynamisch erweiterbares Plugin-Konzept

Eine ausführliche Beschreibung der Architektur und die Eingliederung der unterschiedlichen Systemkomponenten der Grid-Middleware UNICORE, sowie des Sicherheitsmodells, ist in Kapitel 2 nachzulesen.
UNICORE wird unter anderem im europäisch geförderten Projekt UniGrids [4] eingesetzt und weiterentwickelt. Das UniGrids Projekt entwickelt eine Grid-Dienst Infra-

struktur basierend auf OGSA[4]. Diese Infrastruktur baut auf der UNICORE Grid-Software und den abgeschlossenen europäischen Projekten EUROGRID [10] und GRIP [11] auf. Das Projektziel ist es, UNICORE in ein System mit Schnittstellen zu überführen, welches konform mit dem Web-Service Ressource Framework (WS-RF) [11] ist und so mit anderen WS-RF konformen Softwarekomponenten interoperieren kann.

1.3 Web-Services

In den letzten Jahrzehnten hat sich die Softwareentwicklung mit der Entwicklung eines Konzepts beschäftigt, welches den Prozess der Softwareentwicklung effizienter gestalten und dadurch die Softwarequalität erhöhen sollte. Das Ergebnis dieser Entwicklung war, dass sich die objektorientierte Entwicklung gegenüber der prozeduralen durchsetzte. Jedoch war diese in Bezug auf Integration von Applikationen nicht ausreichend und so entstand die *Service-orientierte Architektur.* Die *Service-orientierte Architektur* ist eine spezielle Architektur die das Konzept der Integration besonders hervorhebt. Dabei beschränkt sie sich auf die Integration von Applikationen in die Geschäftslogik. Um die Integration zu gewährleisten, wird eine Dienstschicht definiert, die Funktionalitäten von Softwarelösungen als Dienst zur Verfügung stellt, so dass dieser Dienst über standardisierte Internetprotokolle, wie z.B. HTTP oder SMTP aufgerufen werden kann. Deutlich hervorzuheben ist, dass Web-Services primär eine Integrationstechnologie zur Verfügung stellen und selbst keine Entwicklungstechnologie darstellen.
Ausgehend von dieser Differenzierung von Integrations- und Entwicklungstechnologie sind verschiedene Technologien der letzten Jahre, wie z.B. CORBA, CGI-Scripting etc. keine Web-Service-Technologien. Der Hauptunterschied zwischen diesen Technologien und der neuen Technologien, die mit Web-Services gekennzeichnet sind, ist deren Standardisierung in Bezug auf Schnittstellenbeschreibung und Datenaustauschformat.
Diese Web-Service Technologien basieren auf standardisiertem XML [13]. XML bietet eine sprachen-unabhängige Möglichkeit zur Darstellung von Daten.
Ein Web-Service soll verschiedene Eigenschaften besitzen:

- XML-basiert
  Durch die Nutzung von XML als Datendarstellungsschicht für alle Web-Service Protokolle und Technologien, können diese Technologien interoperabel auf ih-

[4] Grundidee von OGSA ist die Darstellung der beteiligten Komponenten als Grid-Service

ren Kernlevel sein. Als Datentransport umgeht XML alle Netzwerk-, Betriebssystem- und Plattformbindungen, die ein Protokoll haben.

- Lose gekoppelt

  Ein Nutzer des Web-Service ist nicht direkt an den Web-Service gebunden. Die Web-Service Schnittstelle kann sich im Laufe der Zeit ändern, ohne den Client in seiner Möglichkeit einzuschränken, die Funktionalität des Web-Service zu nutzen. In einer lose gekoppelten Architektur sind Komponenten einfacher zu verwalten ebenso wie eine einfachere Integration zwischen verschiedenen Systemen möglich ist.

- Grobkörnig

  Objektorientierte Technologien, wie z.B. Java, stellen ihre Funktionalität über individuelle Methoden zur Verfügung. Ein Java Programm verfügt über viele feinkörnige Methoden, welche in einen grobkörnigen Dienst eingefügt werden, der von einem Klienten oder einem anderen Dienst genutzt wird. Die Web-Service Technologie unterstützt auf intuitive Art eine Definition von grobkörnigen Diensten.

- Möglichkeit der synchronen und asynchronen Nutzung

  Synchronität bezieht sich auf die Bindung des Klienten an die Ausführung des Dienstes. In synchronen Aufrufen wartet der Klient auf das Beenden der Ausführung durch den Dienst. Asynchrone Ausführungen erlauben es dem Klienten einen Dienst auszuführen und während der Dienstausführung andere Funktionen aufzurufen. Solche Klienten erhalten ihre Ergebnisse zu einem späteren Zeitpunkt. Asynchrone Abläufe sind unter anderem der Schlüssel zu einer losen Kopplung von Diensten.

- Unterstützung von Remote Procedure Calls (RPC)

  Web-Services erlauben es Klienten auf Basis XML-basierter Protokolle, Funktionen auf entfernten Objekten aufzurufen. Remote Procedure Calls benötigen Input und Output Parameter, die ein Web-Service unterstützen muss.

- Unterstützung von Dokumentenaustausch

  Ein Vorteil von XML ist die generische Möglichkeit nicht nur einfache Daten, sondern auch komplexere Dokumente darzustellen. Einfache Daten können z.B. elementare Datentypen sein, während ein komplexes Dokument z.B. aus der Darstellung eines Buchs bestehen könnte. Web-Services unterstützen den trans-

parenten Austausch von Dokumenten um die Integration der Dienste zu ermöglichen bzw. zu erleichtern.

Diese Eigenschaften verdeutlichen den Vorteil von Web-Services, das Gestalten einer homogenen Umgebung unter Beibehaltung der Kernfähigkeiten und existierenden Softwareapplikationen. Die Informationstechnologie benötigt eine einfache, plattformunabhängige Möglichkeit der Kommunikation zwischen den Anwendungen um deren Funktionalität in einer Service-Architektur zur Verfügung zu stellen.
Eine kurze Darstellung der technischen Merkmale der Web-Service Technologie und der Anwendungsentwicklung von Web-Services wird in Kapitel 3 beschrieben.

2 Die UNICORE-Architektur

In Kapitel 1.2 wurde die UNICORE Software als eine Middleware-Lösung für den Zugang zu Grids beschrieben. Innerhalb des UNICORE bzw. UNICORE Plus Projekts wurde eine umfangreiche Funktionalität entwickelt, die den Nutzer bei der Erstellung und Verwaltung komplexer Arbeitsabläufe unterstützt, welche anschließend auf verschiedenen Zielsystemen zur Ausführung gelangen. Die Projektergebnisse sind im UNICORE Plus Final Report [14] ausführlich beschrieben. Die technische Struktur von UNICORE wird in Kapitel 2.1 dokumentiert. UNICORE wird durch folgende Schlüsselfunktionen charakterisiert:

- Job creation and submission

  Ein graphischer Client unterstützt den Benutzer bei der Erstellung von Arbeitsanweisungen, welche auf den verfügbaren Zielsystemen innerhalb des UNICORE-Grids zur Ausführung gelangen. Dazu werden verschiedene Abhängigkeiten zwischen den Arbeitsabläufen definiert, die bei der Ausführung berücksichtigt werden.
- Job management

  Dem Benutzer obliegt die Kontrolle über den Status der Arbeitsabläufe. Des Weiteren werden Log-Informationen über Fehlersituation über den Client zur Verfügung gestellt.
- Data management

  UNICORE unterstützt den Import bzw. Export der Daten vom Client zum Zielsystem und vice versa. Weiterhin kann UNICORE-Daten innerhalb von Vsites[5] transferieren.
- Anwendungsunterstützung

  Der Client wird durch ein Plugin-Konzept erweitert, welches über definierte Schnittstellen anwendungsspezifische Eingaben bzw. Ausgaben innerhalb der graphischen Benutzerschnittstelle des Client zur Verfügung stellt, wie z.B. das CPMD-Plugin.
- Flusskontrolle der Arbeitsabläufe

  Zum einen kann über einen azyklischen Graph eine Abhängigkeit der Arbeitsabläufe innerhalb des graphischen Client definiert werden. Zum anderen ermögli-

[5] Vsite ist der UNICORE spezifische Bezeichnung für ein Zielsystem

chen Kontrollstrukturen (if-else, do-while, until etc.) eine Beschreibung der Abarbeitungsreihenfolge.

- Single sign-on

  Single sign-on beschreibt ein einmaliges Anmelden am Grid, welches in UNICORE über X509-Zertifikate realisiert wird. Diese Zertifikate stellen einen eindeutigen Zusammenhang zwischen Benutzer und lokalem Account des Benutzers auf dem Zielsystem her. Jede Vsite kann den Zugang zu den Ressourcen einschränken, indem sie Zertifikate mit der UNICORE User Database (UUDB) abgleicht.

- Resource management

  UNICORE implementiert ein einfaches Ressourcenmodel. Die auf dem Zielsystem verfügbaren Ressourcen werden durch den lokalen Administrator statisch eingetragen. Zum Zeitpunkt der Submission der Arbeitsanweisungen stehen die Ressourceinformationen zur Verfügung und können unter Berücksichtigung der gestellten Ressourceanforderungen innerhalb der Anweisungen vergeben werden.

Diese Merkmale sind keine vollständige Beschreibung der Funktionalität von UNICORE, sondern beschreiben Aspekte der Realisierung, die aus Anforderungen an eine Grid-Middleware entstanden sind. Für die ausführliche Dokumentation aller Eigenschaften ist auf den Projektreport [14] zu verweisen. Auf die technische Architektur von UNICORE wird im folgenden Abschnitt eingegangen.

2.1 Drei-Schichten-Architektur

UNICORE besteht aus einer Client-Server-Architektur, welche softwareseitig drei voneinander logisch getrennte Schichten implementiert. Die Präsentationsschicht ist für die Repräsentation der Daten und Benutzereingaben zuständig und wird innerhalb von UNICORE durch einen graphischen Benutzerklienten abgebildet. Die Logikschicht beinhaltet die Verarbeitungsmechanismen der Software und wird durch das Gateway und den Network Job Supervisor (NJS) innerhalb eines Servers in UNICORE realisiert. Die Datenschicht ist verantwortlich für die Ausführung auf dem Zielsystem und wird durch das Target System Interface (TSI) implementiert. Die einzelnen Schichten kommunizie-

ren über definierte Schnittstellen miteinander, dazu werden Datenströme über Socketverbindungen verwendet.

Die Vorteile einer Drei-Schichten-Architektur liegen in der einfachen Skalierbarkeit sowie in der logischen Trennung der einzelnen Schichten. Einzelne Komponenten lassen sich somit in dieser Architektur besser warten bzw. austauschen.

Die folgende Abbildung stellt die Anbindung der einzelnen Komponenten von UNICORE graphisch dar.

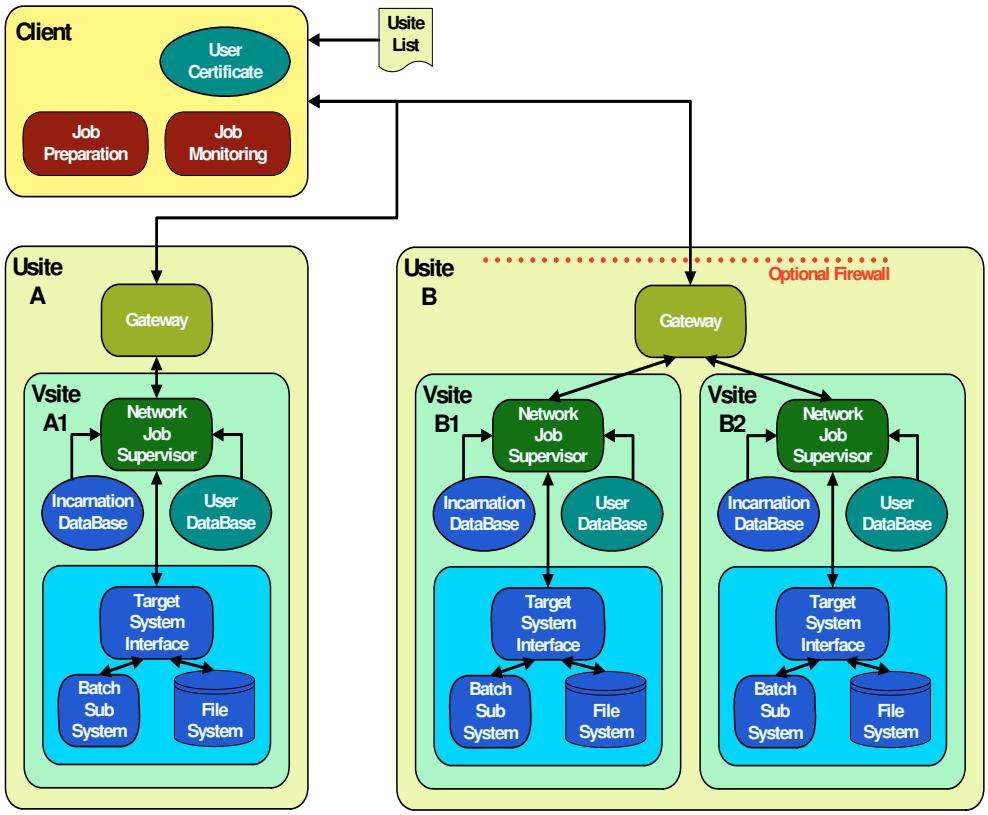


Abbildung 2-1 UNICORE Architektur (Autor: D. Mallmann, FZ Jülich)

Die einzelnen Systemkomponenten sind folgendermaßen realisiert:

- Der **Client** umfasst eine in Java implementierte graphische Benutzeroberfläche, die zur Joberstellung und Jobverwaltung verwendet wird und auf dem lokalen System des Benutzers läuft. Abbildung 2-2 zeigt eine Darstellung des Client in UNICORE. Der Client ist die einzige Schnittstelle des Endanwenders zum Gateway einer Usite[6] von UNICORE. Über ein XML-Dokument ist im Client eine

[6] Eine Usite stellt einen logischen Verbund von Vsites dar

Liste von Kommunikationsendpunkten verfügbarer Gateways konfigurierbar. Über diese Endpunkte kann mit einem gültigen X509-Nutzerzertifikat eine Verbindung zu den Gateways aufgebaut werden. Dieses Zertifikat wird zusätzlich zum Signieren der submittierten Arbeitsanweisungen (AJO) an das Gateway verwendet.

Eine genauere Beschreibung der sicheren Kommunikation des Klienten zum Gateway ist in Kapitel 2.1.3 zu finden.

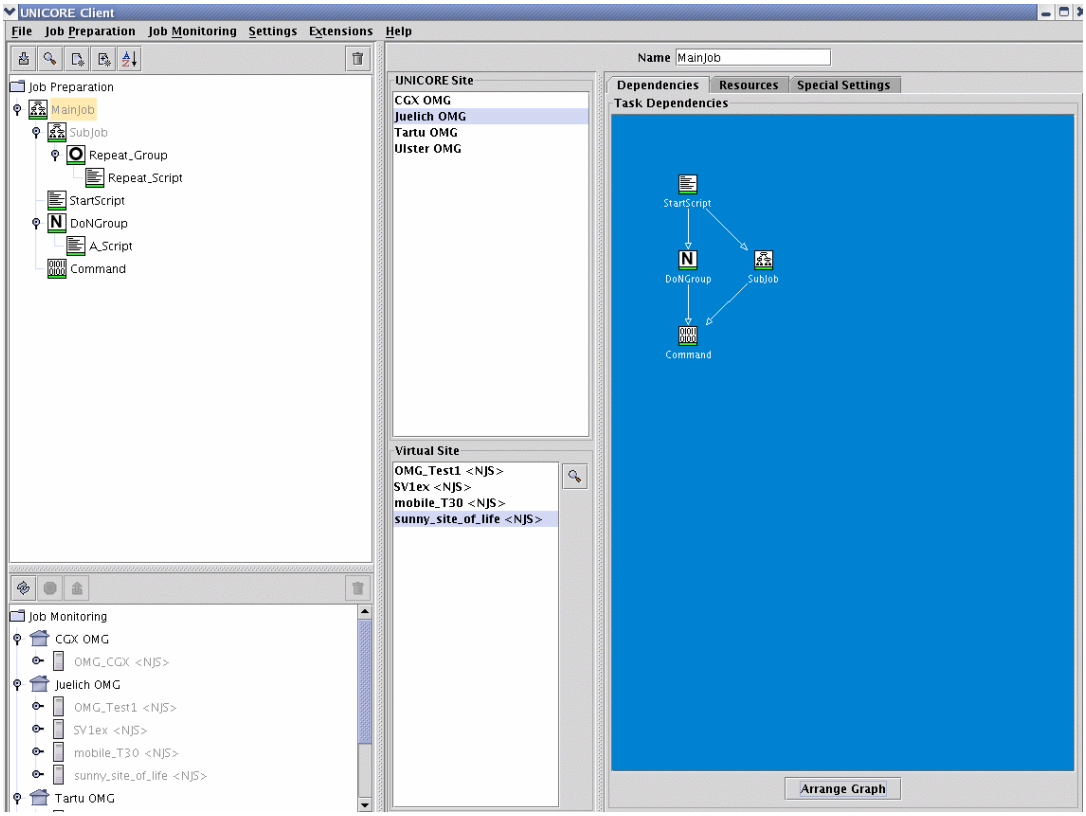


Abbildung 2-2 Graphischer UNICORE-Client

- Das UNICORE **Gateway** ist der einzige Kommunikationspunkt über den ein Nutzer auf Usites zugreifen kann. Der Kommunikationsendpunkt für SSL-Verbindungen wird über eine Internet-Adresse und einen Port[7] zugänglich gemacht. Das Gateway ist zuständig für die Überprüfung der Autorisierung von Nutzern in Hinblick auf die erlaubte Nutzung von Vsites, es sollte auf einem sicheren System laufen und kann optional von einer Firewall geschützt werden. Pro Usite existiert jeweils ein Gateway.

[7] Ein Gateway kann weltweit eindeutig über eine Internetadresse und einen Port identifiziert werden.

- Gemeinsam mit dem Gateway bildet der **Network Job Supervisor (NJS)** die Logikschicht der Drei-Schichten-Architektur. Jede Vsite hat einen eigenen NJS. Der NJS verwaltet alle an die Vsite submittierten Arbeitsanweisungen und überführt diese anhand der Beschreibung des AJO in die spezifischen Arbeitsanweisungen des Zielsystems. Diese Überführung basiert auf den Spezifikationen der IDB (siehe nächster Punkt). Falls voneinander abhängige Arbeitsanweisungen definiert werden, kann der submittierende NJS auch in der Rolle eines Klienten agieren.
- Die Spezifikationen in der **Incarnation Data Base (IDB)** sind abhängig vom Zielsystem. Innerhalb der IDB wird definiert, wie die einzelnen Arbeitsanweisungen, die im AJO beschrieben, sind auf dem Zielsystem zur Ausführung gelangen. Die Zielsysteme können sich dabei in Bezug auf Architektur, Batch-System[8], Betriebssystem etc. voneinander unterscheiden. Weitere Einträge in der IDB umfassen Konfiguration und Definition von Ressourcen und Umgebungsvariablen. Die IDB ist durch ein einfaches ASCII-File realisiert.
- Das **Target System Interface (TSI)** ist ein Dienstprogramm, welches einen direkten Zugang zur ausführbaren Umgebung des Zielsystems und eventuell vorhandenen Batch-Sub-Systemen hat. Die TSI-Prozesse kommunizieren über verschiedene Datenkanäle mit dem NJS. Innerhalb dieser Ausführungsprozesse werden Daten getrennt von Befehlen über Sockets übertragen. Das TSI wird durch Perlskripte realisiert, die dem Zielsystem angepasst sind.

Eine weitere Komponente der UNICORE-Architektur ist die **UNICORE User Database (UUDB)**, die bisher nicht weiter beschrieben wurde. Aufgabe der UUDB ist es, Nutzer innerhalb des UNICORE-Grids zu autorisieren. Dieses Konzept bildet die Basis für die in Kapitel 4 beschriebene Weiterentwicklung der UUDB zu einem Autorisierungsdienst, der einige Nachteile der ursprünglichen Realisierung beseitigt und somit als Kern dieser Arbeit zu sehen ist.

[8] In einem Batch-System werden Arbeitsanweisungen zuerst einer Queue zugewiesen und werden erst zu einem definierten Zeitpunkt ausgeführt, der von einem Scheduling-Algorithmus festgelegt wird

2.1.1 UNICORE User Database (UUDB)

Innerhalb des UNICORE-Grids autorisiert die UUDB den Benutzer[9] gegenüber dem Grid. Autorisieren bedeutet dabei, dem Benutzer eine Rolle und einen lokalen Benutzeraccount zuzuweisen. Die UNICORE-spezifische Rolle differenziert z.B. den User von einem NJS, der ebenfalls als Client agieren kann Der Benutzeraccount ist der spezifische UNIX-Login auf dem Zielsystem, innerhalb dessen Privilegien das TSI die Arbeitsanweisungen ausführt. Durch Eintrag einer Autorisierungskennung in die UUDB wird dem Benutzer erlaubt, auf das entsprechende Zielsystem zuzugreifen. Aus dieser Tatsache wird direkt ersichtlich, dass jede Vsite eine eigene UUDB haben muss, um die Zugangsinformationen zu verwalten. Innerhalb dieses Konzepts ist die Relation zwischen Autorisierungskennzeichen und Zugangsinformationen wichtig. Mehrere Autorisierungskennzeichen können auf eine gemeinsame Zugangsinformation verweisen. D.h. mehrere Nutzer arbeiten physisch auf dem Zielsystem unter demselben Benutzeraccount. Der umgekehrte Fall, ein Autorisierungskennzeichen verweist auf mehrere Benutzeraccounts, muss ausgeschlossen werden, da das eindeutige Autorisierungskennzeichen keine multiplen Identitäten innerhalb von Benutzeraccounts annehmen darf. Aus dieser Zuordnung ergibt sich eine n:1 Relation zwischen den Autorisierungskennzeichen und den Zugangsinformationen, bestehend aus Rolle und Benutzeraccount. Die technische Realisierung dieses Konzepts ist in Kapitel 2.1.2 beschrieben. Die UUDB ist Teil des UNICORE-Servers und damit in der Logikschicht der Drei-Schichten-Architektur angeordnet. Die Sicherheitsmechanismen von UNICORE, die das Konzept der Autorisierung benötigen, werden in Kapitel 2.2 dargestellt.

2.1.2 Konzept der ursprünglichen Realisierung

Die technische Realisierung der UUDB ist relativ einfach gehalten. Innerhalb der Konfigurationsumgebung des NJS ist es möglich modulare Implementierungen der UUDB einzufügen, ohne dass dabei die anderen Softwarekomponenten beeinflusst bzw. angepasst werden müssen. Es existieren dazu eigene Schnittstellenbeschreibungen zum NJS, die somit eine Integration einer neuen UUDB gewährleisten. Die ursprüngliche Implementierung der UUDB ist ein serialisiertes Java-Objekt, welches zur Speicherung in eine binäre Datei geschrieben wird. Die Datenhaltung wird durch die Datenstruktur

[9] Der Benutzer kann auch ein NJS sein, der als Client agiert

HashMap in Java realisiert. Eine *HashMap* basiert auf dem *Map* Interface und realisiert eine effiziente Speicherung eines *(key, value)* Tupels, wobei der *key* dem Zertifikat entspricht und der *value* aus den Zugangsinformationen besteht. Eine Darstellung der UUDB wird in Abbildung 2-3 aufgezeigt.

Für die Implementierung der UUDB-Klasse wird auf die SourceForge-Seiten von UNICORE [9] verwiesen. Zur elementaren Administration der UUDB stehen einige UNIX Shellskripte zur Verfügung.

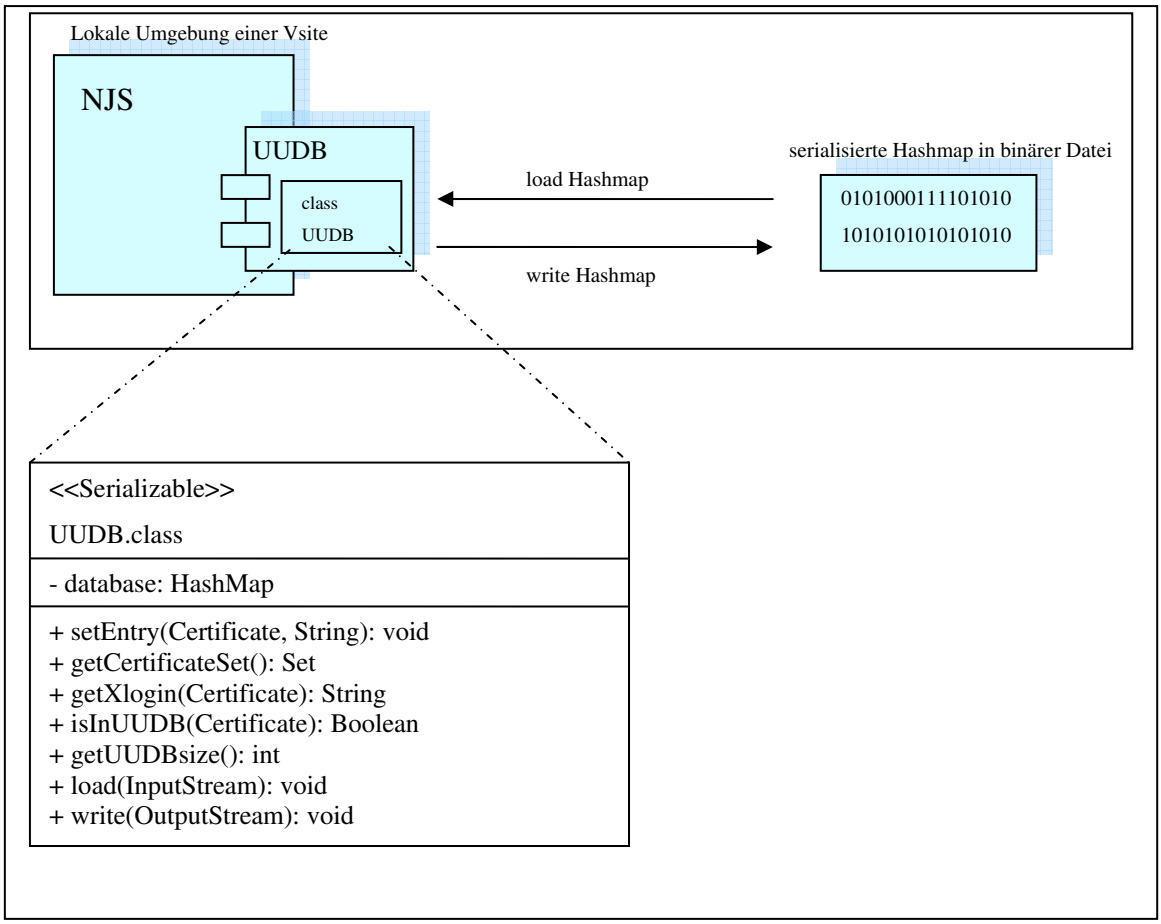


Abbildung 2-3 Darstellung der ursprünglichen UUDB

2.1.3 Restriktionen des Konzepts und der Realisierung

In Kapitel 2.1.2 wurde die rudimentäre Realisierung und Implementierung der UUDB beschrieben. Hieraus ergeben sich unmittelbar wesentliche Restriktionen und Nachteile. Diese bilden somit die Motivation für diese Arbeit. Ein verbessertes, auf Web-Service basiertes Konzept, wird in Kapitel 4 beschrieben. Die Hauptnachteile sind:

- Durch die Speicherung eines serialisierten Java-Objekts ist die Datenspeicherung und die Verwendung der Daten an eine feste Programmiersprache (Java) gebunden, da sie durch deren Serialisierungsmechanismen in die Datei geschrieben wird. Wei-

terhin kann diese Datei nicht ohne weiteres editiert werden, da sie im binären Format vorliegt.

- Es existiert keine Trennung zwischen der logischen Funktionalität und einer Funktionalität die die physisch Speicherung und Darstellung der Daten umfasst. Ebenso ist es hier durch eine fehlende Speicherschnittstelle nicht möglich die Form der Datenspeicherung variabel zu halten. Die logische Funktionalität umfasst die Überprüfung, das Hinzufügen, das Löschen etc. von Zertifikaten und die Zuordnung von Zertifikaten zu den Zugangsinformationen des Zielsystems. Die Speicherfunktionalität umfasst alleine die physische Speicherung in Anhängigkeit der verwendeten Datenspeicherung.
- Die UUDB ist fest an das System gebunden, auf dem der NJS arbeitet. Es ist somit nicht möglich die Funktionalität der UUDB an einer dezentralen Lokation zu verwenden, da die Form der Implementierung keine Kommunikation unterstützt, die über die Speicherung in das lokale File-System hinausgeht.
- Es existieren keine Schnittstellen zur UUDB, die ein Abrufen von Autorisierungsinformationen durch andere Grid-Komponenten möglich macht.
- Es ist nicht möglich die UUDB von entfernten Rechnern zu administrieren, d.h. Zertifikate hinzuzufügen, zu löschen, zu erneuern, etc.
- Ein sehr bedeutender Nachteil ist, dass innerhalb größerer UNICORE-Grids jede Vsite eine eigene UUDB definiert, um die für die Vsite notwendigen Zugangs- und Autorisierungsinformationen zu verwalten. Somit muss jede einzelne UDDB separat administriert werden. Ein Fehlen von „entfernten“ Administrationsschnittstellen macht sich auch hier bemerkbar.

Diese Nachteile bilden die Grundlage für die in Kapitel 4.1 beschriebene Motivation der Anforderung an ein neues Autorisierungskonzept.

2.2 UNICORE-Sicherheitsmodell

Das UNICORE Sicherheitsmodell basiert auf der Verwendung von X.509v3 Zertifikaten zur Authentifizierung von Nutzern wie auch Softwarekomponenten. Ein Zertifikat assoziiert die Identität mit einem öffentlichen Schlüssel, wobei die Identität selbst als *subject* bezeichnet wird. Ein Zertifikat wird dann akzeptiert, wenn es von einer vertrauenswürdigen Instanz, der Certification Authority (CA) signiert wurde. Die Identität die das Zertifikat signiert wird *issuer* genannt. Das Zertifikat beinhaltet Informationen über

das *subject*, den öffentlichen Schlüssel des *subjects* sowie Informationen über den *issuer*. Die Kombination der Information wird kryptographisch signiert und die Signatur wird auch zum Bestandteil des Zertifikats. Das Zertifikat ist signiert ist, es kann über unsichere Kanäle verteilt werden.

Verschiedene Standards spezifizieren den Inhalt eines Zertifikats. Der am meisten verbreitete Standard ist der durch die ITU veröffentlichte X.509-Standard [15]. Die aktuellste Version ist X.509v3. Tabelle 1 beschreibt den Inhalt eines X.509-Zertifikats.

Element	*Beschreibung*
Version	X.509 v1, v2 oder v3
Serial Number	Eindeutige Nummer des Ausstellers
Signature Algorithm	Name des verwendeten Signatur Algorithmus
Issuer	Der Name des Ausstellers
Validity Period	Gültigkeitszeitraum
Subject	*subject* Name
Subject's public key	Der öffentliche Schlüssel des *subject*
Issuer's unique identifier	Eindeutige Identifizierung des Austellers
Subject's unique identifier	Eindeutige Identifizierung des *subject*
Extension	Zusätzliche Informationen
Signature	Die Signatur aller genannten Elemente

Tabelle 1 Inhalt eines X.509 Zertifikats

Eine Darstellung der X509 sind textbasierte PEM Dateien. Abbildung 2-4 zeigt die Darstellung eines X.509 Zertifikats im PEM Format.

```
-----BEGIN CERTIFICATE-----
MIIEMTCCAxmgAwIBAgIBADANBgkqhkiG9w0BAQQFADCBpDELMAkGA1UEBhMCREUx
DDAKBgNVBAgTA05SVzEQMA4GA1UEBxMHSnVlbGljaDEnMCUGA1UEChMeRm9yc2No
dW5nc3plbnRydW0gSnVlbGljaCBHbWJIMQwwCgYDVQQLEwNaQU0xFzAVBgNVBAMT
Dk9wZW5Nb2xHUklEIENBMSUwIwYJKoZIhvcNAQkBFhZvbWctY2FAb3Blbm1vbGdy
aWQub3JnMB4XDTA0MTIwODEyMTAxNVoXDTA2MDExMjEyMTAxNVowgaQxCzAJBgNV
BAYTAkRFMQwwCgYDVQQIEwNOUlcxEDAOBgNVBAcTB0p1ZWxpY2gxJzAlBgNVBAoT
L3d3dy5vcGVubW9sZ3JpZC5vcmcvY2EvY2VydHMuY3JsMA0GCSqGSIb3DQEBBAUA
A4IBAQARuHDO8aiVQ629yGJy6o+CfamuideE2YTS09ct84ctBxE6JGkh8orxFSIq
w9yMPNBfgDRxR2x2mo3K5UUXobGU4Uwq8guu46kCKyCXMotsBv9SOwhAQabHaNo1
kbdNY4bjT2A3GEvjgdUJ7kUSeo+SqKI3jesUEig7zWDU7HQeonlcaSudWednJ9Za
fMqhN8t9TOwxUb2BGpMxYk9hJ5Ta2iI3HU2bxYgXfYB6Vm8WelNPu9fXaFor/Pl6
gNw76HwiLL0s8xuqYSqx7OKzv7y16Zu7dUIYiuTZqEUn+bVTSiOHY07Hb/tXKS7v
m9unPycN+TiyrnEAY7E91qZ9GiSm
-----END CERTIFICATE-----
```

Abbildung 2-4 Darstellung eines X.509 Zertifikats im PEM Format

Das UNICORE-Sicherheitsmodell basiert auf Zertifikaten, die von einer vertrauenswürdigen Instanz, der Certification Authority, ausgestellt wurden. Die gesamte Kommuni-

kation zwischen dem Klienten und dem Gateway wird über eine SSL-Verbindung[10] vorgenommen, die somit auch die Vertraulichkeit und Integrität der Transaktionen sicherstellt. Die SSL-Verbindung wird erst aufgebaut, wenn ein Austausch der öffentlichen Schlüssel zwischen dem Gateway und dem Klienten stattgefunden hat. Der Client überprüft, ob die Identität des Gateways als vertrauenswürdig eingestuft wurde und das Gateway überprüft ob die ausstellende CA des öffentlichen Schlüssel des Client bekannt ist. Der Client besitzt einen Keystore[11], der die öffentlichen und privaten Schlüssel für den Zugang zu den Usites verwaltet. Die Arbeitsanweisungen, die submittiert werden, werden zunächst mit dem privaten Schlüssel signiert, um die Integrität innerhalb des Grids sicherzustellen. Neben der Integrität wird auch die Authentizität gewährleistet, da nur der Benutzer im Besitz des privaten Schlüssels sein kann. Innerhalb des Arbeitsanweisungen (AJO) wird der öffentliche Schlüssel versendet, dessen Aussteller vom Gateway als vertrauenswürdig erkannt werden muss. Anschließend autorisiert der NJS den Benutzer anhand des Schlüssels durch die UUDB und prüft zusätzlich den Job mit dem öffentlichen Schlüssel auf Integrität. Danach wird der Job vom TSI ausgeführt.

[10] SSL bezeichnet ein asymmetrisches Verschlüsselungsverfahren auf Transportschicht. Es wird genauer in Kapitel 4.4.1 beschrieben.

[11] Der Keystore ist ein Container von öffentlichen und privaten Schlüsseln z.B. im PKCS12 Format

3 Web-Services

In Kapitel 1.3 wurde bereits eine Einführung in das Konzept der Web-Service-Technologie innerhalb einer Service-orientierten Architektur gegeben. Innerhalb der Web Service Kategorie sind verschiedene Technologien innerhalb der letzten Jahre entstanden. Diese sind nicht mit existieren Ansätzen wie DCOM, Java RMI oder CORBA zu vergleichen, da diese Technologien zum Teil komplexe und systemspezifische Infrastrukturkomponenten verwenden. Beispielsweise gibt es zwischen CORBA und der Web-Service-Technologie zwei wesentliche Unterschiede. Der erste Unterschied besteht darin, dass SOAP wesentlich weniger komplex als frühere Ansätze ist, weshalb die Hürde für eine standardisierte SOAP-Implementierung bedeutend geringer ist. Der andere wesentliche Vorteil, den Web-Services gegenüber früheren Ansätzen haben, besteht darin, dass diese mit den Standardwebprotokollen HTTP und TCP/IP sowie der Auszeichnungssprache XML zusammenarbeiten.

Innerhalb dieser Zeit wurden die drei primären Kerntechnologien SOAP, WSDL und UDDI als weltweiter Standard deklariert. Die Grundlagen der elementaren technischen Kernkonzepte der Web-Services werden in Kapitel 3.1 beschrieben und bilden eine technische Grundlage des Web-Service Standards und sind für das eigentliche Verständnis der Entwicklung und Implementierung des UAS in Kapitel 4 Voraussetzung. Durch ihre Eigenschaft einer weltweiten nahtlosen standardisierten Integrationstechnologie stellen Web-Services eine natürliche Integrationsumgebung für Grid-Dienste dar, denn die in Kapitel 1.3 beschriebenen Merkmale der Web-Service-Technologie lassen sich fast analog auf eine Grid-Technologie übertragen.

Die Web-Service-Technologie ist eine reine Integrationstechnologie und keine Entwicklungstechnologie. Sie bildet eine konzeptionelle Grundlage. Für eine konkrete Anwendungsentwicklung auf Basis des Web-Service Konzepts bedarf es einer eigenen Entwicklungtechnologie, die in einer Implementierungsebene auf dem Konzept der Web-Services aufsetzt. Innerhalb der Entwicklungstechnologie gab es innerhalb der letzten Jahre viele kommerzielle[12] und auch Open-Source Softwarelösungen, die die Kernkonzepte implementieren. Die in dieser Arbeit beschriebenen Entwicklungen basieren auf Axis. Axis wird von der Apache Group [16] entwickelt und stellt ein umfangreiches Software Framework zur Implementierung und Nutzung von Web-Sevices zur Verfü-

[12] Kommerzielle Entwicklungen werden häufig innerhalb von Applikationsservern angeboten

gung. Kapitel 3.2 gibt eine kurze Einführung in die elementaren Begriffe und Konzepte der programmtechnischen Entwicklung von Web-Service mit Axis und sind ebenfalls für das Verständnis der Entwicklung des UAS in Kapitel 4 von Vorteil.

3.1 Grundlagen der Web-Services

Die vier Kerntechnologien der Web-Services sind XML, SOAP[13], WSDL und UDDI. Eine weitere für die Web-Services wichtige Technologie stellt die Auszeichnungssprache XML dar. Abbildung 3-1 zeigt zunächst eine schematische Darstellung und Verwendung der Technologien, bevor diese anschließend genauer erläutert werden.

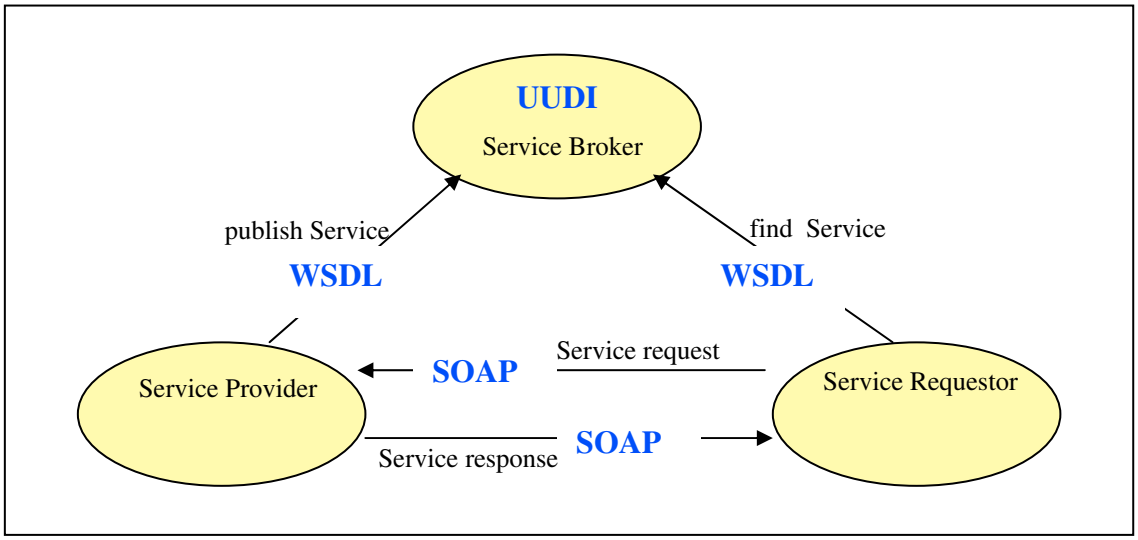


Abbildung 3-1 Schematische Darstellung eines Web-Service

Die Basis für SOAP und WSDL stellt **XML** dar. XML ist eine strukturierte und selbstbeschreibende Möglichkeiten Daten unabhängig von Protokollen, Applikationen, Betriebsystemen oder Programmiersprachen darzustellen. XML-Schema beschreibt dabei die Gültigkeit bzw. Konformität eines XML-Dokuments anhand bestimmter Regeln [55]. XML ist das Fundament der Web-Services, auf dem alle Standards zum Beschreiben, Finden und Ausführen von Web-Services basieren. SOAP und WSDL werden durch XML Schema beschrieben wie auch die Sicherheitsstandards XML-Encryption, XML-Signature etc. XML ist textbasiert und somit einfach zu lesen und auch zu manipulieren. Für das Auswerten und Verarbeiten von XML Dokumenten gibt es eine Vielzahl von Bibliotheken.

[13] Ursprünglich stand SOAP für Simple Object Access Protocol, innerhalb der W3C SOAP 1.2 Spezifikation ist SOAP nicht länger ein Akronym

SOAP ist ein Protokoll, mit dem man XML Dokumente über eine Vielzahl von Transportprotokollen austauschen kann, wobei HTTP das am meisten verbreitete ist. Eine SOAP-Nachricht ist ein XML Dokument, welches eine einfache, konsistente aber auch erweiterbare Möglichkeit zur Verfügung stellt mit der XML Daten zwischen Applikationen bzw. Diensten ausgetauscht werden können. Das SOAP-Modell erlaubt dabei eine deutliche Trennung zwischen einem infrastrukturellen Verarbeiten der Nachricht über ein Netzwerk und dem eigentlichen Transport der Daten. Abbildung 3-2 zeigt schematisch den Aufbau einer SOAP-Nachricht. Innerhalb des SOAP-Envelope wird eine XML Nachricht verschickt. Der Envelope ist ein einheitlicher Container, der von einem Transportprotokoll versendet werden kann. Innerhalb des SOAP-Envelope gibt es den SOAP-Header. Dieser beinhaltet Informationen über die SOAP-Nachricht. Diese Informationen beziehen sich auf die Verwaltung und z.B. das Sichern des Pakets. SOAP ist erweiterbar und bindet diese Erweiterungen in den Header-bereich ein. Der SOAP-Body enthält die eigentlichen Daten, die zwischen den Endpunkten der Transmission ausgetauscht werden. Das kann ein beliebiges XML-Dokument sein, das z.B. einen RPC beschreibt.

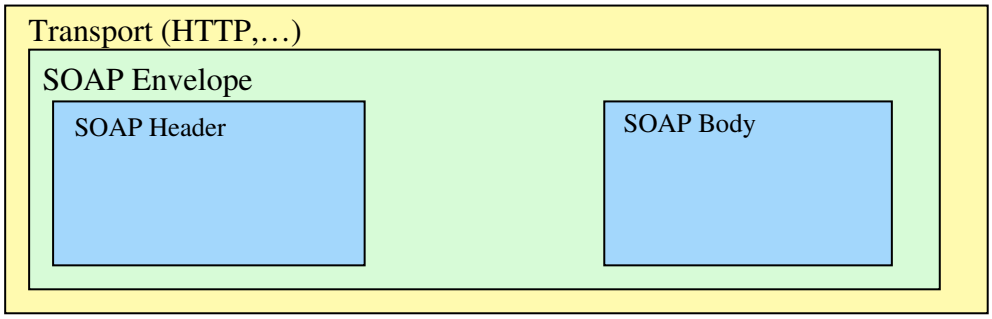


Abbildung 3-2 Aufbau einer SOAP-Nachricht

WSDL ist eine XML-Schema, die die Struktur eines Web-Service und die auszutauschenden SOAP-Nachrichten definiert. WSDL beschreibt somit, wo ein Dienst zu finden ist, was der Dienst machen kann und wie man mit dem Dienst kommuniziert. Die WSDL-Definition eines Dienstes wird durch Methodenaufrufe beschrieben, die auf entfernten Objekten arbeiten. WSDL ist der Teil, den man veröffentlicht, um die Regeln festzulegen innerhalb derer man mit dem Dienst interagieren kann. Eine WSDL-Spezifikation umfasst einen Teil, der die Input- und Output-Nachrichten definiert. Ein weiterer Teil beschreibt wie die Nachrichten innerhalb des SOAP-Envelope verpackt sein müssen und wie sie übertragen werden. Der letzte Teil legt fest, wo der Endpunkt für eine spezifische Web-Service Implementierung zu finden ist.

Die Rolle der **UDDI** ist im Sinne eines Verzeichnisdienstes zu sehen, der Informationen zum Finden von Web-Services bzw. deren Funktionalität bereitstellt. Es wird hier zwischen den *White Pages, Yellow Pages* und *Green Pages* unterschieden. Die *White Pages* enthalten ein Namensregister mit der Auflistung der Anbieter und notwendigen Kontaktinformationen. Die *Yellow Pages* beschreiben als Branchenverzeichnis die spezifische Suche nach der Funktionalität eines Dienstes und referenzieren die *White Pages*. Die *Green Pages* enthalten spezifische technische Details zu den angebotenen Web-Services. Die UDDI kann innerhalb von größeren Organisationen sehr nützlich sein um das Finden und Nutzen von Web-Services zu erleichtern. Innerhalb der Beschreibung der Entwicklungen dieser Arbeit wird die UDDI keine Rolle spielen.

3.2 Web-Service Entwicklung mit Axis

Apache Axis ist die Open Source Implementierung des Web-Service Standards SOAP unter der Lizenz der Apache Group. Die Vorteile der Axis Implementierung sind die Open-Source-Lizenz, flexible Konfiguration und Erweiterbarkeit, Unterstützung verschiedener Transportprotokolle (HTTP, FTP, JMS oder SMTP), Unterstützung des SOAP 1.2 Standard und WSDL Unterstützung. Weiterhin werden Bibliotheken für die Erstellung von Client-Anwendungen bereitgestellt, sowie ein TCP-Monitor zum Überwachen und Manipulation von SOAP-Nachrichten.
Für das Verständnis der weiteren Beschreibungen werden zwei Begriffe definiert die im weiteren Verlauf verwendet werden. *Hosting* bezeichnet das Bereitstellen des Dienstes innerhalb des Kontextes einer ausführenden Softwareumgebung. *Deployment* bezeichnet den Vorgang des Einbindens eines Dienstes in die Laufzeitumgebung einer Webanwendung, die dann den entsprechenden Dienst bereitstellen kann.
Für das Hosting, sowie das Deployment von Web-Services bedarf es einer Serverumgebung innerhalb der ein Web-Service angesprochen werden kann. Axis liefert zwar innerhalb der Implementierung einen sehr einfachen Standalone-Server mit, jedoch wird die Verwendung des Apache Tomcat Servers für das Hosting des Dienstes empfohlen. Innerhalb der Entwicklung von Web-Service Anwendungen stellt Axis dem Entwickler leistungsfähige Deployment-Mechanismen zur Verfügung. Die einfachste Form des Deployment wird durch JWS Dateien realisiert. Hierzu werden die JAVA-Dateien mit dem Quellcode der Web-Service Implementierung in die Laufzeitumgebung der Webanwendung des Webservers kopiert und können dort einfach über die entsprechende

URL aufgerufen werden und werden erst dann kompiliert, sobald die Webanwendung von einer Applikation aufgerufen wurde. Dieses automatische Deployment stellt eine relativ einfache Möglichkeit dar, Web-Services zu veröffentlichen, jedoch gibt es einige Restriktionen. Es können nur Standarddatentypen[14] innerhalb der SOAP-Message verwendet werden. Serialisierung bzw. Deserialisierung von komplexeren Datentypen ist hier nicht möglich.

Das zu empfehlende Vorgehen für das Deployment von Web-Service ist das benutzerdefinierte Deployment mittels WSDD Dateien und dem AdminClient der von Axis mitgeliefert wird. Im Gegensatz zum automatischen Deployment bilden kompilierte Klassen die Basis für den Web-Service. Eine XML-Konfigurationsdatei[15] fasst die weiteren Installations- und Konfigurationsparameter zusammen. Über ein von Axis mitgeliefertes Servlet, dem AdminClient, wird dann der konfigurierte Web-Service in die Laufzeitumgebung des Webservers integriert. Der AdminClient selbst stellt wiederum auch einen Web-Service dar und kann somit selbst als Dienst zur entfernten Administration des veröffentlichten Web-Services verwendet werden. In beiden Fällen können Web-Services durch ein Undeployment wieder aus der Laufzeitumgebung des Webservers entfernt werden. Im Falle des automatischen Deployment geschieht dies durch Löschen der entsprechenden JWS Dateien. Das Undeployment für den benutzerdefinierten Fall wird durch eigene Undeployment Deskriptoren realisiert.

Die Wahl des Deployment-Verfahrens hängt von den Anforderungen an den Web-Service ab. JWS-Dateien stellen eine einfache Möglichkeit zur Verfügung Web-Services zu veröffentlichen, jedoch besteht bei diesem Deployment kein Einfluss auf die Konfiguration der Axis-Mechanismen, die im weiteren Verlauf als Axis Engine definiert wird. Es werden stets alle Methoden einer Klasse veröffentlicht. Weiterhin ist der Quelltext des Web-Services all denen frei zugänglich, die Zugriff auf das Dateisystem des Webservers haben. Alle diese Einschränkungen besitzt das Deployment mittels Deskriptoren nicht. Eine komplette Referenz der Web-Service-Deployment-Deskriptoren ist in [18] nachzulesen. Ein weiteres Merkmal, welches zum flexiblen Einsatz von Axis beiträgt ist dessen Architektur. Axis besteht aus mehreren Subsystemen die zusammenarbeiten um die Gesamtfunktionalität zu erbringen. Den Kern dieses Systems bildet die Axis Engine. Innerhalb dieser Engine werden zum einen die SOAP-Nachrichten verar-

[14] Der SOAP-Standard spezifiziert einige elementaren Datentypen, beispielsweise einen String

[15] Gewöhnlich tragen die Konfigurationsdateien auch die Endung .wsdd

beitet und zum anderen fungiert sie die Koordinator von Nachrichten innerhalb des Systems. Axis kann sowohl auf dem Client wie auch dem Server eingesetzt werden. Auf Clientseite werden mit Hilfe der Axis API, SOAP-Nachrichten erstellt und versendet. Auf der Serverseite können die erhaltenden Informationen der jeweiligen Dienst-Implementierung übergeben werden. Anschließend wird dann die Antwort an den Client zurücktransferiert. Zur Verarbeitung einer SOAP-Nachricht ist nicht nur eine Kernkomponente verantwortlich. Die Axis Engine ruft sogenannte *Handler* auf, die von der Nachricht nacheinander durchlaufen werden. Jedem Handler wird ein Message Context übergeben der neben der Request-Nachricht (bzw. Response-Nachricht) auch verschiedene Details über den Aufruf enthält, wie z.B. das verwendete Transportprotokoll. Jeder Handler kann nun innerhalb seiner Aufgaben die Request- bzw. Response-Nachricht verarbeiten bzw. manipulieren. Eine geordnete Menge von Handlern nennt man *Chains*. Eine Chain ist ein spezieller Handler der mehrere Handler hintereinander aufruft. Das Erzeugen des Message Contextes geschieht auf Clientseite nicht automatisch durch die Axis-Engine, sondern durch ein Objekt der Klasse Call. Auf dem Objekt werden Endpunkt, Parameterlisten etc. gesetzt, bevor die Methode invoke() dann den Message Context an die Axis-Engine weitergibt. Abbildung 3-3 stellt die Axis Engine am Beispiel des Servers dar.

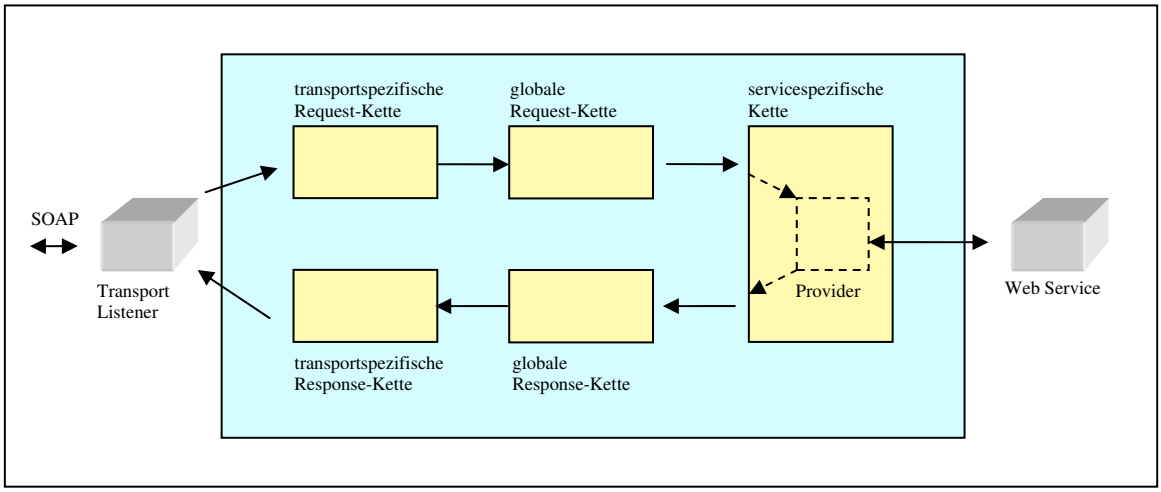


Abbildung 3-3 Ablauf einer Nachricht auf dem Server

Die Nachricht wird durch drei verschiedene Ketten geleitet. Die *transportspezifische Kette* ruft die Handler die mit einem speziellen Transportprotokoll assoziiert sind, d.h. der weitere Ablauf der Nachricht kann vom verwendeten Protokoll (HTTP, SMTP etc.) abhängen. Die globalen Handler innerhalb der *globalen Request-Kette* werden immer aufgerufen unabhängig davon welcher Web-Service angesprochen wird. Die *dienstspe-*

zifische Kette enthält die Handler die zum Einsatz kommen, wenn ein spezifischer Web-Service aufgerufen wird. Für jeden Web-Service kann somit eine spezifische Kette konfiguriert werden. Innerhalb der serverspezifischen Kette wird dann die SOAP-Nachricht an einen Provider übergeben, der dann mit der Realisierung des Web-Servies interagiert. Eine analoge dreistufige Hierarchie ist in der Client-Seite der Axis Engine enthalten. Axis stellt damit einen mächtigen modularen Mechanismus zur Verfügung der variabel den Ablauf und die Verarbeitung von SOAP-Nachrichten bestimmen kann und zudem erweiterbar ist. In der Axis-Distribution mitgelieferte Beispiel-Handler können leicht zum Entwickeln eigener Handler adaptiert werden.

4 UNICORE Authorization Service (UAS)

Dieses Kapitel bildet den entwicklungtechnischen Kern der Diplomarbeit. Das Ziel der Entwicklung ist es, eine Autorisierungsfunktionalität als Web-Service zur Verfügung zu stellen. Die Entwicklung dieser Grid-Autorisierungskomponente entsteht im Rahmen des UniGrids Projekts, welches eine WS-RF-konforme Infrastruktur für Grid-Dienste entwickelt. Die konzeptionellen wie auch implementierungstechnischen Anforderungen an den UAS resultieren im Wesentlichen aus den Forderungen, welche die UNICORE Architektur an ein Autorisierungskonzept richtet. In Kapitel 2.1.2 und 2.1.3 ist die bisherige Realisierung beschrieben worden und deren Nachteile in Bezug auf eine vielseitig verwendbare Komponenten. Kapitel 4.1 fasst die wesentlichen Nachteile zusammen und beschreibt die Anforderungen und das Konzept der Neuentwicklung. Nachdem in das Konzept eingeführt wurde, wird detaillierter auf die Implementierung des UAS in Java eingegangen. Dies umfasst eine Beschreibung der unterschiedlichen Bibliotheken, welche zum Einsatz kommen, sowie eine Übersicht über die verwendeten Schichten, die innerhalb des Autorisierungsdiensts eigene Funktionalitäten haben. Weiterhin werden die Interfaces beschrieben, welche die Funktionalität des Dienstes nach außen zur Verfügung stellen und somit direkt ein WSDL-Dokument zur Nutzung des UAS definieren. Kapitel 4.3 beinhaltet detaillierte Spezifikation zur Administration des UAS in Bezug auf Anforderungen, die im Rahmen einer Verwaltung der Autorisierungsinformationen entstehen. Da sicherheitskritische Informationen über ein möglicherweise unsicheres Transportprotokolle übertragen werden muss eine sichere Kommunikation gewährleistet werden. Kapitel 4.3 beschreibt zwei unterschiedliche Ansätze zur Sicherung eines Web-

Services, von denen aber nur der erste innerhalb dieser Arbeit realisiert wird. Im Rahmen der Verwendung des Web-Services bedarf es noch einiger Konfigurationen der verwendeten Komponenten bzw. Mechanismen[16] die im abschließenden Kapitel beschrieben sind.

4.1 Anforderungen an den UAS

Grundlage für die Anforderungen bildet das UNICORE-Sicherheitsmodell, welches ausgehen von X.509 Zertifikaten Zugangsinformationen auf einem Zielsystem zur Verfügung stellt. Dieses stellt aber keine Restriktion der Autorisierungskennzeichen an ein X.509 Zertifikat dar. Für eine flexible Nutzung kann soll innerhalb des Autorisierungsdienstes auch jede andere Form der Darstellung von Autorisierungsinformationen möglich sein. Analog gilt dieses auch für die logische Verwaltung der Zugangsinformationen, die hier aus einer Rolle und einem Benutzer-Login auf dem Zielsystem bestehen, dies aber in anderen Umgebungen nicht zwangsläufig müssen.
Im Folgenden soll nun aufgezeigt werden, wie die Nachteile der ursprünglichen Realisierung direkt in das Konzept der Neuentwicklung eingegangen sind. Die ausführliche Beschreibung der Restriktionen wurde in Kapitel 2.1.3 vorgenommen:

Einschränkung 1: Die Datenhaltung der UUDB erfolgt in einem serialisierten Java-Objekts, welches binär in eine Datei geschrieben wird und somit nicht editiert werden kann.

Konzept 1: Das XML-Format hat sich schon bei der Beschreibung der Web-Service Komponenten (SOAP, WSDL und UUDI) als leistungsfähiges Format zur Repräsentation von Daten herausgestellt. Zudem ist es in seiner Beschreibung nicht an eine Programmiersprache, Architektur etc. gebunden. Weiterhin ist eine XML-Datei textbasiert und kann somit problemlos gelesen und auch editiert werden. Ein weiterer nützlicher Aspekt der XML-Darstellung von Daten ist, dass es Standardisierungsentwürfe wie z.B. XML-Encryption [19] gibt mit deren Hilfe komplette XML Dokumenten kryptographisch gesichert werden können und die somit auch ein Verschlüsseln auf Daten-

[16] Mechanismus meint hier den Deployment-Vorgang von Web-Services

ebene zur Verfügung stellen. Die Neuentwicklung stellt die Autorisierungsinformationen und Zugangsinformationen innerhalb eines XML-Dokuments zur Verfügung.

Einschränkung 2: Die Datenhaltung und das Verwalten der Autorisierungsinformationen sind gemischt und nicht deutlich von einander getrennt.

Konzept 2: Es werden zwei neue Schichten eingeführt die ihre Funktionalität von einander unabhängig über definierte Schnittstellen zur Verfügung stellen. Die *Logik-Schicht* stellt über ihre Methoden die Schnittstelle des Web-Services zur Verfügung und ist getrennt in einen *Public*-Teil, welcher den reinen Autorisierungsprozess[17] beinhaltet und einen *Admin*-Teil, der für die Verwaltung[18] der Zertifikate verantwortlich ist. Die Logikschicht kommuniziert über Schnittstellen mit der *Speicher-Schicht*, welche für die separate Datenhaltung in Form eines XML-Dokuments verantwortlich ist. Es existiert somit eine strikte Trennung zwischen den Aufgaben der Logik-Schicht die Anfragen bearbeitet und einer Speicher-Schicht die Informationen physisch in ein Dateisystem schreibt.

Einschränkung 3: Die UUDB ist innerhalb des UNICORE-Servers mit dem NJS verbunden und kann nicht außerhalb dieses Kontextes z.B. auf einem anderen Rechner laufen.

Konzept 3: Durch die Web-Service-Technologie kann die Funktionalität eines Autorisierungsdienstes zentral zur Verfügung gestellt werden und ist nicht länger an das Serversystem einer Vsite gebunden.

Einschränkung 4: Es existieren keine Schnittstellen zur UUDB die durch andere Komponenten innerhalb des Grid zum Abrufen von Autorisierungsinformationen verwendet werden können.

Konzept 4: Dieser Nachteil entsteht dadurch, dass die UUDB innerhalb des NJS fest integriert ist und es somit keiner zusätzlichen Schnittstellen bedarf, welche Autorisie-

[17] Autorisierungsprozess meint hier den Verweis es X.509 Zertifikats auf die Zugangsinformationen

[18] Verwaltung bezieht sich auf Hinzufügen, Löschen etc. von X.509 Zertifikaten

rungsinformationen anderen Komponenten außerhalb des NJS im Grid zur Verfügung stellen. Durch das Herauslösen der Autorisierungsfunktionalität in einen eigenständigen Dienst, der dezentral zur Verfügung steht, ist es notwendig geworden, detaillierte Schnittstellen zur Nutzung des Autorisierungsdienstes zur Verfügung zu stellen, die dann innerhalb eines WSDL-Dokuments veröffentlicht werden. Diese Schnittstellen können dann weiterhin vom NJS angesprochen werden, aber ebenso von anderen Komponenten innerhalb des Grids, die eine Autorisierung benötigen.

Einschränkung 5: Ein entferntes Administrieren der UUDB ist nicht möglich. Sie kann nur auf dem entsprechenden System verwaltet werden.

Konzept 5: Die Logik-Schicht implementiert eine eigene Admin-Schnittstelle. Innerhalb dieser Schnittstelle wird die elementare Funktionalität, wie das Löschen, Hinzufügen etc. von Zertifikaten unterstützt. Diese Schnittstelle steht als Web-Sercie nach außen zur Verfügung und kann somit zum entfernten Administrieren verwendet werden.

Einschränkung 6: Die UNICORE-Architektur gibt vor, dass jede Vsite genau eine UUDB zur Verfügung stellen muss, um die Autorisierungsinformationen zu verwalten.

Konzept 6: Der UAS stellt die Möglichkeit zur Verfügung innerhalb des Datenmodells die Autorisierungsinformationen mehrerer Vsites zu verwalten. Um eine eindeutige Zuordnung der Zugangsinformationen zur korrespondierenden Vsite herzustellen muss eine Vsite spezifische Identifizierung den zu speichernden Informationen zugefügt werden. Im Laufe einer Autorisierungsanfrage identifiziert sich der NJS über einen Parameter und kann so die für ihn wichtigen Autorisierungsinformationen über den UAS erhalten.

4.2 Implementierung des UAS

Ausgehend von den Anforderungen, welche im vorangegangen Kapitel beschrieben wurden, lässt sich nun mit Abbildung 4-1 der UAS schematisch darstellen.

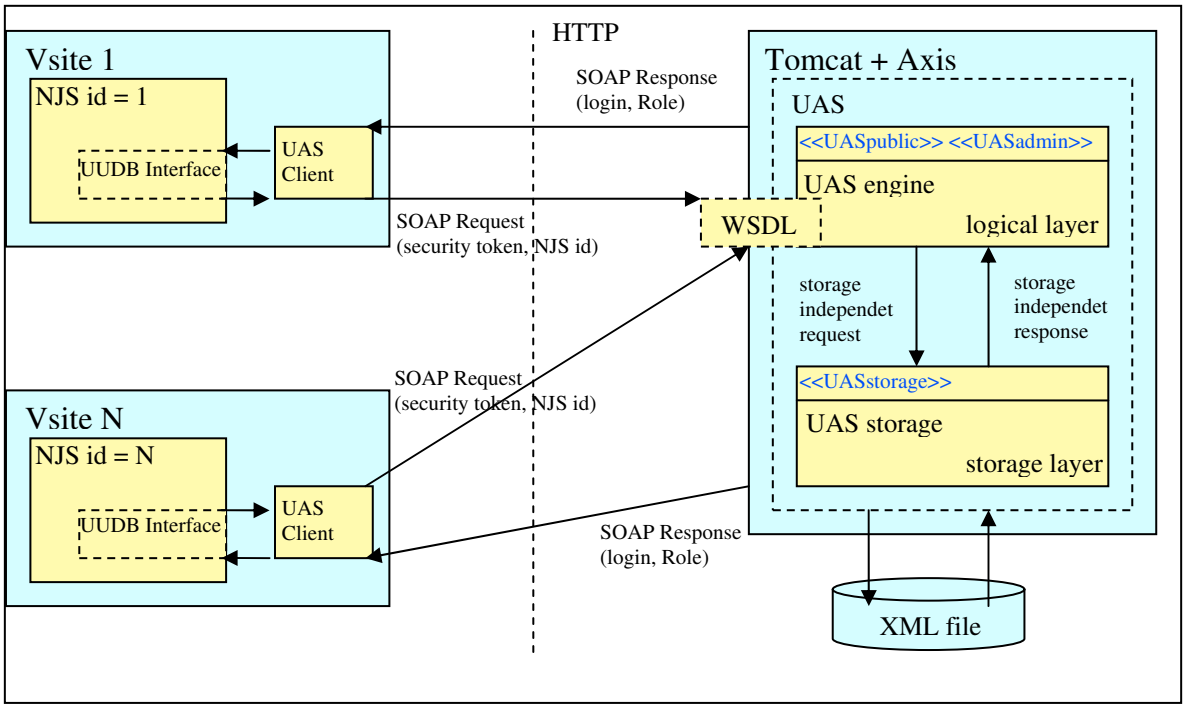


Abbildung 4-1 Schematische Darstellung des UAS

Der NJS gibt die Implementierung der UUDB[19] nicht zwingend vor. Innerhalb der Schnittstellenbeschreibungen des NJS existieren innerhalb einer abstrakten UUDB-Basisklasse Vorgaben, welche Methoden abgeleitete Klassen implementieren müssen, damit der NJS die notwendigen Autorisierungsinformationen über die entsprechenden Methodenaufrufe erhält. Innerhalb der Konfigurationsdateien des NJS kann dann optional die Nutzung einer separaten UUDB definiert werden. Weiterhin muss innerhalb der Konfiguration des NJS eine eindeutige Identifizierung festgelegt werden über die er die, für sich relevanten Autorisierungsinformationen, erhält. Dieser SOAP-Request wird nun über das HTTP-Protokoll[20] an den UAS gerichtet. Die *UAS engine* implementiert zwei verschiedene Schnittstellenbeschreibungen. Die *UASPublic* Schnittstelle beschreibt die Methoden die z.B. vom NJS aufgerufen werden können um die notwendigen Autorisisierungsinformationen in Abhängigkeit des gegebenen Zertifikats zu erhalten. Die *UA-*

[19] Es wird hier noch die NJS-spezifische Bezeichnung *UUDB* anstelle der Bezeichnung *Autorisierungsfunktionalität* verwendet

[20] In Kapitel 4.3 wird die sichere Nutzung über HTTPS erwähnt

SAdmin Schnittstelle implementiert die Methoden die zum Verwalten des UAS benötigt werden, wie z.B. das Hinzufügen von Zertifikaten. Diese Ebene besitzt eine logische Funktionalität, indem sie bspw. überprüft ob ein hinzuzufügendes Zertifikat bereits vorhanden ist bevor es zur Datenhaltung zugefügt wird. Weiterhin besteht die Aufgabe dieser Schicht in der Fehlerbehandlung. Es werden hier logische Fehler und fatale Fehler unterschieden. Ein logischer Fehler kann beispielsweise durch ein Hinzufügen eines bereits vorhandenen Zertifikats verursacht werden. Ein fataler Fehler ist ein Fehler, der innerhalb der Speicherebene entstanden ist und von der aufrufenden Schicht abgefangen wurde. Die Speicherschicht wird durch die *UAS storage* repräsentiert, die eine Schnittstelle *UASStorage* implementiert, die die zur Speicherung notwendigen Methoden vorgibt. Die physische Datenspeicherung erfolgt dann als XML-Dokument in eine Datei.

4.2.1 Verwendete Bibliotheken

Für die Implementierung des Autorisierungsdienstes bedarf es einer Vielzahl von Bibliotheken, die verwendet werden. Diese Bibliotheken werden von den verschiedenen Komponenten beispielsweise für Anfagen der Axis Engine verwendet. Tabelle 2 listet die verwendeten Bibliotheken auf und beschreibt sie kurz.

Bibliothek	*Beschreibung*
ajo.jar	Diese Bibliothek beinhaltet die notwendigen Klassen, welche die UNICORE-spezifische Darstellung von Informationen beinhalte, beispielsweise. einen UNICORE User.
axis-ant.jar	Für das Deployment eines Web-Services existieren eigene Ant[21]-XML-Beschreibungen, die in dieser Bibliothek enthalten sind.
axis.jar	Hierin sind alle nötigen Komponenten enthalten die innerhalb der Axis Engine benötigt werden um die SOAP-Nachrichten zu erstellen und zu bearbeiten.
commons-discovery-0.2.jar	Diese Komponente lokalisiert Klassen die ein bestimmtes JAVA-Inteface implementieren und wird von der Axis benötigt.
commons-logging-1.0.4.jar	Innerhalb der *commons*-Komponenten wird diese Bibliothek für die Erfassung von Informationen verwendet. Sie wird ebenfalls von Axis benötigt.
jaxrpc.jar	Diese Bibliothek stellt eine Java Schnittstellenbeschreibung für XML-basierte Remote Procedure Calls dar die innerhalb der SOAP-Nachricht übertragen werden.

[21] Ant ist ein make-ähnliches Werkzeug, welches zum automatisierten Erzeugen von Programmen verwendet wird

jdom.jar	Mit Hilfe dieser Bibliothek können XML-basierte Dokumente bearbeitet und ausgegeben werden. Sie wird für die Bearbeitung der Datenhaltung innerhalb der Speicher-Schicht verwendet.
junit.jar	Junit stellt zum Testen der einzelnen Komponenten ein Test-Framework bereit mit dem die einzelnen Funktionalitäten überprüft werden können.
log4j-1.2.8.jar	Über eine Konfigurationsdatei können zur Laufzeit log-Informationen für ein Debbuging zur Verfügung gestellt werden ohne dazu den Quelltext zu ändern.
njs_interfaces.jar	Dieses sind die Schnittstellenbeschreibungen des NJS die eine separate UUDB implementieren muss. Hierbei wird gewährleistet, dass der NJS die erhaltenen Autorisierungsinformationen verarbeiten kann.
saaj.jar	Die saaj-Bibliothek stellt eine standardisierte Möglichkeit zur Verfügung um XML-Dokumente über das Internet zu versenden. Sie unterstützt den Austausch der SOAP-Nachrichten.
wsdl4j-1.5.1.jar	Diese Bibliothek stellt Funktionalitäten bereit, die das Erstellen, die Repräsentation und die Verarbeitung von WSDL-Dokumenten ermöglichen.

Tabelle 2 Innerhalb des UAS verwendete Bibliotheken

4.2.2 Struktur des UAS package

Der UAS besteht aus einem Hauptpaket und mehreren Unterpaketen. Abbildung 4-2 beinhaltet eine schematische Darstellung des UAS package.

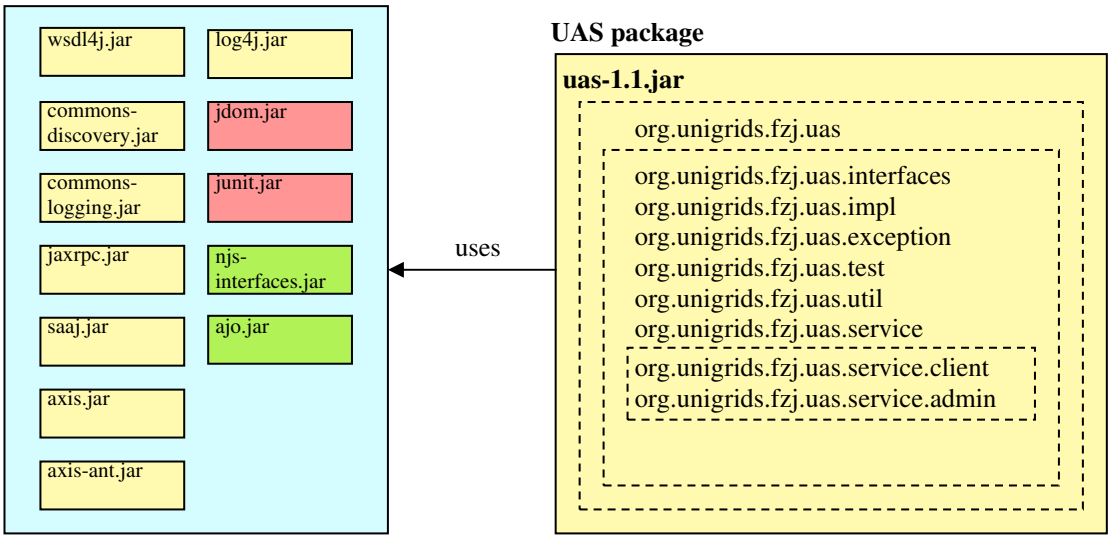


Abbildung 4-2 Struktur des UAS package

Die Inhalte bzw. Aufgaben und Funktionalitäten der einzelnen Unterpakete werden im Folgenden detaillierter ausgeführt:

- `org.unigrids.fzj.uas.interface.`
  Der UAS definiert sich über die folgenden Schnittstellen, sie werden deshalb im Kapitel 4.2.3 genauer vorgestellt.
  - `UASPublic`
    Die UASPublic Schnittstelle gibt die öffentlichen Methoden vor, die vom NJS zum Abfragen der Autorisierungsinformationen benötigt werden.
  - `UASAdmin`
    Innerhalb der UASAdmin Schnittstelle werden die Methoden zum Administrieren des Autorisierungsinformationen definiert.
  - `UASStorage`
    Der UAS ist unabhängig von der verwendeten Form der Speicherschicht. Um die elementaren Operationen ausführen zu können muss die UASStorage Schnittstelle implementiert werden.
- `org.unigrids.fzj.uas.impl.`
  Innerhalb dieses Unterpakets wird die Funktionalität der einzelnen Schichten abgebildet. Da die beiden Schichten eine zentrale Rolle innerhalb des UAS spielen werden sie in Kapitel 4.2.4 und Kapitel 4.2.5 genauer beschrieben.
  - `UASEntry`
    UASEntry realisiert einen Container über den die Informationen eines UAS Eintrag zwischen der Logikschicht und der Speicherschicht ausgetauscht werden können. Da die Informationsübertragung zwischen den Schichten unabhängig von der Implementierung der Speicherschicht realisiert werden soll, muss dieser Container verwendet werden.
  - `UASStorageXML`
    Innerhalb der Implementierung der UASStorageXML wird die Verwaltung und Speicherung aller Autorisierungsinformationen innerhalb eines XML-Dokuments realisiert.
  - `UASEngine`
    UASEngine ist die Implementierung der Logikschicht, welche die logische Funktionalität zum Verwalten der Autorisierungsinformationen innerhalb des AUS zur Verfügung stellt.
- `org.unigrids.fzj.uas.exception.`
  Dieses Unterpaket definiert eine Exception-Hierarchie von UAS Fehlern, die sowohl auf der Speicherschicht wie auch innerhalb der Logikschicht auftreten können.

Die Fehlerinformationen werden innerhalb der SOAP-Nachricht als fault-Element übertragen.

- `UASException`
  Oberste Instanz eines UAS Fehler. Alle weiteren Fehler werden von dieser Basisklasse abgeleitet.
- `UASStorageException`
  Alle Fehler die Innerhalb der Datenhaltung bzw. Speicherung auftreten, werden abgefangen und als UASStorageException an die aufrufenden Methoden der Logikschicht weitergeworfen.
- `UASLogicalException`
  Logische Fehler treten bspw. auf, wenn ein hinzuzufügendes Zertifikat schon existiert. Diese Art Fehler wird abgefangen und an über den Web-Service zum aufrufenden Client übertragen.
- `UASFatalException`
  Fatale Fehler sind Fehler, die von der Speicherschicht in Form von UASStorageExceptions geworfen werden. Fehler die innerhalb der Datenverarbeitung bzw. Speicherung auftreten werden als fatale Fehler von der Logikschicht behandelt und an den Client weitergegeben.

- `org.unigrids.fzj.uas.test.`
  Innerhalb dieses Unterpaket wurde unabhängig von der gesamten Implementierung die fehlerfreie Realisierung der elementaren Komponenten UASEngine, UASStorage und UASEntry getestet. Diese Tests laufen innerhalb einer Testsuite, die vom Junit-Framework bereitgestellt wird, und verifizieren die korrekte Funktionalität der Implementierung.
- `org.unigrids.fzj.uas.util.`
  Dieses Unterpaket enthält Werkzeuge die innerhalb der Implementierung benötigt werden, aber keine Funktionalität nach außen bieten.
  - `CertficateParser`
    Dieses Werzeug erstellt aus einem Eingabestrom der den öffentlichen Schlüssel eines Zertifikats im PEM-Format enthält, eine X.509 Zertifikat und überprüft dessen Gültigkeit.
- `org.unigrids.fzj.uas.service.`
  In diesem Unterpaket wird sowohl die Clientfunktionalität zur Verfügung gestellt mir der z.B. der NJS die notwendigen Zugangsinformationen erhält. Zum anderen

wird eine Vielzahl an Client-Klassen angeboten, die zur entfernten Administration des Dienstes verwendet werden können.

- `org.unigrids.fzj.uas.service.client.`
  Diese Klassen werden vom NJS benötigt, da dieser zu weiteren Verarbeitung definierte Objekte wie IncarnatedUlogin und IncarnatedUser benötigt.
  - `UASClient`
    Die Aufgabe des UASClient ist unmittelbar aus Abbildung 4-1 zu erkennen.
  - `IncarnatedUloginImpl`
    Über eine NJS-Schnittstelle IncarnatedUlogin gibt der NJS die Implementierung dir Klasse vor.
  - `IncarnatedUserImpl`
    Hier wird ebenfalls über die NJS-Schnittstelle IncarnatedUser vom NJS die Implementierung vorgegeben.
- `org.unigrids.fzj.uas.service.admin.`
  Innerhalb des Admin-Unterpakets sind Clientklassen realisiert die ausschließlich zur Administration der Datenhaltung des UAS benötigt werden. Perlskripte nutzen die Funktionalität dieser Clientklassen und bieten so einen Kommandozeilenzugang zum UAS. Dies ist im Detail in Kapitel 4.3 nachzulesen.
  - `AddLogin`
    Diese Komponente wird zum Hinzufügen von Autorisierungsinformationen zum UAS benötigt.
  - `RemoveLogin`
    Diese Komponente wird zum Löschen von Autorisierungsinformationen zum UAS benötigt.
  - `UpdateLogin`
    Diese Komponente wird zum Ändern von Autorisierungsinformationen zum UAS benötigt.
  - `CheckLogin`
    Innerhalb dieser Komponente wird ein Zertifikat überprüft und die Zugangsinformationen ermittelt.
  - `ListLogin`
    Diese Komponente listet die Autorisierungsinformationen die innerhalb des UAS gespeichert sind auf.

- `ListPEMFilesLogin`
  Mit dieser Komponente ist es möglich Zertifikate im PEM Format vom UAS zum Client zu übertragen.

4.2.3 UAS Interfaces

Innerhalb der Implementierung des UAS existieren drei wichtige Schnittstellenbeschreibungen. Die UASPublic- und die UASAdmin-Schnittstelle beschreiben die Funktionalität des Autorisierungsdienstes nach außen. Die WSDL Spezifikation des UAS lässt sich direkt aus den beiden Schnittstellenbeschreibungen ableiten. Die UASStorage-Schnittstelle legt die Zugriffsmöglichkeiten der Logikschicht auf die Speicherschicht fest. Die Bedeutung der einzelnen Parameter der Schnittstellenbeschreibung ist in Tabelle 3 beschrieben Die Spezifikationen der einzelnen Schnittstellenbeschreibungen sind folgende:

- `UASPublic`

```
String checkSecurityToken(String gcId, String securityToken)
```

Ein Beispielaufruf für diese Schnittstelle sieht folgendermaßen aus:

```
checkSecurityToken("zam288:4444", "MIIEMTCCAxmgAwIBAgIBADANBgkqhkiG9w0BAQQFADCBp...")
```

Die gcId ist die eindeutige Identifizierung der Vsite und wird hier durch den Hostname und Port definiert. Das securityToken ist der öffentliche Schlüssel im PEM-Format.
Innerhalb des UASPublic Interface werden die Methoden definiert, welche zur Autorisierung von Nutzern des Grids benötigt werden. Diese Schnittstelle kann um weitere Methoden ergänzt werden. Der UASClient erhält vom NJS eine Autorisierungsanfrage. Ein Nutzer wird über seinen öffentlichen Schlüssel autorisiert. Dieser liegt im PEM-Format vor und wird neben der eindeutigen Kennung des NJS an den UAS weitergeleitet. In Abhängigkeit der eindeutigen NJS Identifizierung werden dann die entsprechenden Autorisierungsinformationen bestehend aus Xlogin, Role und einem optionalem Project über den Web-Service Aufruf an den UASClient weitergeleitet, der dann die eigentliche Anfrage des NJS bedient.

- `UASAdmin`

```
void removeSecurityToken(String gcId, String securityToken)
void addSecurityToken(Sring gcId, String securiyToken, String role, Sring xlogin)
void addSecurityToken(Sring gcId, String securiyToken, String role, Sring xlogin, String projects)
void addSecurityToken(Sring gcId, String securiyToken, String role, Sring xlogin, String[ ] projects)
void updateSecurityToken(String gcId, String securityToken, String role, String xlogin)
void updateSecurityToken(String gcId, String oldSecurityToken,  String newSecurityToken)
void updateSecurityToken(String gcId, String securityToken, String role, String xlogin, String[ ] projects)
String[ ] listUAS(String search)
```

Beispielaufrufe für diese Schnittstelle sehen folgendermaßen aus:

```
removeSecurityToken("zam288:4444", "MIIEMTCCAxmgAwIBAgIBADANB…")
addSecurityToken("zam288:4444", "MIIEMTCCAxmgAwIBAgIBADANB…","User","latour")
addSecurityToken("zam288:4444", "MIIEMTCCAxmgAwIBAgIBADANB…","User","latour","Project Uni-
Grids")
updateSecurityToken("zam288:4444", "MIIEMTCCAxmgAwIBAgIBADANB…","Administrator","latour")
updateSecurityToken("zam288:4444", "MIIEMTCCAxmgAwIBAgIBADANB…","AFnwrtTZWfjnwrttwhREW")
listUAS("User")
```

Die einzelnen Parameter sind in Tabelle 3 ausführlich beschrieben.

Die Logikschicht des UAS implementiert neben der UASPublic Schnittstelle auch die UASAdmin Schnittstelle. Im Gegensatz zur UASPublic Schnittstellenbeschreibung ist diese ausschließlich administrativen Zwecken vorbehalten. Diese Schnittstelle kann ebenfalls erweitert werden. Momentan enthält sie die Grundfunktionalität, die innerhalb der Administration des UAS benötigt wird. Diese Schnittstellenbeschreibung ist direkter Teil der WSDL-Beschreibung des Dienstes und kann somit zum entfernten Administrieren des UAS eingesetzt werden. Zu den Basisoperationen gehört das Hinzufügen, das Löschen, das Aktualisieren und das Auflisten von Zertifikaten, die im PEM-Format in der Datenhaltung des UAS abgelegt sind. Abhängig von einer variablen Anzahl an Parametern existieren diese Operationen in verschiedenen Ausprägungen.

- `UASStorage`

```
void add(UASEntry entry)
void remove(UASEntry entry)
void update(UASEntry oldEntry, UASEntry newEntry)
UASEntry[ ] getUASEntry(UASEntry entry)
boolean hasEntry(UASEntry entry)
int getSize()
```

UASStorage ist eine Schnittstelle, die nicht mehr innerhalb der WSDL-Spezifikationen erscheint. Diese Schnittstelle wird von der Speicherschicht implementiert und realisiert einen, von der spezifischen Art der Speicherung, unabhängigen Austausch von Informationen mit der Logikschicht. Basis für den Informationsaustausch ist das Objekt UASEntry, der als Container die notwendigen Informationen zur Speicherung beinhaltet bzw. die Informationen an die aufrufende Logikschicht zurückgibt. Da die explizite Form der Speicherung nicht vorgegeben ist, muss definiert werden, wie innerhalb der konkreten Speicherimplementierung Informationen gelesen, gelöscht und aktualisiert werden.

Tabelle 3 beschreibt die Art und Funktion der einzelnen Parameter der Schnittstellenbeschreibung.

Parametername	*Typ*	*Beschreibung*
gcId	String	Die *grid component id* beinhaltet die eindeutige Zuordung der angeforderten Autorisierungsinformationen zu einem NJS.
securityToken	String	Das *security token* enthält den PEM kodierten Schlüssel des Nutzers der über UAS autorisiert werden soll.
role	String	Rollenzuordung innerhalb des UNICORE Konzept bspw. User, NJS etc.
login	String	Zugangskennung auf dem Zielsystem innerhalb deren Privilegien die Arbeitsanweisungen vom TSI ausgeführt werden.
project	String	Unterschiedliche *projects* erlauben eine weitere Differenzierung der Zugangsdaten, die über den *login* hinausgehen.
search	String	In Abhängigkeit dieses Suchbegriffs werden alle Autorisierungsinformationen durchsucht
entry	UASEntry	Innerhalb dieses Containers werden einzelne Autorisierungsinformationen zwischen Logikschicht und Speicherschicht ausgetauscht.

Tabelle 3 Beschreibung der Interface Parameter

4.2.4 UAS logical layer

Die Aufgabe der Logikschicht ist es die Autorisierungsanfragen bzw. Administrationsvorgänge entgegenzunehmen, die spezifischen Informationen dann in einen UASEntry Container zu geben, der dann an die Speicherschicht zur Bearbeitung weitergegeben wird.

Die genauere Vorgehensweise soll beispielhaft an der Methode removeSecurityToken verdeutlich werden, wobei alle anderen implementierten Methoden der Schnittstellen[22] ein analoges Vorgehen aufweisen. Diese Methode wird von der Schnittstelle UASAdmin vorgegeben und gehört somit zu der administrativen Funktionalität. Die Aufgabe dieser Methode ist es, ein Zertifikat eines Nutzers auf einer bestimmten Vsite zu löschen und somit dem Benutzer den Zugang zum Grid zu verweigern. Die benötigten Parameter sind das zu lösche Zertifikat und die entsprechende Identifizierung des NJS bzw. der Vsite. Abbildung 4-3 zeigt schematisch die Vorgehensweise.

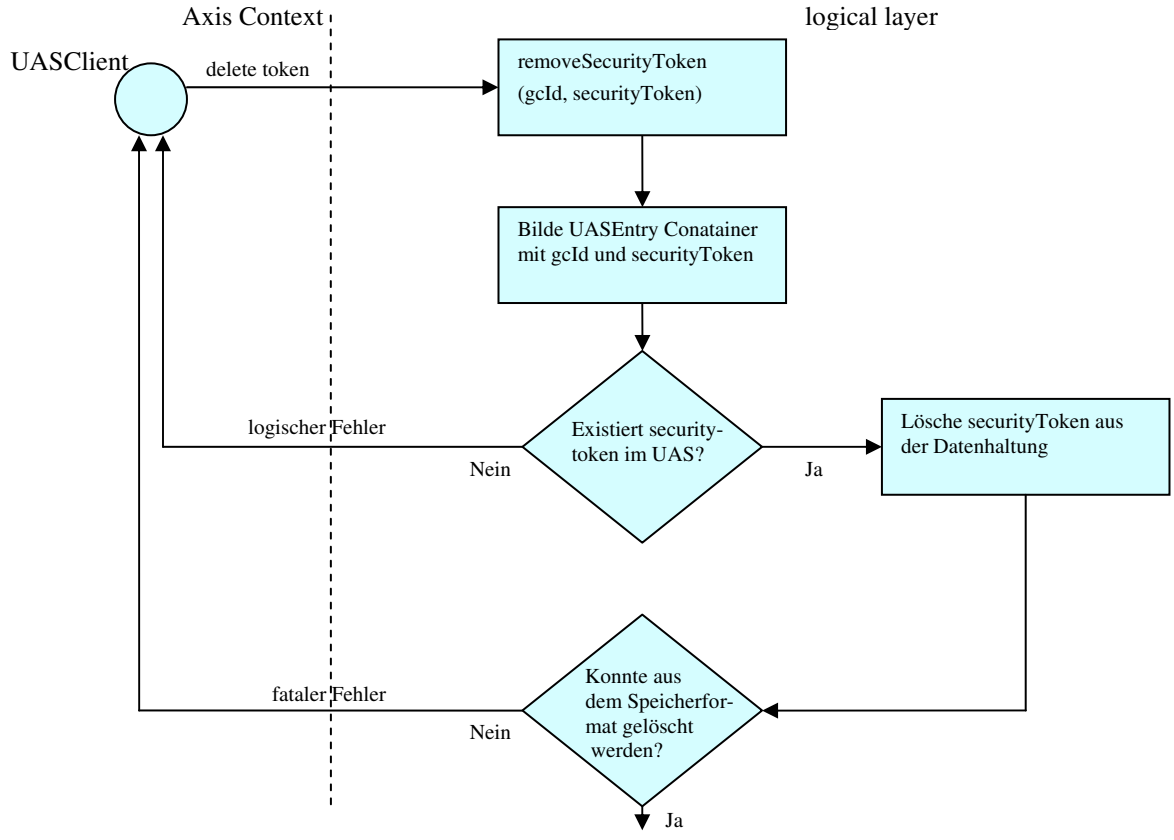


Abbildung 4-3 Schematische Vorgehensweise der Logikschicht

[22] Hier sind die Schnittstellen UASPublic und UASAdmin der Logikschicht gemeint

Der UASClient überträgt das zu löschende Zertifikat sowie die Identifikation des NJS innerhalb dessen Kontextes das Zertifikat gelöscht werden soll als Parameterlisten innerhalb eines Remote Procedure Calls zur Axis Engine auf der Serverseite. Dort wird mit den genannten Paramtern die aufgerufene Methode removeSecurityToken ausgeführt. Da die Logikschicht mit der Speicherschicht nur über UASEntry Objekte kommuniziert, muss ein UASEntry Container die Parameter aufnehmen. Es wird zuerst überprüft, ob das Tupel (gcId, securityToken) im UAS gespeichert ist. Ist dieses nicht der Fall, so wird der weitere Vorgang abgebrochen und ein logischer Fehler (Exception) über den Kommunikationskanal innerhalb einer SOAP-Nachricht zurück an den Client übermittelt. Ist das Datentupel enthalten, so kann es gelöscht werden. Dazu wird die entsprechende Methode der Speicherschicht[23] aufgerufen. Ist während des Löschvorgangs kein Fehler bzw. Problem entstanden, so ist der Löschvorgang abgeschlossen. Tritt ein Fehler auf der Speicherebene auf, so wird dieser als fataler Fehler ebenfalls an den Client zurückgesendet.

4.2.5 UAS storage layer

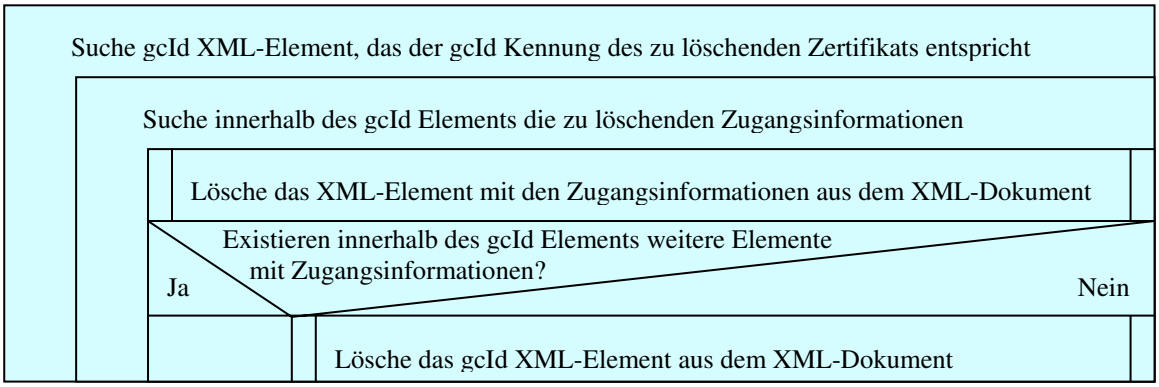
Die Speicherschicht ist für die physische Speicherung der Zugangs- bzw. Autorisierungsinformationen zuständig. Eine explizite Anforderung an den UAS ist es, dass das Speicherkonzept nicht vom Rest der Implementierung abhängt, sondern modular austauschbar ist. So können Veränderungen in der Komplexität der Autorisierungsinformationen durch eine individuelle Speicherschicht berücksichtig werden. Die Schnittstelle UASStorage gibt ausschließlich Basisoperationen vor, die von einer konkreten Speicherschichtimplementierung realisiert werden müssen, da diese Operationen von der aufrufenden Logikschicht benötigt werden.

Die in dieser Arbeit verwendete Form der Speicherung basiert auf einem XML-Dokument in dem die Zugangs- bzw. Autorisierungsinformationen gespeichert sind. XML bietet den Vorteil einer einfachen Verarbeitung die durch Programmbibliotheken[24] gewährleistet wird. Neben der selbst beschreibenden Syntax ist es ebenfalls möglich die XML Struktur bei Bedarf kryptographisch zu verschlüsseln.

[23] Die entsprechende Methode zum Löschen auf der Speicherschicht ist remove

[24] Zur Verarbeitung des XML-Dokuments wird JDOM verwendet

Die Identifikation ID des NJS bildet einen Schlüssel mit dem die Zugangsinformationen gruppiert werden, d.h. werden die alle Informationen die zu einem bestimmten NJS gehören XML-technisch gruppiert um über diesen Schlüssel anschließend die Zugangsinformationen zu finden. Ein Beispiel für ein UAS XML-Dokument ist dem Anhang zu entnehmen. Folgendes Diagramm veranschaulicht das Löschen eines Zertifikats vom Tupel (gcID, securityToken) ausgehend.



Im vorigen Kapitel wurde die Möglichkeit beschrieben, dass neben den Autorisierungsanforderungen auch eine administrative Funktionalität des UAS gewährleistet wird. Neben der Dienst-seitigen Funktionalität muss ebenfalls eine Clientkonzept existieren, welches dem Benutzer zur Administration zur Verfügung gestellt werden kann. In Kapitel 4.2.2 wurde innerhalb des Programmpakets `org.unigrids.fzj.uas.service.admin` die Clientklassen beschrieben, welche die entsprechenden Aktionen zum Verwalten der Zertifikate etc. bereitstellen. Die Clientklassen werden von Perlskripten aufgerufen. Diese Perlskripte bieten damit einen administrativen Zugang von der Kommandozeile aus zur entfernten Verwaltung der Autorisierungsinformationen. Es folgen die Spezifikationen der einzelnen Administrierungsbefehle.

4.3.1 Hinzufügen von Zertifikaten (add)

Überblick:

`add` – fügt den öffentlichen Schlüssel und die zugehörigen Zugangsinformationen dem UAS hinzu

Syntax:

```
add [ <endpoint> ] <grid component id> <pemfile> <role>
<xlogin> [ <project1> [ <project2> ] ... ]
```

Beispiel:

```
add
https://zam291.zam.kfa-juelich.de:8080/axis/services/UAS
zam291:4444 latour.pem User latour
```

Parameterbeschreibung:

Parameter	*Beschreibung*
endpoint	URL des Autorisierungsdienstes die optional definiert werden kann. Wird kein *endpoint* angegeben so wird ein Default-Wert verwendet.
grid component id	Vsite Identifizierung innerhalb derer die Zuordnung zwischen Zertifikat und Zugangsinformationen eingetragen wird.
pemfile	Vollqualifizierter Name der Datei mit dem öffentlichen Schlüssel des Benutzerzertifikats.
role	Rolle mit der ein Nutzer innerhalb der Zugangsinformationen autorisiert wird.
xlogin	Zugang zum Zielsystem mit dem ein Nutzer innerhalb der Zugangsinformationen eingetragen wird.
project	Optionale Zuordnung eines Nutzer zu (mehreren) Projekten

Tabelle 4 Parameterbeschreibung des Admin-Befehls add

4.3.2 Entfernen von Zertifikaten (remove)

Überblick:

`remove` – löscht den öffentlichen Schlüssel und die zugehörigen Zugangsinformationen aus dem UAS.

Syntax:

```
remove [ <endpoint> ] <grid component id> <pemfile>
```

Beispiel:

```
remove
https://zam291.zam.kfa-juelich.de:8080/axis/services/UAS
zam291:4444 latour.pem
```

Parameterbeschreibung:

Parameter	*Beschreibung*
endpoint	URL des Autorisierungsdienstes die optional definiert werden kann. Wird kein *endpoint* angegeben so wird ein Default-Wert verwendet.
grid component id	Vsite Identifizierung innerhalb derer die Zuordnung zwischen Zertifikat und Zugangsinformationen eingetragen wird.
pemfile	Vollqualifizierter Name der Datei mit dem öffentlichen Schlüssel des Benutzerzertifikats.

Tabelle 5 Parameterbeschreibung des Admin-Befehls remove

4.3.3 Aktualisieren von Zertifikaten (update)

Überblick:

`update` – ändert zu einem existierenden öffentlichen Schlüssel, die zugehörigen Zugangsinformationen oder tauscht einen öffentlichen Schlüssel gegen einen neuen öffentlichen Schlüssel aus.

Syntax 1:

```
update [ <endpoint> ] <grid component id> <oldpemfile>
<newpemfile>
```

Beispiel 1:

```
update
https://zam291.zam.kfa-juelich.de:8080/axis/services/UAS
zam291:4444 latour_revoke.pem latour_new.pem
```

Syntax 2:

```
update [ <endpoint> ] <grid component id> <pemfile> <role>
<xlogin> [ <project1> [ <project2> ] ... ]
```

Beispiel 2:

```
update
https://zam291.zam.kfa-juelich.de:8080/axis/services/UAS
zam291:4444 latour.pem Admin root
```

Parameterbeschreibung:

Parameter	*Beschreibung*
endpoint	URL des Autorisierungsdienstes die optional definiert werden kann. Wird kein *endpoint* angegeben so wird ein Default-Wert verwendet.
grid component id	Vsite Identifizierung innerhalb derer die Zuordnung zwischen Zertifikat und Zugangsinformationen eingetragen wird.
pemfile	Vollqualifizierter Name der Datei mit dem öffentlichen Schlüssel des Benutzerzertifikats.
role	Rolle mit der ein Nutzer innerhalb der Zugangsinformationen autorisiert wird.
xlogin	Zugang zum Zielsystem mit dem ein Nutzer innerhalb der Zugangsinformationen eingetragen wird.
project	Optionale Zuordnung eines Nutzer zu (mehreren) Projekten
oldpemfile	Vollqualifizierter Name der Datei mit dem zu ersetzenden öffentlichen Schlüssel des Benutzerzertifikats.
newpemfile	Vollqualifizierter Name der Datei mit dem öffentlichen Schlüssel des Benutzerzertifikats, welcher *oldpemfile* ersetzt.

Tabelle 6 Parameterbeschreibung des Admin-Befehls update

4.3.4 Überprüfen von Zertifikaten (check)

Überblick:

`check` – überprüft ob ein Zertifikat innerhalb des UAS existiert und listet die entsprechenden Zugangsinformationen auf.

Syntax:

`check [ <endpoint> ] <grid component id> <pemfile>`

Beispiel:

```
check
https://zam291.zam.kfa-juelich.de:8080/axis/services/UAS
zam291:4444 latour.pem
```

Parameterbeschreibung:

Parameter	*Beschreibung*
endpoint	URL des Autorisierungsdienstes die optional definiert werden kann. Wird kein *endpoint* angegeben so wird ein Default-Wert verwendet.
grid component id	Vsite Identifizierung innerhalb derer die Zuordnung zwischen Zertifikat und Zugangsinformationen eingetragen wird.
pemfile	Vollqualifizierter Name der Datei mit dem öffentlichen Schlüssel des Benutzerzertifikats.

Tabelle 7 Parameterbeschreibung des Admin-Befehls check

4.3.5 Durchsuchen des UAS nach Begriffen (list)

Überblick:

`list` – durchsucht den UAS nach einem Schlüsselwort und liefert zu allen gefunden Einträgen die Zugangsinformationen.

Syntax:

`list [ <endpoint> ] <keyword>`

Beispiel:

```
list
https://zam291.zam.kfa-juelich.de:8080/axis/services/UAS
User
```

Parameterbeschreibung:

Parameter	*Beschreibung*
endpoint	URL des Autorisierungsdienstes die optional definiert werden kann. Wird kein *endpoint* angegeben so wird ein Default-Wert verwendet.
keyword	Schlüsselwort für die zu suchenden Einträge im UAS

Tabelle 8 Parameterbeschreibung des Admin-Befehls list

4.3.6 Auflisten aller Zertifikate eines Suchbegriffs (list_pem_files)

Überblick:

`list_pem_files` – identische Funktionsweise wie `list`, mit der Eigenschaft, dass zusätzlich zu allen gefunden Informationen die jeweiligen Zertifikate zum Client übertragen werden und dort ins Dateisystem geschrieben werden.

Syntax:

`list_pem_files [ <endpoint> ] <keyword> <outputdir>`

Beispiel:

```
list_pem_files
https://zam291.zam.kfa-juelich.de:8080/axis/services/UAS
zam291:4444 ../certs
```

Parameterbeschreibung:

Parameter	*Beschreibung*
endpoint	URL des Autorisierungsdienstes die optional definiert werden kann. Wird kein *endpoint* angegeben so wird ein Default-Wert verwendet.
keyword	Schlüsselwort für die zu suchenden Einträge im UAS
outputdir	Verzeichnis, in dem die Zertifikate gespeichert werden

Tabelle 9 Parameterbeschreibung des Admin-Befehls list_pem_files

4.4 Sicherheit des UAS

Innerhalb des UAS ist es mögliche Zertifikate und Zugangsinformationen zu administrieren und Zugangsinformationen bzw. Autorisierungsinformationen abzurufen. Sowohl das Administrieren, wie auch das Abfragen von Zertifikatsinformationen stellt einen sicherheitskritischen Vorgang dar und kann durch Man-in-the-middle-Attacken [20] manipuliert werden. Grundsätzlich gibt es verschiedene Möglichkeiten eine sichere Kommunikation des Client mit dem Web-Service zu gewährleisten. Die eigentliche Aufgabe der Sicherung des Web-Service besteht darin die SOAP-Nachrichten, die zwi-

schen dem Client und dem Dienst ausgetauscht werden[25] vor unbefugter Einsicht bzw. Veränderung zu schützen.
Hierzu gibt es verschiedene Ansätze. Innerhalb des OSI-Modells kann diese Sicherheit auf Transportschicht angeordnet sein in Form einer gesicherten Verbindung des Kommunikationskanals oder auf der Anwendungsschicht selbst.

4.4.1 Sicherheit auf Transportschicht mit SSL

SOAP-Nachrichten sind aus logischer Sicht XML-Dokumente die über ein Netzwerk[26] transportiert werden müssen. Innerhalb dieses Prozesses wird ein Protokoll zur Übertragung benötigt, welches aber durch den SOAP-Standard nicht explizit vorgegeben ist. In den meisten Fällen wird für die Kommunikation das HTTP-Protokoll verwendet. Sicherheit auf Transportschicht bedeutet eine Sicherung des HTTP-Protokolls mit SSL. Weiterhin beseht die Möglichkeit alternativ HTTP Digest Authentication oder HTTP Basic Authentication einzusetzen. SSL wird über Sockets genutzt. Es existiert keine Separation zwischen der Verschlüsselung und der Transmission der Daten. Die Nutzung von SSL ist an das Transmission Control Protocol gebunden. SSL wurde ursprünglich von der Firma Netscape entwickelt [21]. Innerhalb des SSL-Protokolls wird sichere Verbindung zwischen Client und Server, im Rahmen eines Handshake-Verfahrens, aufgebaut, über die dann die Daten verschlüsselt verwendet werden. Die Funktionalität von SSL besteht dabei aus der Authentifikation der beteiligten Kommunikationspartner auf Basis von X.509 Zertifikaten. Ein Nachteil der Sicherung auf Transportschicht ist, dass nur Punkt-zu-Punkt Sicherheit existiert und nicht Ende-zu-Ende Sicherheit. Dies ist besonders dann wichtig, wenn die Kommunikation über mehrere Systeme bis zu ihrem Endpunkt übertragen wird. Abbildung 4-4 zeigt schematisch den Aufbau einer SSL Verbindung die zur beidseitigen Authentifizierung für den UAS benötigt wird. Der Client muss sich beim UAS authentifizieren, um beispielsweise administrative Aufgaben durchführen zu können. Der UAS muss sich beim Client authentifizieren, um sicherzustellen, dass vertrauenswürdige Zugangsinformationen vom richtigen Dienst zurückgeliefert werden. Die Authentifizierung wird dadurch gewährleistet, dass der Server eine entsprechende Liste von Client-Zertifikaten bereithält. Ein Client, der die Funktio-

[25] Nicht nur Client – Web Service Kommunikation muss sicher sein, sondern auch Web-Service – Web-Service Kommunikation

[26] Das Netzwerk kann ein lokales sein, wie auch ein Netzwerkverbund im Sinne des Internet

nalität des Dienstes nutzen möchte, muss über ein entsprechendes Zertifikat verfügen, mit dem er sich beim Server authentifizieren kann.

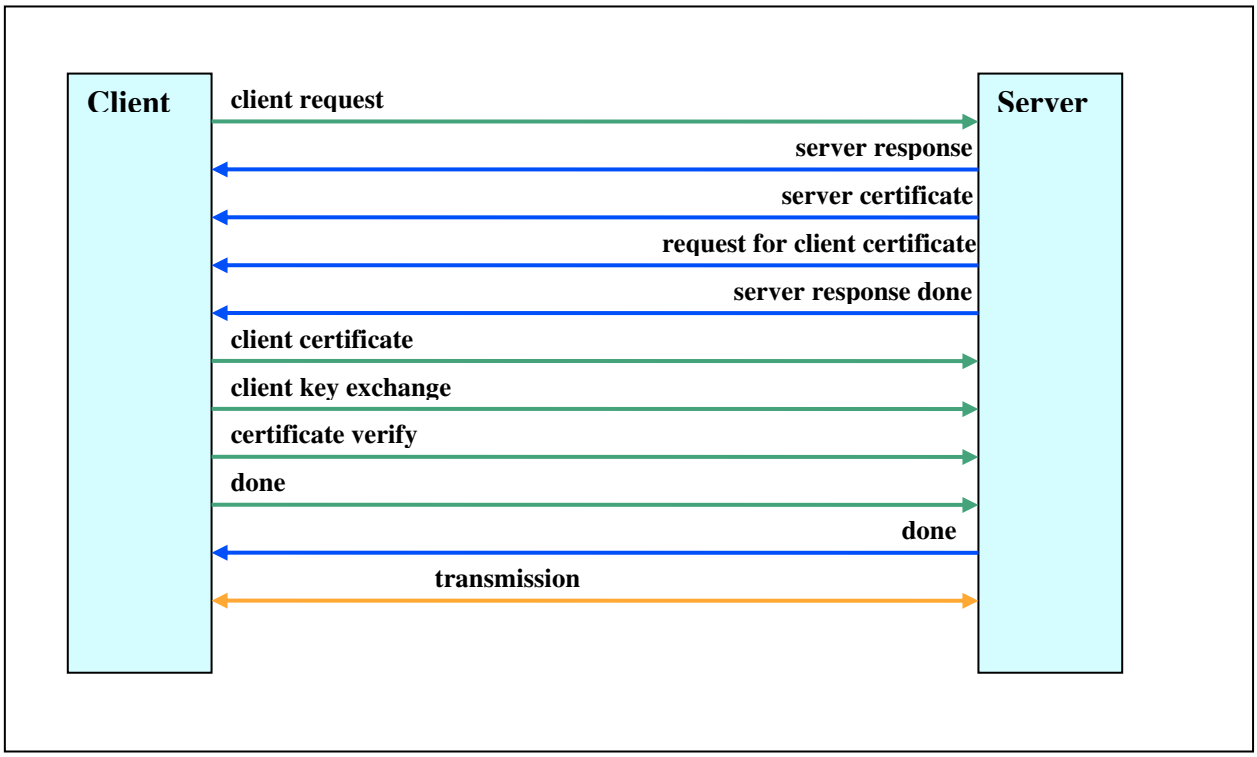


Abbildung 4-4 Aufbau der SSL-Verbindung

Im Rahmen dieser Arbeit wird eine beidseitige Authentifizierung von Client und Server über SSL realisiert. Der Dienst hat somit keinen Einfluss auf die sichere Kommunikation, sondern der Webserver innerhalb dessen der Dienst angeboten wird. Die entsprechenden Client- bzw. Server-seitigen Konfigurationen sowie die Erstellung von der Client- bzw. Serverzertifikate wird innerhalb von Kapitel 4.5 genauer beschrieben.

4.4.2 Sicherheit auf Anwendungsschicht mit WS-Security

Eine andere Form der Sicherheit kann auf der Anwendungsschicht selbst realisiert werden, wobei dieser Vorgang wesentlich komplexer ist. Innerhalb der Axis Engine wird die SOAP-Nachricht innerhalb des Message Context modifiziert. Dazu wird auf beiden Seiten ein Handler definiert, der von der SOAP-Nachricht innerhalb der dienstspezifischen Kette durchlaufen wird. Die gewünschte Sicherheit wird dadurch gewährleistet, dass der Handler ein spezielles Verschlüsselungsverfahren auf die Nachricht anwendet, bevor diese versendet werden. Die Informationen über die Art der Verschlüsselung wird dabei innerhalb der SOAP-Nachricht im SOAP-Header übertragen und die verschlüsselte Nachricht wird innerhalb des SOAP-Body übertragen. Somit ist garantiert, dass die

eigentliche Nachricht gesichert bleibt, auch wenn sie über verschiedene unsichere Transportkanäle versendet wird oder von nicht vertrauenswürdigen Zwischenstationen verarbeitet wird. Somit ist eine Ende-zu-Ende-Sicherheit gewährleistet. Allerdings entsteht sowohl auf Client wie auch auf Dienstseite zusätzlicher Aufwand, da dort die Logik zur Ver- bzw. Entschlüsselung der XML basierten SOAP-Nachrichten implementiert werden muss. Es existieren Standards zur Verschlüsselung der SOAP-Nachrichten wie z.B. XML-Encryption und WSS4J [22].
Abbildung 4-5 zeigt den schematischen Ablauf der Sicherheit mittels WS-Security.

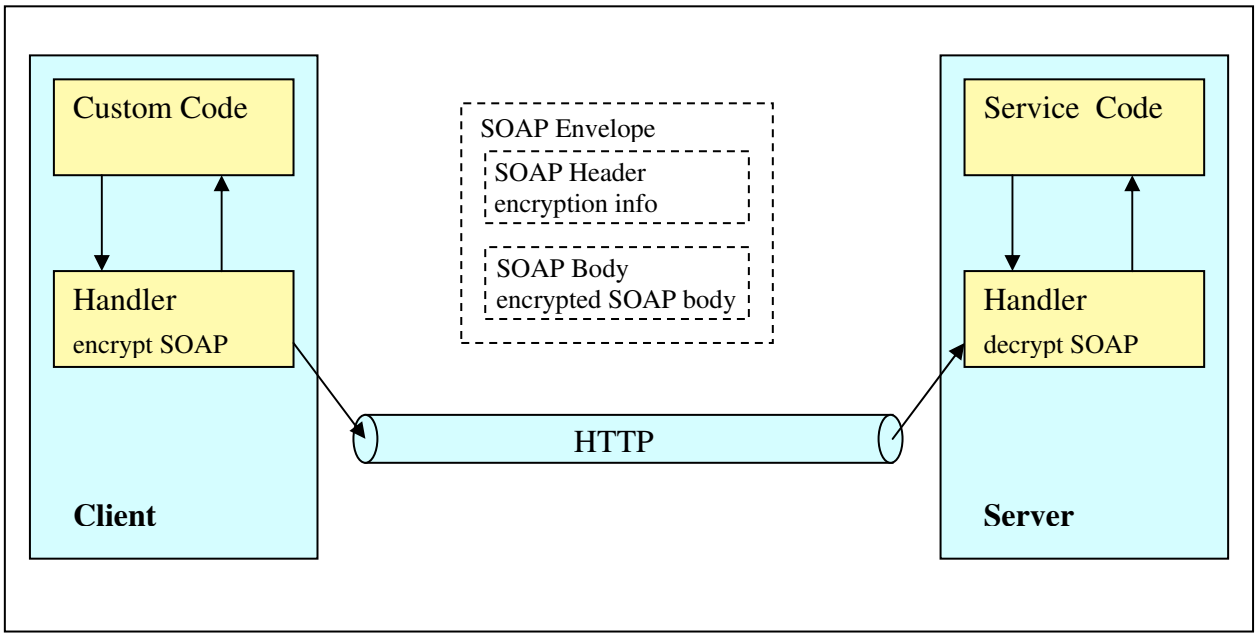


Abbildung 4-5 WS-Security Schema

Es gibt keine optimale Sicherheitsrealisierung die die beiden unterschiedlichen Verfahren gewährleisten. Beide Verfahren haben sowohl Vorteile, wie auch Nachteile die vom jeweiligen Sicherheitskontext des Dienstes abhängen. Tabelle 10 listet unterschiedliche Kriterien auf um die unterschiedlichen Sicherheitsaspekte zu differenzieren.

Sicherheitsaspekt	Sicherheit auf Transportschicht	Sicherheit auf Anwendungsschicht
Die Anwendung kommuniziert direkt mit dem Web-Service	Die Transportschicht HTTPS unterstützt den Schutz der Nachricht	Die Sicherung der Nachricht bedarf zusätzlichen Aufwand, da die Daten der SOAP-Nachricht zunächst verschlüsselt werden, bevor sie auf Dienstseite durch einen Handler wieder entschlüsselt werden.
Eine Firewall ist zwischen Anwendung und Web-Service geschaltet	Es ist nicht immer gewährleistet, dass Port 443 für die HTTPS Kommunikation innerhalb des Firewall-Systems frei gegeben ist.	Der Schutz auf Nachrichtenebene wird durch keine Firewall-Systeme beeinflusst.

Eine Web-Service-Anfrage wird durch mehrere Server geleitet	Die Sicherheit der Kommunikation ist nicht mehr gewährleistet, wenn verschiedene Zwischenstationen keine Sicherheit auf Transportebene garantieren.	Durch die Ende-zu-Ende-Sicherheit wird die Sicherheit auf Anwendungsebene nicht durch Routing auf verschiedene Server beeinflusst.
Der Web-Service benötigt eine Unterstützung für unterschiedliche Transportprotokolle (HTTP, FTP, SMTP etc.)	Verschiedene Protokolle haben auch eigene Sicherheitsmechanismen. Es ist somit schwierig eine einheitlichen Standard für die Sicherheit auf Transportebene zu garantieren.	Die Sicherheit innerhalb der SOAP-Nachricht ist unabhängig vom verwendeten Transportprotokoll.

Tabelle 10 Unterschiede der Sicherheitsmechanismen

Innerhalb dieser Arbeit wird die Sicherung des Autorisierungsdienstes über die Transportschicht gewährleistet. Die Gründe liegen darin, dass innerhalb der UNICORE-Architektur ein sehr häufiges und zeitnahes Autorisieren der Nutzer stattfindet. Es wird bereits für die Anfrage an den Web-Service und das Abfragen der Informationen über das Netzwerk eine akzeptable Zeit benötigt, die nicht unnötig verlängert werden soll durch einen zusätzlichen Zeitaufwand, der für die Verschlüsselung bzw. Entschlüsselung der SOAP-Nachrichten entsteht.

4.5 Konfiguration des UAS

Für die Funktionsfähigkeit des Autorisierungsdienstes bedarf es neben der Implementierung auch einiger Konfigurationseinstellungen, die das Deployment definieren innerhalb dessen der Dienst in die Laufzeitumgebung des Webservers integriert wird. Weiterhin bedarf es Abänderungen der NJS Konfigurationsdateien da innerhalb der UNICORE-Architektur ein anderer Weg der Autorisierung abläuft, der nicht mehr über die Standard UUDB realisiert wird. Für die Sicherung des Kommunikationskanals mit SSL bedarf es weiterhin einiger Konfigurationen die das Erstellen und Einbinden von Zertifikaten, sowie Abänderungen der Webserverkonfiguration bedürfen. Diese werden im Folgenden genauer beschrieben

.**Deployment mit WSDD**

Um den Autorisierungsdienst in die Laufzeitumgebung des Webservers zu integrieren wird folgende WSDD-Datei verwendet:

```
<deployment xmlns="http://xml.apache.org/axis/wsdd/"
xmlns:java="http://xml.apache.org/axis/wsdd/providers/java">
 <service name="UAS" provider="java:RPC">
  <parameter name="className" value="org.unigrids.fzj.uas.impl.UASEngine"/>
  <parameter name="allowedMethods" value="addSecurityToken removeSecurityToken
updateSecurityToken listUAS checkSecurityToken"/>
  <parameter name="UASFilename" value="UAS.xml"/>
  <parameter name="UASDirectory" value="/absolute/path/to/UAS"/>
 </service>
</deployment>
```

Abbildung 4-6 UAS WSDD - Datei

Mittels <deployment> wird dem Axis-Server mitgeteilt, dass ein neuer <service> eingebunden wird. Der Provider ist dafür verantwortlich für die entsprechenden SOAP-Anfragen die entsprechende Java-Klasse zu instanzieren. Der Parameter *className* referenziert die Klasse die durch den Dienst aufgerufen wird. *allowedMethods* beschreibt die Methoden, die als Web-Service-Schnittstelle nach außen zur Verfügung stehen soll. Über die Parameter *UASDirectory* und *UASFilename* kann die zu verwendende Datei für die XML-Datenhaltung spezifiziert werden.

NJS – Konfiguration

Innerhalb der `njs.properties`[27] Datei müssen folgende zusätzlichen Eintragungen gemacht werden:

```
uas.service.endpoint=https://uashost:port/path/to/service
uas.gcid=unique_identifier_1
uudb.class_name= org.unigrids.fzj.uas.service.client.UASClient
```

Abbildung 4-7 Zusätzliche Eintragungen in der NJS – Konfiguration

Über den *endpoint* Parameter wird die URL des UAS spezifiziert und kann so konfiguriert werden. Der Parameter *gcid* muss eine eindeutige Identifizierung des NJS sein, da über diesen Parameter innerhalb des UAS die Autorisierungsinformationen dem entsprechenden NJS zugeordnet werden. Über den Parameter *uudb.class_name* können andere Implementierungen der UUDB in die die Architektur von UNICORE mit einge-

[27] njs.properties definiert alle Konfigurationsparameter des NJS

bracht werden, da das Konzept der UUDB modular in das UNICORE System eingegliedert ist.

SSL - Verschlüsselung

Für die Nutzung des Dienstes über eine gesicherte Transportverbindung müssen folgende Änderungen am Webserver Tomcat vorgenommen

Zum einen müssen Zertifikate für den Client sowie für den Webserver erstellt werden um einen Aufbau der SSL-Verbindung wie in Abbildung 4-4 sicherzustellen.

```
echo Creating RSA key pair for the client ...

keytool -genkey -alias Client -keyalg RSA -dname "CN=Client, O=UAS, C=DE" \
-keypass kpss123 -keystore keystore_client_JKS -storepass spss123 -storetype \
JKS

echo Copying certificate for Client A's public key to tomcat's keystore...

keytool -export -alias Client -file foo.cer -keystore keystore_client_JKS
-storepass spss123 -storetype JKS
keytool -import -noprompt -alias Client
-file foo.cer -keystore keystore_tomcat_JKS -storepass spss123 -storetype JKS
rm foo.cer

echo Creating RS key pair for Tomcat...

keytool -genkey -alias tomcat -keyalg RSA -dname "CN=tomcat, O=UAS, C=DE" \
-keypass kpss123 -keystore keystore_tomcat_JKS -storepass spss123 -storetype \
JKS

echo Copying certificate of tomcat's public key to client's keystore ...

keytool -export -alias tomcat -file foo.cer -keystore keystore_tomcat_JKS
-storepass spss123 -storetype JKS
keytool -import -noprompt -alias tomcat -file foo.cer
-keystore truststore_client_JKS -storepass spss123 -storetype JKS
rm foo.cer
```

Abbildung 4-8 Zertifikate für Client-Server Authentifizierung

Dieses Skript erzeugt für den Client wie auch für den Tomcat Webserver einen Keystore, der jeweils, neben den eigenen, den öffentlichen Schlüssen des anderen enthält.

Weiterhin muss der Tomcat Webserver für die Nutzung von HTTPS konfiguriert werden. Folgende Änderungen müssen in der server.xml[28]- Datei eingetragen werden:

```
<!-- Define a SSL HTTP/1.1 Connector on port 8443 -->

<Connector port="8443" maxHttpHeaderSize="8192"
       maxThreads="150" minSpareThreads="25" maxSpareThreads="75"
       enableLookups="false" disableUploadTimeout="true"
       acceptCount="100" scheme="https" secure="true"
       keystoreFile="/absolute/path/to/keystore_tomcat_JKS"
       keystorePass="spss123" clientAuth="true" sslProtocol="TLS"
       truststoreFile="/absolute/path/to/keystore_tomcat_JKS"
       truststorePass="spss123"
/>
```

Abbildung 4-9 SSL-Konfiguration Tomcat

Neben den globalen Konfigurationen des Webservers kann in den entsprechenden web.xml Dateien der Anwendung explizit ein Sicherheitskontext definiert werden, innerhalb dessen die Anwendung laufen soll. Somit kann der UAS dann über SSL genutzt werden.

5 Ausblick

Abschließend soll noch ein kurzer Einblick in ein zukünftig relevantes Sicherheitsmerkmal des UAS gegeben werden, welches ein eigenes Autorisierungskonzept auf Basis von XML-Dokumenten beschreibt.

SAML

Innerhalb von Organisationen gibt es eigene Sicherheitsstandards die den Zugang zu vertraulichen Informationen an den Nachweis einer bestimmten Identität knüpfen. Es ist nicht standardisiert, wie man den Zugang und die Identitätsinformationen einheitlich beschreibt. Somit ist es schwierig eine portable Identität zu definieren, die über Organisationsgrenzen hinweg den Zugang zu Informationen regelt. Ein Berechtigungsnachweis der von einer vertrauenswürdigen Instanz ausgestellt wird soll auch für den Nachweis der Identität innerhalb einer anderen Organisation reichen.

SAML wird von OASIS entwickelt. Die genauen Spezifikationen sind den Webseiten zu entnehmen [23].

Die Aufgabe von SAML besteht nun im Austausch der Authentifizierungs- bzw. Autorisierungsinformationen zwischen verschiedenen Organisationen. SAML stellt dabei aber kein kryptographisches System bereit, das die Informationen codiert, sondern nur

[28] server.xml definiert die globalen Konfigurationsparameter des Webserver Tomcat

die Sprache mit der die Sicherheitsinformationen übertragen werden. Die Sprache muss allerdings auch über entsprechende Sicherheitsmechanismen verfügen, da über sie Passwörter bzw. andere vertrauliche Informationen übertragen werden. Innerhalb des SAML Modells existiert eine Asserting Party (AP) die Bestätigungen ausstellt und eine Relying Party (RP) die von der AP ausgestellte Bestätigungen entgegennimmt. Es soll bspw. möglich sein, Abfragen zu formulieren, die einen feinkörnigen Zugang zu bestimmten Ressourcen erlauben, ähnlich den Zugangsberechtigungen in einem Dateisystem. Dabei kommen ACL zum Einsatz. Die AP verwaltet die Zugangsinformationen, die dann im Rahmen einer Anfrage die zugehörigen Bestätigungen für die Autorisierung ausstellt.

Auch wenn bisher von SAML nur im Zusammenhang mit Web-Services die
Rede war, ist die Spezifikation in dieser Hinsicht weiter gefasst. Allerdings schreibt die Spezifikation vor, dass jede Implementierung mindestens ein SOAP Binding mitbringen muss. Somit ist die Intention und Ausrichtung von SAML trotz prinzipieller Erweiterungsmöglichkeiten eindeutig. SAML basiert auf drei XML-Elementen:

- Assertions – Ein XML Schema, welches die Sicherheitsbestätigungen definiert.
- Protocol – Ein XML Schema, welches die Anfrage- und Antwortprotokolle definiert
- Bindings – Eine Beschreibung, wie SAML Bestätigungen über Standard-Transportprotokolle genutzt werden können.

SAML-Bestätigungen sind als einfache XML-Dokumente codiert. Abbildung 5-1 zeigt ein einfaches Beispiel einer SAML-Nachricht

```
<saml:Assertion
  MajorVersion="1" MinorVersion="0" AssertionID="134.94.173.143.47112"
  Issuer="company.org"
  IssuerInstant="2006-01-24T13:00:00Z">
  <saml:Conditions
    NotBefore="2006-01-24T13:00:00Z"
    NotAfter="2006-01-24T16:00:00Z" />
  <saml:AuthenticationStatement
    AuthenticationMethod="password"
    AuthenticationInstant="2006-01-24T13:00:00Z" >
    <saml:Subject>
      <saml:NameIdentifier
        SecurityTokenDomain="company.org"
        Name="smith" />
    </saml:Subject>
  </saml:AuthenticationStatement>
</saml:Assertion>
```

Abbildung 5-1 Beispiel einer SAML - Bestätigung

Neben der reinen Authentifizierung die hier bescheinigt wird, können auch optionale Autorisierungselemente in Form von ACL mit in die Struktur einfließen und innerhalb des SOAP-Headers mit versendet werden.

6 Status

Im Rahmen dieser Arbeit ist ein Autorisierungsdienst entstanden, der folgende Funktionalitäten realisiert: Der NJS autorisiert die Nutzer nicht mehr über die ursprüngliche UUDB-Implementierung, sondern nutzt nun einen neu entwickelten Autorisierungsdienst der als entfernter Dienst über das Netzwerk angeboten wird. Abbildung 6-1 zeigt die Kommunikation zwischen NJS und UAS im TCP-Monitor. Der TCP-Monitor ist ein Teil der Axis Installation und fungiert als Proxy, der die Daten einer Anwendung entgegennimmt, sie darstellt und anschließend an ihren Bestimmungsort weiterleitet. In der Abbildung ist deutlich zu sehen, dass die Anfrage aus zwei Parametern besteht. Zum einen die *gcid*, die den NJS und damit

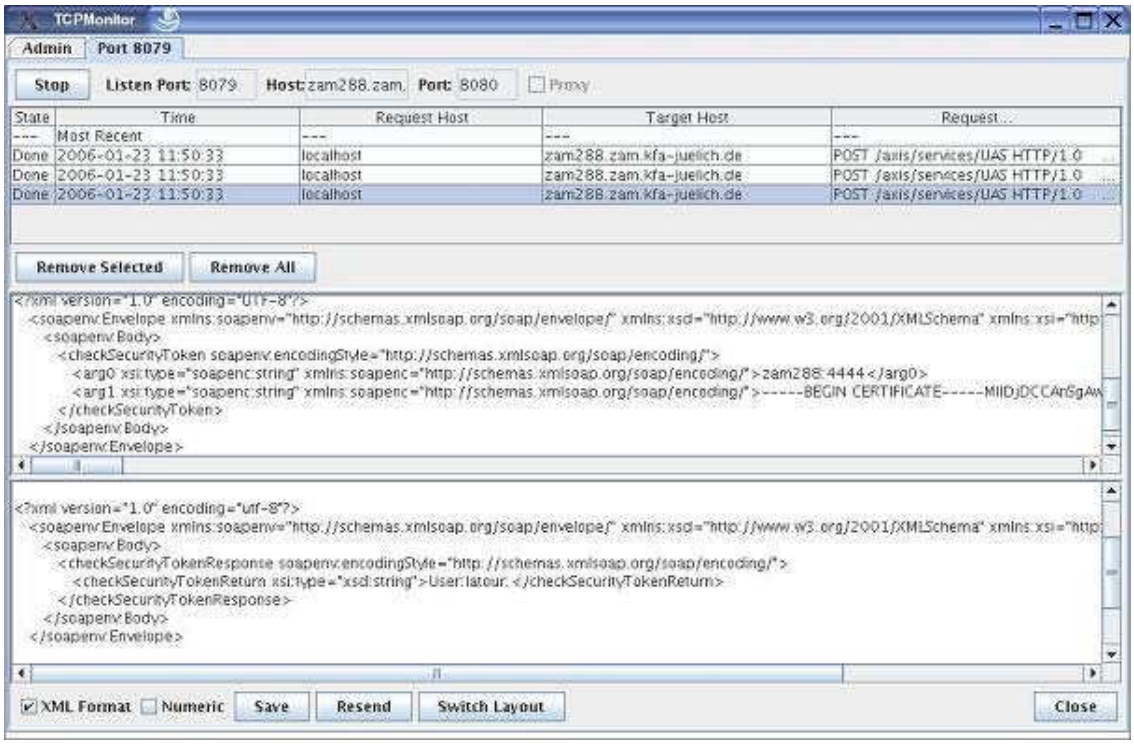


Abbildung 6-1 Kommunikation dargestellt in TCP Monitor

die Vsite eindeutig gegenüber dem UAS ausweist und als zweiten Parameter das öffentliche Zertifikat des Nutzers, welches es zu autorisieren gilt. Die Antwort auf diese Anfrage ist im unteren Teil der Abbildung innerhalb der SOAP-Nachricht zu erkennen. Sie besteht aus den Zugangsinformationen sowie einer Rolle im Sinne des UNICORE-Modells. Der Dienst stellt nun eine einzige Autorisierungsstelle im UNICORE-Grid zur

Verfügung und behebt die Einschränkung des ursprünglichen Modells, welches jeder Vsite eine eigene UUDB zugeordnet hat. Durch die entfernte Kommunikation bedarf es auch der entsprechenden Absicherung die über eine SSL-Verbindung realisiert wird. Weiterhin erfolgt die Datenhaltung des Dienstes nicht mehr in einer Programmiersprachen-spezifischen Objektserialisierung sondern in einem XML-Dokument, welches sowohl lesbarer ist, wie auch programmiertechnisch einfacher zu verarbeiten.
Die Funktionalität des Administrierens des UAS wurde von der Datenhaltung getrennt über definierte Schnittstellen beschrieben. Die Datenspeicherung ist nicht fest an eine Implementierung gebunden, sondern kann optional den Bedingungen angepasst werden und kann bei komplexeren Anforderungen an die Autorisierung auch von einem DBMS übernommen werden. Die Speicherimplementierung erfolgt über eine eigene Schnittstelle, die diese Variabilität gewährleistet. Da der Dienst die Schnittstellen für ein entferntes Administrieren implementiert, können alle Informationen über Kommandozeilenskripte dem UAS zugefügt werden oder entsprechend aus dem UAS gelöscht werden. Um einen Administrator auf einfache Art die Verwaltung der Informationen zu ermöglichen, würde eine mögliche Erweiterung der Kommandozeilenskripte, eine graphische Benutzeroberfläche vorsehen, die über die Clientklassen mit dem UAS kommuniziert und so eine einfachere Schnittstelle zur Verwaltung der Autorisierungsinformationen zur Verfügung stellt. Ebenfalls kann als zusätzliches Sicherheitsmerkmal neben der Sicherheit auf Transportschicht über das SSL Protokoll, auch eine zusätzliche Sicherheit auf Anwendungsschicht implementiert werden, die auf einer Verschlüsselung der SOAP-Nachricht basiert.[29]

[29] Voraussetzung hierfür ist weiterhin, die Gewährleistung eines schnelle und echtzeitnahen Autorisierungsablaufs

Literaturverzeichnis

1. Internetseite der Globus Software, http://www.globus.org

2. Internetseite der gLite Software, http://glite.web.cern.ch/glite

3. Internetseite der Condor Software, http://www.cs.wisc.edu/condor

4. Internetseite des UniGrids Projekts, http://www.unigrids.org

5. P. Lindner, T. Beisel, M. Resch , T. Imamura, R. Menday, P. Wieder, D. Erwin Final Report „GRIDWELTEN: User Requirements an Environments for GRID-Computing“, 2003, pp 6-7, http://webdoc.sub.gwdg.de/ebook/ah/dfn/gridwelten.pdf

6. A. Reinefeld, F. Schintke „Dienste und Standards für das Grid-Computing“, in J. von Knop, W. Haferkamp (Hrsg.), 18. DFN Arbeitstagung über Kommunikationsnetze, Düsseldorf, Lecture Notes in Informatics, Series of the German Informatics Society (GI), 2004, vol. P-55, pp. 293-304, http://www.zib.de/schintke/papers/lni04-schintke.pdf

7. H. Kishimoto, T. Maguire, The Open Grid Service Architecture Working Group, https://forge.gridforum.org/projects/ogsa-wg

8. Internetseite der UNICORE Software, http://www.unicore.org

9. Entwicklungsseiten der UNICORE Software, http://unicore.sourceforge.net

10. Internetseiten des EUROGRID Projekts, http://www.eurogrid.org

11. Internetseiten des GRIP Projekts, http://www.grip-interoperability.org

12. Spezifikation des WS-RF Standards, http://globus.org/wsrf

13. Spezifikation des XML Standards,
http://www.w3c.org/XML

14. D. Erwin (Ed.) „UNICORE Plus Final Report“ – Uniform Interface to Computing Ressources“ UNICORE Forum e.V., 2003, ISBN 3-00-011592-7

15. Spezifikation des X.509 Standards,
http://www.itu.int/ITU-T/asn1/database/itu-t/x/x509/1997/

16. Internetseite der Axis Software,
http://ws.apache.org/axis

17. Spezifikation des XML Schema Standards,
http://www.w3.org/XML/Schema

18. D. Wang, T. Bayer, T. Frotscher, M. Teufel, „Java Web Services mit Apache Axis“, Javamagazin, 2004, 3-935042-57-4

19. Internetseite der XML-Encryption Working Group,
http://wwww3.org.Encryption/2001

20. Wikipedia-Eintrag zu Man-in-the-Attacken,
http://en.wikipedia.org/wiki/Man_in_the_middle

21. Spezifikationen des SSL Protokolls
http://wp.netscape.com/eng/ssl3/ssl-toc.html

22. WSS4J Projekt
http://ws.apache.org/wss4j/

23. Spezifikationen des SAML-Standards
http://www.oasis-open.org/committees/tc_home.php?wg_abbrev=security

Anhang: Programmquellcode (CD)

Katalog der wissenschaftlichen Publikationen des ZAM (Stand: 08.03.2006)

Die mit *www* gekennzeichneten Publikationen stehen im PostScript-Format auf dem WWW-Server des Forschungszentrums unter der URL:**<http://www.fz-juelich.de/zam/files/docs/>** zur Verfügung.
Bitte richten Sie Bestellungen mit E-Mail an **literatur.zam@fz-juelich.de** oder an:

Zentralinstitut für Angewandte Mathematik
FORSCHUNGSZENTRUM JÜLICH GmbH
Informationszentrum
D-52425 Jülich

IB-2004-01		Tony C. Scott, Monique Aubert-Frécon, Gisèle Hadinger, Dirk Andrae, Johannes Grotendorst, John D. Morgan III: *Asymptotically Exact Calculation of the Exchange Energies of One-Active-Electron Diatomic Ions with the Surface Integral Method*
IB-2004-02		T. C. Scott, M. Aubert-Frécon, D. Andrae, J. Grotendorst, J. D. Morgan III, M. L. Glasser: *Exchange Energy for Two-Active-Electron Diatomic Systems within the Surface Integral Method*
IB-2004-03	*www*	Oliver Bücker: *Lösung restringierter nichtlinearer Optimierungsprobleme in Maple*
IB-2004-04	*www*	Angela Etzbach: *Performance-Analysen von seriellen und parallelen Algorithmen zur Bestimmung von Eigenwerten*
IB-2004-05		Claudia Druska: *Robuste statistische Verfahren für das Data Mining*
IB-2004-06	*www*	Felix Wolf, Bernd Mohr: *EPILOG Binary Trace-Data Format*
IB-2004-07	*www*	Carolin Schmitz: *Entwicklung einer optimalen Migrationsstrategie für ein hierarchisches Datenmanagement System*
IB-2004-08	*www*	Inge Gutheil: *Performance of single-processor BLAS on IBM p690*
IB-2004-09		John Brooke, Thomas Eickermann, Wolfgang Frings, Paul Gibbon, Lidia Kirtchakova, Ulrich Lang, Daniel Mallmann, Mark McKeown, Stephen Pickles, Andrew Porter, Mark Riding, Mathilde Romberg, Anke Visser, Uwe Wössner: *Application Steering in a Collaborative Environment*
IB-2004-10	*www*	Andrea Trensch: *Linux-Zugang mit Benutzer-Authentisierung*
IB-2004-11	*www*	Rüdiger Esser (Hrsg.): *Beiträge zum Wissenschaftlichen Rechnen - Ergebnisse des Gaststudentenprogramms 2004 des John von Neumann-Instituts für Computing*
IB-2004-12	*www*	Armin Seyfried, Marcus Strupp, Thomas Lippert: *Verfeinerte Auswertungsmethoden für Evakuierungssimulationen*
IB-2004-13		Godehard Sutmann, Vladimir Stegailov: *Optimization of Neighbor List Techniques in Liquid Matter Simulations*
IB-2004-14		Godehard Sutmann, Bernhard Steffen: *A Particle-Particle Particle-Multigrid Method for Long-Range Interactions in Molecular Simulations*
IB-2004-15	*www*	Marc-André Hermanns: *Ereignisbasierte Leistungsanalyse von Remote-Memory-Access-Operationen*
IB-2005-01		Boris Orth, Thomas Lippert, Klaus Schilling: *Finite-Size Effects in Lattice QCD with Dynamical Wilson Fermions*
IB-2005-02	*www*	Sonja Dominiczak: *Entwicklung einer Online-Visualisierung zu dem Simulationsprogramm NBODY6++ mit Hilfe der Kopplungsbibliothek VISIT*
IB-2005-03	*www*	Sandra Vogt: *Performance-Analyse paralleler Löser für dünnbesetzte lineare Gleichungssysteme auf IBM p690*
IB-2005-04	*www*	Silke Wädow: *Schnelle Wavelet basierte Berechnung von langreichweitigen Wechselwirkungen in Vielteilchensystemen*

IB-2005-05	*www*	Markus Meier: *Parameteridentifikation für Batteriegitterwachstum bei Korrosion*
IB-2005-06		Viorel Chihaia, Godehard Sutmann, Chang-Sup Lee, Soong-Hyuck Suh: *Divergence-Free Description of Molecular Rotation in Cartesian Coordinates: Derivation of the Axis-Rotation Formula and Application to Molecular Modelling*
IB-2005-07		Tatjana Eitrich, Bruno Lang: *Analysis of Support Vector Machine Training Costs for Large and Unbalanced Data from Pharmaceutical Industry*
IB-2005-08		Godehard Sutmann: *Compact Finite Difference Schemes of Sixth Order for the Helmholtz Equation*
IB-2005-09		Tony C. Scott, Monique Aubert-Frécon, Johannes Grotendorst: *New Approach for the Electronic Energies of the Hydrogen Molecular Ion*
IB-2005-10		Tony C. Scott, Robert Mann, Roberto E. Martinez II, Johannes Grotendorst: *General Relativity and Quantum Mechanics: Towards a Generalization of the Lambert W Function*
IB-2005-11		Tatjana Eitrich, Bruno Lang: *Shared Memory Parallel Support Vector Machine Learning*
IB-2005-12		Tatjana Eitrich, Bruno Lang: *Efficient Implementation of Serial and Parallel Support Vector Machine Training with a Multi-Parameter Kernel for Large-Scale Data Mining*
IB-2005-14	*www*	Brian J. N. Wylie, Bernd Mohr, Felix Wolf : *Holistic Hardware Counter Performance Analysis of Parallel Programs*
IB-2005-15	*www*	Guido Arnold, Thomas Lippert, Nikolas Pomplun, Marcus Richter: *Large Scale Simulation of Ideal Quantum Computers on SMP-Clusters*
IB-2005-16		Tatjana Eitrich, Bruno Lang: *On the Advantages of Weighted L_1-Norm Support Vector Learning for Unbalanced Binary Classification Problems*
IB-2005-17	*www*	René Puttin: *Modulare und parallele Implementierung des Jacobi-Davidson-Verfahrens*
IB-2005-18	*www*	Britta Janssen: *Eine Entwicklungsumgebung für iterative Eigenwertverfahren in MATLAB*
IB-2005-19	*www*	Otto Büchner, Ulrike Schmidt, Wolfgang Gürich, Lothar Wollschläger: *Konzept für das zentrale Datamanagement im Forschungszentrum Jülich*
IB-2006-01	*www*	Bastian Tweddell: *Ein flexibler und effizienter Dateitransfer für UNICORE*
IB-2006-02	*www*	André Latour: *Ein Web-Service-basierter Autorisierungsdienst für Grid-Umgebungen am Beispiel von UNICORE*
Jül-4154		Morris Riedel: *Prospects and Realization of Flexible Service Offers in a Grid Environment*
Jül-4157		Maik Boltes: *Adaptive Netzvereinfachung auf der Basis des Quadrik-Fehlermaßes*
-		M. Reshetnyak, B. Steffen: *The subgrid problem of thermal convection in the Earth's liquid core*
-		N. Attig (Hrsg.), K. Binder, H. Grubmüller, K. Kremer (Hrsg.): *Computational Soft Matter: from Synthetic Polymers to Proteins; NIC Winter School, 29 February - 6 March 2004, Gustav-Stresemann-Institut, Bonn, Germany - Poster Abstracts*
-		N. Attig (Hrsg.), K. Binder, H. Grubmüller, K. Kremer (Hrsg.): *Computational Soft Matter: from Synthetic Polymers to Proteins; NIC Winter School, 29 February - 6 March 2004, Gustav-Stresemann-Institut, Bonn, Germany - Lecture Notes*

- Dirk Breuer, Dietmar Erwin, Daniel Mallmann, Roger Menday, Mathilde Romberg, Volker Sander, Bernd Schuller, Philipp Wieder: *Scientific Computing with UNICORE*
- Johannes Grotendorst: *Computermathematik*
- A. Kless, T. Dickhaus, W. Meyer, J. Grotendorst: *Data Mining in der pharmazeutischen Forschung und Entwicklung*
- A. Kless, T. Eitrich: *Cytochrome P450 classification of drugs with support vector machines implementing the nearest point algorithm*
- A. Kless, T. Eitrich, W. Meyer, J. Grotendorst: *Data Mining in F & E*
- Thomas Lippert: *Recent development and perspectives of machines for lattice QCD*
- T. Lippert, B. Orth, K. Schilling: *Light Hadrons in a Box*
- T. Lippert, K. Schilling: *Quarks auf dem Gitter*
- G. Kneller, G. Sutmann: *Scaling of the Memory Function and Brownian Motion*
- E. Pomplun, G. Sutmann: *Is Coulomb explosion a damaging mechanism for 125-IUdR?*
- Stefan Birmanns, Maik Boltes, Herwig Zilken, Willy Wriggers: *Adaptive Visuo-Haptic Rendering for Hybrid Modeling of Macromolecular Assemblie*
- F. Wolf, B. Mohr, J. Dongarra, S. Moore: *Efficient Pattern Search in Large Traces through Successive Refinement.*
- S. Krieg: *Simulation der Quantenchromodynamik mit dynamischen Overlap-Fermionen*

- Maxim Reshetnyak, Bernhard Steffen: *A dynamo model in a spherical shell*
- B. Steffen, K. P. Müller, M. Komenda, R. Koppmann, A. Schaub: *A new mathematical procedure to evaluate peaks in complex chromatograms*
- T. Eitrich, B. Lang: *Efficient Optimization of Support Vector Machine Learning Parameters for Unbalanced Data Sets*
- T. Eitrich, B. Lang: *Parallel Tuning of Support Vector Machine Learning Parameters for Large and Unbalanced Data Sets*
- S. Moore, F. Wolf, J. Dongarra, S. Shende, A. Malony, B. Mohr: *A Scalable Approach to MPI Application Performance Analysis*
- Marc-André Hermanns, Felix Wolf, Bernd Mohr: *Event-based Measurement and Analysis of One-sided Communication*
- N. Bhatia, F. Song, F. Wolf, J. Dongarra, B. Mohr, S. Moore: *Automatic Experimental Analysis of Communication Patterns in Virtual Topologies*
- N. Bhatia, S. Moore, F. Wolf, J. Dongarra, B. Mohr: *A Pattern-Based Approach to Automated Application Performance Analysis*
- S. Moore, F. Wolf, B. Mohr, J. Dongarra: *Improving Time to Solution with Automated Performance Analysis*
- Achim Streit, Dietmar Erwin, Thomas Lippert, Daniel Mallmann, Roger Menday, Michael Rambadt, Morris Riedel, Mathilde Romberg, Bernd Schuller, Philipp Wieder: *UNICORE - From Project Results to Production Grids.*
- Morris Riedel, Volker Sander, Philipp Wieder, Jiulong Shan: *Web Services Agreement based Resource Negotiation in UNICORE.*
- M. Riedel, V. Sander, M. Fidler: *Self-managing Functions in Web Services Agreement-based Autonomic Grids*
- Morris Riedel: *Object Migration Pattern: Towards Stateful Web Services in Grid Environments*
- Morris Riedel, Daniel Mallmann, Achim Streit: *Enhancing Scientific Workflows with Secure Shell Functionality in UNICORE Grids*

- A.K. Vogt, G. Wrobel, W. Meyer, W. Knoll, A. Offenhäusser: *Synaptic plasticity in micropatterned neuronal networks*
- Rudolf Berrendorf, Marc-André Hermanns, Jan Seidel: *Remote Parallel I/O in Grid Environments*
- Paul Gibbon: *Short Pulse Laser Interactions with Matter: An Introduction*
- S. Pfalzner, P. Gibbon: *Many Body Tree Methods in Physics*
- Paul Gibbon: *Resistively enhanced proton acceleration in high intensity laser-foil interactions with cold foil targets*
- Paul Gibbon: *Mesh-free plasma simulation with a parallel tree code*
- Bernd Schuller, Mathilde Romberg, Lidia Kirtchakova: *Application Driven Grid developments in the OpenMolGRID project*
- G. S. Bali, T. Düssel, T. Lippert, H. Neff, Z. Prkacin, K. Schilling: *String breaking with dynamical Wilson fermions*
- D. Baric, B. Kovacevic, Z.B. Maksic, Th. Müller: *A Novel Approach in Analyzing Aromaticity by Homo- and Isostructural Reactions: An ab-initio Study of Fluorobenzenes*
- Dietmar Erwin: *NOMCOM - Selecting GGF Leadership*
- A. Streit, O. Wäldrich, Ph. Wieder, W. Ziegler: *On Scheduling in UNICORE - Extending the Web Services Agreement based Resource Management Framework*
- N. Attig, G.S. Bali, Th. Düssel, Th. Lippert, H. Neff, Z. Prkacin, K. Schilling: *Demonstration of string breaking in quantum chromodynamics by large-scale eigenvalue computations*
- Thomas Eickermann: *In a Magic Triangle: IT-Security in a Research Centre - Virus - Sicher im Netz*
- Armin Seyfried, Bernhard Steffen, Wolfram Klingsch, Maik Boltes: *The fundamental diagram of pedestrian movement revisited*
- Armin Seyfried, Bernhard Steffen, Wolfram Klingsch, Thomas Lippert, Maik Boltes: *Steps toward the fundamental diagram - Empirical results and modelling*
- Oliver Passon, Maik Boltes, Stefan Birmanns, Herwig Zilken, Willy Wriggers: *Laplace-filter enhanced haptic rendering of macromolecules*
- Holger Brunst, Bernd Mohr: *Performance Analysis of Large-scale OpenMP and Hybrid MPI/OpenMP Applications with VampirNG*
- Paul Gibbon, Wolfgang Frings, Bernd Mohr: *Performance analysis and visualization of the N-body tree code PEPC on massively parallel computers*
- Brian J. N. Wylie, Bernd Mohr: *Holistic hardware counter performance analysis of parallel programs*
- Bernd Mohr, Andrej Kühnal, Marc-André Hermanns, Felix Wolf: *Performance Analysis of One-sided Communication Mechanisms*
- Bernd Mohr, Luiz DeRose, Jeffrey Vetter: *A Performance Measurement Infrastructure for Co-Array Fortran*
- M. Fidler, V. Sander, W. Klimala: *Traffic Shaping in Aggregate-Based Networks: Implementation and Analysis*
- F. Heine, M. Hovestadt, O. Kao, A. Streit: *On the Impact of Reservations from the Grid on Planning-Based Resource Management*
- Jörg Striegnitz: *Integration von Programmiersprachen durch strukturelle Typanalyse und partielle Auswertung*
- Mathilde Romberg (Hrsg.): *OpenMolGRID - Open Computing Grid for Molecular Science and Engineering*
- S. Sild, U. Maran, M. Romberg, B. Schuller, E. Benfenati: *OpenMolGRID: Using Automated Workflows in GRID Computing Environment*

- T. Holtschneider: *Modellanalyse der Nachbarschaftslistentechnik für parallele Molekulardynamik-Simulationen*