INSTITUT FÜR INFORMATIK
DER LUDWIG–MAXIMILIANS–UNIVERSITÄT MÜNCHEN


**Fortgeschrittenenpraktikum**

# VO Membership Management mit VOMS und AJAX

Jakob Lepiarczyk

INSTITUT FÜR INFORMATIK
DER LUDWIG–MAXIMILIANS–UNIVERSITÄT MÜNCHEN


**Fortgeschrittenenpraktikum**

# VO Membership Management mit VOMS und AJAX

Jakob Lepiarczyk

| | |
|---|---|
| Aufgabensteller: | Prof. Dr. D. Kranzlmüller |
| Betreuer: | Dr. Michael Schiffers<br>Christoph Spielmann |
| Abgabetermin: | 20. August 2010 |

Hiermit versichere ich, dass ich die vorliegende Diplomarbeit selbständig verfasst und keine anderen als die angegebenen Quellen und Hilfsmittel verwendet habe.

München, den 18. August 2010

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

**Abstract**

Diese Arbeit beschreibt wie man Ajax in Kombination mit einem VOMS-Server und VOMS Admin sinnvoll nutzen kann. Zu diesem Zweck wurde im Rahmen dieser Arbeit eine Ajax-Anwendung namens VOMS-AJAX-GUI entwickelt, die die Vorteile von zwei Seiten beleuchtet. Einerseits wird aufgezeigt, wie man durch eine neue kompakte Benutzeroberfläche und den Einsatz von Ajax-Request eine Möglichkeit schaffen kann, VOs durch eine bessere Übersichtlichkeit und Usability schneller und bequemer zu verwalten. Andererseits wird gezeigt, wie man mit Hilfe von Ajax SAML Queries nach Wunsch zusammenstellen und anschließend mit diesen SAML Assertions abfragen kann, um diese z.B. für die Anmeldung bei Grid-Diensten zu nutzen.
In der Arbeit werden sowohl das Funktionsprinzip und die Vorteile der neu entwickelten Anwendung gegenüber dem bestehenden System dargestellt, als auch wird auf aufgetretene Probleme mit der verwendeten Grid-Middleware gLite eingegangen und es werden einige Empfehlungen für die produktive Umsetzung der Konzepte abgegeben.

# Inhaltsverzeichnis


**1 Einführung** **1**

**2 Grundlagen** **3**
2.1 Virtuelle Organisation . . . . . 3
2.2 Virtual Organization Membership Service . . . . . 3
2.3 VOMS Admin . . . . . 4
2.4 Security Assertion Markup Language . . . . . 4
2.5 Ajax . . . . . 5

**3 Vorbereitungen** **7**
3.1 Installation von Scietific Linux . . . . . 7
3.2 Installation einer gLite-VOMS-Node . . . . . 8
3.3 Einrichtung von VOMS und VOMS Admin . . . . . 8
3.4 Einrichtung von VOMS-AJAX-GUI . . . . . 9
3.5 Hinweise zur Einrichtung einer eigenen CA für Testzwecke . . . . . 10

**4 Genutzte Bibliotheken und Frameworks** **11**
4.1 Clientseitig: JavaScript-Frameworks . . . . . 11
4.2 Serverseitig: Java-Bibliotheken . . . . . 11

**5 Verbessertes VO-Management am Beispiel Gruppen-, Rollen- und Attributverwaltung** **13**
5.1 Beschreibung der GUI . . . . . 13
5.2 Konzeptionelle Unterschiede zwischen VOMS Admin und VOMS-AJAX-GUI 15
5.3 Funktionsprinzip . . . . . 16
5.4 Beschreibung der einzelnen Funktionen . . . . . 18

**6 Neue Möglichkeiten durch den Einsatz von Ajax am Beispiel des Abrufs von SAML Assertions** **23**
6.1 Beschreibung der GUI . . . . . 23
6.2 Funktionsprinzip . . . . . 24
6.3 Einsatzmöglichkeiten . . . . . 27
6.4 Beschreibung der einzelnen Funktionen . . . . . 29
6.5 Probleme mit der aktuellen VOMSSaml-Schnittstelle . . . . . 30

**7 Vorschläge für eine produktive Implementierung** **33**
7.1 Alternative JavaScript-Frameworks . . . . . 33
7.2 Verbessertes Tabellenlayout mit CSS3 . . . . . 34

**8 Fazit** **37**

**9 Anhang** **39**
9.1 Virtuelle Maschine . . . . . 39
9.2 Troubleshooting . . . . . 39

**Abbildungsverzeichnis** **41**

**Literaturverzeichnis** **43**

# 1 Einführung

Im Laufe des Web 2.0-Hypes hat sich Ajax zu einer der Basistechnologien des Webs entwickelt. Es stellt sich die Frage, ob sich diese Technologie sinnvoll im Umfeld von Grid-Computing und hier im speziellen zusammen mit einem VOMS-Server nutzen lässt. In dieser Arbeit wird dieser Frage aus zwei Richtungen nachgegangen.
Einerseits soll VOMS aus Administratorsicht betrachtet werden. Da Ajax dafür bekannt ist dynamische, sich wie Desktop-Anwendungen anfühlende, Web-Oberflächen zu ermöglichen, stellt sich die Frage, ob man auf diese Weise nicht auch die Administration eines VOMS-Servers vereinfachen könnte. In diesem Teil der Arbeit wird versucht durch den verstärkten Einsatz von Ajax und eine neu entwickele Benutzeroberfläche eine Alternative zu VOMS Admin zu schaffen, die eine bessere Usability bietet. Es soll dem Administrator ein besserer Überblick über die VO geboten werden und erforderliche Veränderungen sollen sich in kürzerer Zeit durchführen lassen.
Andererseits soll VOMS aus Endanwendersicht betrachtet werden. Im Bereich des Grid-Computing werden von autonomen Resource-Providern Dienste zur Verfügung gestellt, die von virtuellen Organisationen unter bestimmten Voraussetzungen genutzt werden können [FKT01]. Für den Zugriff auf diese Dienste ist eine Autorisierung erforderlich, die bisher häufig mit Hilfe von Proxy-Zertifikaten realisiert wird. Diese können unter den verschieden Grid-Middlewares mit Hilfe eines VOMS-Servers erstellt werden. Neuere Versionen von VOMS und VOMS Admin stellen aber mittlerweile auch die Möglichkeit zur Verfügung an Stelle von Proxy-Zertifikaten SAML-Assertions für die Autorisierung zu erzeugen. Da einerseits bei Ajax XML-Daten verarbeitet werden und andererseits es sich bei SAML-Assertions um XML-Daten handelt, bietet es sich an dies zu kombinieren. Es soll aufgezeigt werden, wie man mit Hilfe von Ajax eine bequeme Möglichkeit schaffen kann SAML-Assertions abzurufen, um sich mit diesen dann z.B. bei einem Grid-Dienst zu autorisieren.

Begonnen wurde diese Arbeit ursprünglich mit der Zielsetzung Ajax für den Abruf von SAML Assertions zu nutzen. Im Laufe der Untersuchung der bestehenden Systeme hat sich dann aber die Erkenntnis herausgebildet, dass für einen solchen Anwendungsfall ein direkter Zugriff auf die VOMS-Datenbank von großem Vorteil wäre. Da für diesen Zweck ein Backend implementiert werden musste, dass auf die Datenbank zugreift, wurde dieses sogleich dahingehend erweitert, dass es nicht nur den Abruf von SAML Assertions unterstützt, sondern auch die Verwaltung der VO und somit der Attribute die in der Assertion enthalten sind. VOMS Admin wurde auf seine Schwachstellen untersucht und es wurde der Versuch unternommen eine GUI zu entwickeln die besser ist, als die des VOMS Admin. Da Ajax bei allen Überlegungen die zentrale Rolle spielte, wurde eine Architektur entworfen, die besonders starken gebrauch von Ajax-Request macht, wenn auch andere Lösungen denkbar wären.

# 2 Grundlagen

Zunächst sollen einige grundlegende Begriffe geklärt werden, die im Folgenden von zentraler Bedeutung sind.

## 2.1 Virtuelle Organisation

Zu den Hauptmerkmalen eines Grids gehört, dass diese multiorganisational sind und ein verteiltes Management haben. Eine zentrale Organisationseinheit in Grids bilden virtuelle Organisation oder kurz VOs. Dabei handelt es sich um eine sich dynamisch ändernde Vereinigung von unabhängigen Institutionen oder Personen, die Mitglieder realer Organisationen sein können. Eine VO wird gegründet um gemeinsam an einem oder mehreren Zielen zu arbeiten. Häufig teilen sich die Mitglieder der VO zu diesem Zwecke gemeinsame Ressourcen. Die Organisation ist virtuell, da sie keinen physischen Standort hat, sondern durch die virtuelle, meist über das Internet stattfindende, Zusammenkunft der Individuen entsteht. Die Mitglieder einer VO bilden in der Regel eine Hierarchie aus Gruppen, in welchen wiederum bestimmte Rollen besetzt werden können. Die Mitgliedschaft in den Gruppen und die Einnahme bestimmter Rollen berechtigt die Mitglieder zum Zugriff auf bestimmte Ressourcen die von Resource-Providern, welche wiederum VOs sein können, bereit gestellt werden.
Mehr zum Thema virtuelle Organisationen, wie auch zum Aufbau von Grids im Allgemeinen, kann man unter [FKT01] nachlesen.

## 2.2 Virtual Organization Membership Service

Beim Virtual Organization Membership Service (oder kurz VOMS) handelt es sich um ein System zum Management von Autorisierungen innerhalb von multiinstutionalen Kollaborationen, also VOs. Entwicklt wurde VOMS ursprünglich vom European DataGrid Project und war zunächst Bestandteil der Grid-Middleware Glopus Toolkit. Zu einem späteren Zeitpunkt wurde VOMS auch in die Middlewares UNICORE und gLite integriert. Unter Letzterer wird es zur Zeit vor allem weiterentwickelt.
Den Kern von VOMS bildet eine MySQL- oder Oracle-Datenbank, welche Informationen zu den einzelnen Mitgliedern der VO enthält. So gibt es zu jedem Benutzer einen Datensatz mit Informationen, wie Adressen, Telefonnummern, Zertifikaten und ähnlichem. Gegliedert wird die VO durch eine hierarchische Struktur aus Gruppen und Rollen. Jedes Mitglied kann in einer beliebigen Anzahl an Gruppen Mitglied sein und in diesen wiederum bestimmte Rollen wahrnehmen. Des Weiteren können jedem Mitglied beliebige zusätzliche Attribute mit beliebigen Werten zugewiesen werden - die so genannten Generic Attributes. VOMS stellt einige Kommandozeilen-Werkzeuge bereit, um die Datenbank entsprechend zu bearbeiten.
Ein typisches Beispiel wäre die Aufteilung der VO in eine Gruppe der Administratoren, Entwickler und Beta-Tester. In jeder Gruppe wiederum kann es die Rolle eines Stellvertreters geben. Der VO könnte bei einem Grid-Dienst eine bestimmte Menge einer Ressource zur

Verfügung stehen, dessen Anteile die VO mit Hilfe der generischen Attribute an die Mitglieder verteilt.
Die wichtigste Aufgabe von VOMS ist die Ausstellung von Proxy-Zertifikaten. Ein solches Zertifikat kann sich ein Mitglied einer VO mit Hilfe des Tools *voms-proxy-init* erzeugen lassen. Dieses ermöglicht nicht nur eine Autorisierung des Mitglieds als solches, es enthält auch zusätzliche Informationen aus der VOMS-Datenbank, z.B. Gruppenzugehörigkeiten oder Rollen. Benötigt werden solche Zertifikate für die Autorisierung von Grid-Jobs und somit haben diese in der Regel - im Gegensatz zum normalem persönlichem Zertifikat des Mitglieds - eine Gültigkeit von wenigen Stunden.

## 2.3 VOMS Admin

Bei VOMS Admin handelt es sich um eine Erweiterung des VOMS-Servers. Unter gLite ist VOMS Admin bereits ein integraler Bestandteil einer VOMS-Node. VOMS Admin ist in erster Linie für VO-Administratoren gedacht. So wird eine Web-Oberfläche mit einem sehr fein einstellbarem Rechtemanagement angeboten, welche eine bequeme Möglichkeit zur Verwaltung der VOMS-Datenbank bietet. Neben dem Web-Interface bietet VOMS Admin dem VO-Administrator auch einige SOAP-Schnittstellen, um an Stelle der Web-Oberfläche alternative Anwendungen nutzen zu können.
Das umfangreiche Web-Interface für Administratoren wird um kleinere Bestandteile für Endanwender ergänzt, wie etwa eine Registrierungsmöglichkeit für neue Mitglieder, die für kleine VOs durchaus ausreichend ist. Größere und komplexere VOs werden aber in der Hinsicht auf das leistungsfähigere VOMRS (VO Membership Registration Service) zurückgreifen.
Seit kurzem bietet VOMS Admin auch eine SOAP-Schnittstelle um SAML-Assertions abrufen zu können, womit eine Alternative zu Proxy-Zertifikaten besteht, was im weiteren für die Arbeit von besonderer Bedeutung ist.
Weitere Informationen bezüglich des Einsatzes von VOMS Admin findet man unter [VA08].

## 2.4 Security Assertion Markup Language

Bei der Security Assertion Markup Language (oder kurz SAML) handelt es sich um einen vom OASIS-Konsortium verabschiedeten Standard, der dem Austausch von Autorisierungs- und Authentifizierungsinformationen dient. SAML baut hierbei auf andere Standards auf, hierzu gehören XML Signature und XML Encryption. Gliedern läßt sich SAML in drei Komponenten: Profiles, Bindings und Assertions. Ein Subjekt fragt dabei mit einer SAML Query bei einem Identity-Provider eine SAML Assertions ab, die dann in der Regel zu einem Service-Provider übertragen wird. Im Grid Computing-Umfeld dieser Arbeit ist der Identity-Provider der VOMS-Server, der Service-Provider ein Grid-Dienst. Bei SAML wird zwischen Authentication, Attribute und Authorization Decision Statements unterschieden. In dieser Arbeit werden vor allem die Attribute Statements betrachtet, durch welche der Identity-Provider bestimmte Eigenschaften des Subjekts in Form von Attribut-Wert-Paaren zusichern kann. SAML 2.0 unterscheidet zwischen verschiedenen Bindings, welche festlegen wie die SAML-Kommunikation stattfindet. Für diese Arbeit relevant ist das SAML SOAP Binding, bei welchem Queries und Responses über SOAP 1.1-Nachrichten übertragen werden; weitere Bindings wären z.B. das HTTP POST Binding oder das SAML URI Binding.
Die aktuelle und auch in dieser Arbeit verwendete Version ist 2.0. Für diese Arbeit ausrei-

chend ist die zentrale Core-Spezifikation, welche unter [SAM05] nachgelesen werden kann. Einen guten Einstieg in SAML bieten zudem [KK07a] und [KK07b].

## 2.5 Ajax

Ajax ist ein Apronym und steht für „Asynchronous JavaScript and XML“. Die ursprüngliche Idee dahinter ist, dass eigentlich schon lange Zeit existierende Technologien neu verknüpft wurden, um so dynamische Web-Anwendungen zu erschaffen, die sich wie lokal ausgeführte Desktop-Programme anfühlen. Im Zuge der Web 2.0-„Revolution“ wurde Ajax zum neuen Heilsbringer hochgehypt und mit der Zeit mit vielen weiteren Technologien assoziiert. Meistens wird z.B. heutzutage kaum mehr XML verwendet, wie die ursprüngliche Wortbedeutung nahe legt, sondern JSON. In dieser Arbeit wird aber entsprechend der ursprünglichen Bedeutung XML verwendet.
Zentrales Element bei Ajax-Anwendungen ist das XMLHttpRequest oder kurz XHR. Es handelt sich dabei um eine API mit der man im Hintergrund asynchron (oder aber auch synchron) HTTP-Methoden ausführen kann. Es können also z.B. Formulareingaben im Hintergrund ohne die Seite neu zu laden, über einen HTTP-Post-Request zum Server gesendet werden oder z.B. HTML-Fragmente vom Server abgerufen und anschließend direkt in die Seite eingebaut werden. Allerdings bieten ältere Browser, z.B. Internet Explorer 5 bis 6, die API nicht nativ an, sondern stellen diese nur über ein ActiveX-Objekt bereit. Damit eine Anwendung in allen Browsern funktioniert, muss normalerweise mit mehreren Try-Catch-Blöcken gearbeitet werden. Ein Vorteil von JavaScript-Frameworks ist, dass diese die Differenzen weg abstrahieren und einen einheitlichen Weg bieten Ajax-Request durchzuführen.
Die folgende Grafik verdeutlicht das Prinzip eines Ajax-Aufrufs, wobei die angegebenen Datentypen und Komponenten bereits denen, der für diese Arbeit entwickelten VOMS-AJAX-GUI entsprechen.
Literatur zum Thema Ajax gibt es sehr viel, für den Einstieg sei auf [ML07] und [Jäg08] verwiesen.

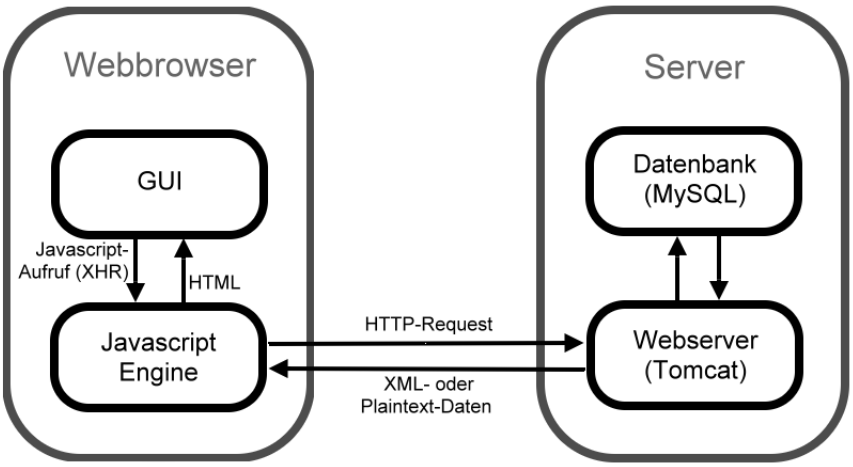



Abbildung 2.1: Ajax-Aufruf

# 3 Vorbereitungen

Benötigt wird für diese Arbeit ein VOMS-Server mit VOMS Admin in einer aktuellen Version, die die SAML-Schnittstelle bereitstellt. Da VOMS vor allem unter gLite weiterentwickelt wird, wird bei dieser Arbeit ein VOMS-Server unter eben dieser Grid-Middleware aufgesetzt. gLite stellt für die verschiedenen benötigten Knoten eines Grids spezifische Installationspakete bereit. Will man einen VOMS-Server aufsetzen, so muss man eine gLite-VOMS-Node einrichten. Im Folgenden wird beschrieben wie man dies bewerkstelligen kann und wie man anschließend den VOMS-Server, VOMS Admin und die für diese Arbeit entwickelte VOMS-AJAX-GUI einrichtet.
Da es sich bei diesen Systemen um Software handelt, die sich in stetiger Weiterentwicklung befindet, kommt es bei einigen Versionen durchaus zu größeren Fehlern. Auch die Dokumentation der Software stimmt häufig nicht mir der Realität überein. Da zudem die SAML-Schnittstelle erst seit kurzem verfügbar ist, stand eine größere Auswahl an möglicherweise stabileren Versionen ohnehin nicht zur Verfügung. Somit wurde für diese Arbeit immer jeweils mit den neusten Versionen der erforderlichen Software gearbeitet. Die Software wird zum Teil nicht entsprechend der offiziellen Dokumentation installiert, da diese zum Teil nicht funktionsfähig ist oder unnötige Aspekte berücksichtigt. Da hier ein Testsystem eingerichtet werden soll, wird darauf Wert gesetzt möglichst schnell ein lauffähiges System zu erhalten, Sicherheitsaspekte werden dabei außen vor gelassen.
Es ist anzumerken, dass sich in einer späteren Version von gLite das Vorgehen deutlich unterscheiden kann, da gerade im Bereich der Installation einer VOMS-Node größere Änderungen zu erwarten sind. Die Anleitung hier soll für spätere Versionen also eher als Orientierungshilfe dienen.
Die Kenntnis über das Funktionsprinzip von Zertifikaten und der Umgang mit OpenSSL wird vorausgesetzt. Zum Schluss dieses Kapitels wird lediglich auf gLite-spezifische Eigenschaften beim Einrichten einer eigenen Certificate Authority für Testzwecke eingegangen.

## 3.1 Installation von Scietific Linux

Die Installation einer gLite-VOMS-Node setzt Scientific Linux (SL) oder Scientific Linux Cern (SLC) als Betriebssystem voraus. Für diese Arbeit wurde Scientific Linux 5.5 verwendet. Zunächst müssen von www.scientificlinux.org zwei DVD-Images oder mehrere CD-Images der aktuellen Version heruntergeladen werden. Wichtig ist darauf zu achten, dass die 64-Bit-Version heruntergeladen und installiert wird, da die VOMS-Node für 32 Bit nicht zur Verfügung steht. Von den DVDs oder CDs kann das Betriebssystem dann wie gewohnt dialoggeführt installiert werden, so dass hierzu keine weiteren Ausführungen erforderlich sind.

## 3.2 Installation einer gLite-VOMS-Node

Als nächstes muss die VOMS-Node eingerichtet werden. Bis zur Version 3.1 von gLite standen hiervon zwei Versionen bereit, die sich in der verwendeten Datenbank unterschieden. Zur Auswahl standen MySQL und Oracle. In gLite 3.2, welches hier verwendet werden soll, steht nur die MySQL-Variante zur Verfügung.

Alle folgenden Aktionen sollen als root ausgeführt werden. Eventuelle Nachfragen sollen mit „Ja“ beantwortet werden.
Zunächst müssen die Yum-Repositories für eine VOMS-Node angepasst werden. Neben dem eigentlichen VOMS-Node-Repository, werden das DAG-Repository und ein Repository mit aktuellen Zertifikaten benötigt. Für die Einrichtung wechselt man ins Verzeichnis */etc/yum.repos.d/* und lädt mit wget die benötigten Konfigurationen herunter:

```
wget -N http://grid-deployment.web.cern.ch/grid-deployment/\
   glite/repos/3.2/dag.repo
wget -N http://grid-deployment.web.cern.ch/grid-deployment/\
   glite/repos/3.2/glite-VOMS_mysql.repo
wget -N http://grid-deployment.web.cern.ch/grid-deployment/\
   glite/repos/3.2/lcg-CA.repo
```

Anschließend aktualisiert man das System und installiert alle benötigten Komponenten:

```
yum update
yum install glite-VOMS_mysql lcg-CA xml-commons-apis
```

## 3.3 Einrichtung von VOMS und VOMS Admin

Normalerweise steht für die Konfiguration der Grid-Dienste unter gLite das Tool YAIM (YAIM Ain't an Installation Manager) zur Verfügung. Unter gLite 3.2 wurde die Unterstützung für VOMS neu eingeführt, ist aber entgegen der Dokumentation nicht lauffähig. Da YAIM zudem darauf ausgelegt ist, die Node in die Grid-Umgebung einzufügen, was für das beabsichtigte Testsystem nicht notwendig ist, wird hier der manuelle Einrichtungsweg über die VOMS-Initialisierungs- und Konfigurationsskripte beschrieben.

Jede gLite-Node benötigt ein Host-Zertifikat. Dieses und der private Schlüssel müssen unter */etc/grid-security/hostcert.pem* und */etc/grid-security/hostkey.pem* abgelegt werden. Diese müssen dem Nutzer tomcat gehören, der Schlüssel darf natürlich nur für diesen lesbar sein. Anschließend müssen einige Umgebungsvariablen gesetzt werden:

```
export GLITE_LOCATION=/opt/glite/
export GLITE_LOCATION_VAR=/var/glite/
export GLITE_LOCATION_LOG=/var/log/
export LD_LIBRARY_PATH="/opt/globus/lib/;/opt/glite/lib64/"
export CATALINA_HOME=/usr/share/tomcat5/
```

Es ist zu empfehlen diese Zeilen der Datei */root/.bash_profile* hinzuzufügen, damit diese Variablen bei einem Neustart automatisch zur Verfügung stehen.
Im MySQL-Server muss ein root-Passwort gesetzt werden.

```
/etc/init.d/mysqld start
/usr/bin/mysqladmin -u root password <Passwort>
```

Damit der Tomcat-Server OpenSAML-konform arbeitet, müssen einige JAR-Dateien in sein Verzeichnis kopiert werden:

```
cp /opt/glite/share/voms-admin/endorsed/* /usr/share/tomcat5/common/endorsed/
```

Der Trustmanager von Tomcat kann mit dem folgenden Kommando automatisch eingerichtet werden:

```
/opt/glite/etc/glite-security-trustmanager/configure.sh
```

Es ist zu empfehlen die Prüfung von CRLs zu deaktivieren, da dies insbesondere bei Nutzung von einer Test-CA zu nicht akzeptierten Zertifikaten führt. Hierzu ändert man in der Datei */usr/share/tomcat5/conf/server.xml* die Variablen `crlEnabled` und `crlRequired` auf `false`.
Durch das Skript *voms-admin-configure* wird die Datenbank eingerichtet und der VOMS-Server passend konfiguriert. Hierbei wird auch der neue MySQL-Nutzer voms eingerichtet für den ein beliebiges Passwort zu wählen ist.

```
/opt/glite/sbin/voms-admin-configure install
   --vo <VO-Name> --admincert <Zertifikat des Administrators>
   --port 15000 --smtp-host localhost --mail-from root@localhost
   --sqlloc /opt/glite/lib64/libvomsmysql.so --dbtype mysql
   --dbusername voms --dbpassword <voms-Passwort> --deploy-database
   --createdb --dbauser root --dbapwd <root-MySQL-Passwort>
```

Die SAML-Schnittstelle ist standardmäßig deaktiviert, um dies zu ändern müssen in der Datei */var/glite/etc/voms-admin/<VO-Name>/voms.service.properties* die Kommentarzeichen in der folgenden Zeile entfernt werden:

```
## voms.aa.activate_saml_endpoint = true
```

Anschließend können alle Server gestartet werden:

```
/opt/glite/etc/init.d/voms start
/opt/glite/etc/init.d/voms-admin start
/etc/init.d/tomcat5 start
```

VOMS Admin sollte jetzt unter der URL *https://<IP-des-Servers>:8443/voms/<VO-Name>* erreichbar sein. Für die SSL-Session sollte das „Zertifikat des Administrators“ verwendet werden, welches beim Ausführen des Befehls *voms-admin-configure* angegeben wurde.
Nach einem Neustart des Systems müssen auch der MySQL-, VOMS- und Tomcatserver neu gestartet werden.

## 3.4 Einrichtung von VOMS-AJAX-GUI

Um VOMS-AJAX-GUI einzurichten, muss zunächst der Ordner *voms-ajax-gui* in den Ordner */usr/share/tomcat5/webapps/* kopiert werden. Anschließend ist die Datei *WEB-INF/web.xml* zu bearbeiten. Hier müssen VO-Name und MySQL-Zugangsdaten eingetragen werden. Anschließend kann auf die GUI über *https://<IP-des-Servers>:8443/voms-ajax-gui/* zugegriffen werden.

## 3.5 Hinweise zur Einrichtung einer eigenen CA für Testzwecke

Für Testzwecke kann es sinnvoll sein eine eigene Certificate Authority aufzusetzen, unter der man nach belieben neue Zertifikate für nicht existierende Personen erstellen kann. Damit diese Zertifikate vom VOMS- oder Tomcat-Server beim Verbindungsaufbau akzeptiert werden, muss die CA zu den bekannten vertrauenswürdigen CAs hinzugefügt werden. Unter gLite findet man die CA-Zertifikate zusammen mit einigen weiteren Konfigurationsdateien im Verzeichnis */etc/grid-security/certificates/*. Um ein neues Zertifikat hinzuzufügen muss Folgendes befolgt werden.
Zunächst sollte sichergestellt werden, dass das CA-Zertifikat im PEM-Format vorliegt. Von diesem Zertifikat muss nun der Subject-Hash ermittelt werden, was mit folgendem Kommando möglich ist:

```
openssl x509 -noout -subject_hash -in <Zertifikat>
```

Der erhaltene Hash muss mit einer 0 als Suffix als Dateiname für das Zertifikat verwendet werden und dieses im genannten Ordner abgelegt werden. Die Zugriffsrechte für die Datei müssen die gleichen sein, wie für die anderen Zertifikate. Zu den anderen CA-Zertifikaten gibt es noch weitere Dateien, die z.B. weitere Einschränkungen für die erlaubten Nutzer-Zertifikate festlegen oder eine URL zum Abruf von CRL-Listen angeben können. Für ein Testsystem ist es aber am einfachsten auf diese Dateien gänzlich zu verzichten.

# 4 Genutzte Bibliotheken und Frameworks

Im Folgenden wird kurz beschrieben, welche Bibliotheken und Frameworks für die entwickelte VOMS-AJAX-GUI genutzt werden.

## 4.1 Clientseitig: JavaScript-Frameworks

Als JavaScript-Framework wird in dieser Arbeit Prototype (www.prototypejs.org) in Version 1.7_rc2 verwendet. Hierbei handelt es sich um ein 2005 von Sam Stephenson entwickeltes und vorgestelltes Framework, welches einerseits diverse Shortcuts, z.B. bei der Selektion von DOM-Elementen bietet, andererseits diese Elemente um zusätzliche Methoden erweitert (so genannte DOM Extensions). So kann man auf ein HTML-Element mit einer bestimmten ID statt über `document.getElementById(<ID>)` mit `$(<ID>)` zugreifen. Das zurückgelieferte Element wird sogleich um die neuen Methoden erweitert. Z.B. kann mit update() der Inhalt des Elements ersetzt werden oder es kann mit setStyle() um neue CSS-Eigenschaften erweitert werden. In der VOMS-AJAX-GUI wird zudem in großem Umfang das Event-Management und die Template-Klasse genutzt - letztere bietet Text-Vorlagen und wird für die Erzeugung von Tabellen und SAML Queries genutzt.

Zusätzlich wird Scriptaculous (script.aculo.us) verwendet. Hierbei handelt es sich um ein Addon für das Prototype-Framework, welches unter anderem diverse optische Effekte und Animationen sowie Drag&Drop bietet. In der VOMS-AJAX-GUI kommt es vor allem wegen den Ajax-GUI-Bausteinen zum Einsatz. So wird die eingesetzte Autovervollständigung und der InPlace-Editor von Scriptaculous zur Verfügung gestellt. Scriptaculous wird in Version 1.8.3 eingesetzt.

## 4.2 Serverseitig: Java-Bibliotheken

Um serverseitig die benötigten XML-Dokumente zu erzeugen, welche als Antworten auf diverse Ajax-Anfragen gesendet werden müssen, wird die Bibliothek JDOM (www.jdom.org) eingesetzt.
Des Weiteren kommt Jaxen (jaxen.codehaus.org) zum Einsatz. Hierbei handelt es sich um eine XPath-Bibliothek. Die hier vorgestellte Version der VOMS-AJAX-GUI versendet zwar keine XML-Dokumente vom Client zum Server, es werden aber XPath-Ausdrücke bei der Erzeugung der XML-Antworten benötigt. Es wird über diese Ausdrücke geprüft, ob bereits bestimmte Informationen im XML-Dokument schon vorhanden sind oder erst hinzugefügt werden müssen.
Schließlich kommt auch der MySQL Connector/J (www.mysql.com/downloads/connector/j/) als JDBC-Treiber für den Zugriff auf die MySQL-Datenbak zum Einsatz.

# 5 Verbessertes VO-Management am Beispiel Gruppen-, Rollen- und Attributverwaltung

Im Folgenden wird am Beispiel von VOMS-AJAX-GUI aufgezeigt, wie man mit Hilfe von Ajax und einer neu konzeptionierten Benutzeroberfläche die Administration einer VO verbessern kann.
Zunächst aber ein paar allgemeine Worte zur VOMS-AJAX-GUI. Es handelt sich hierbei nicht um einen vollwertigen Ersatz für VOMS Admin. Es ist eine prototypische Implementierung, die einzelne Funktionen von VOMS Admin auf eine neue Weise umsetzt. Konkret wurde das Gruppen-, Rollen- und Attributmanagement gewählt, da in diesem Bereich die Möglichkeit bestand die Vorteile von Ajax besonders deutlich zu machen. Würde man z.B. das Anlegen eines neuen Benutzers betrachten, könnte man dies nicht viel anders gestalten, als es bereits in VOMS Admin gelöst ist. Folglich ist VOMS-AJAX-GUI nicht eigenständig lauffähig, sondern ist z.B. für das Anlegen und Löschen von Benutzern von VOMS Admin abhängig. Die VOMS-AJAX-GUI ist auch nicht in allen Browsern lauffähig, sie funktioniert problemlos in den aktuellen Versionen von Firefox, Safari und Chrome, aber nicht im Internet Explorer.

## 5.1 Beschreibung der GUI

Zunächst wird die VOMS-AJAX-GUI aus Benutzersicht beschrieben.
Abbildungen 5.1 und 5.2 zeigen das zentrale Bedienelement der Gruppen- und Rollenverwaltung. Es handelt sich um eine Tabelle, in welcher horizontal die Gruppen und Rollen und vertikal die Mitglieder eingetragen sind. Durch die „X“-Zeichen wird angezeigt, welches Mitglied in welcher Gruppe ist und welche Rollen es wahrnimmt. Jeder Spalte mit einer Gruppe folgen Spalten mit den dazugehörigen Rollen. Jeder Kombination aus Gruppe und Rollen folgen zunächst alle untergeordneten Gruppen mit ihren Rollen, bevor die nächste Gruppe der gleichen Hierarchieebene angezeigt wird. Um die Hierarchie besser sichtbar zu machen, werden die verschiedenen Ebenen mit unterschiedlichen Farbintensitäten dargestellt. Um eine desto tiefere Untergruppe es sich handelt, desto heller wird der Farbton. Die Rollen wiederum werden zur besseren Unterscheidbarkeit in einer gänzlich anderen Farbe angezeigt.
Das Verändern von Gruppenmitgliedschaften und Rollen erfolgt einfach über einen Klick auf die betreffende Tabellenzelle. Alle davon abhängigen Gruppen und Rollen werden automatisch gesetzt. Wird z.B. eine Rolle in einer bestimmten Untergruppe ausgewählt, wird die betreffende Person automatisch in eben diese Untergruppe, sowie alle übergeordneten Gruppen aufgenommen, sofern dies noch nicht der Fall ist. Wird dagegen z.B. die Mitgliedschaft in einer bestimmten Gruppe aufgehoben, werden auch die Rollen in eben dieser Gruppe, sowie alle untergeordneten Gruppenmitgliedschaften mit ihren Rollen aufgehoben.
Wird mit der Maus über den Namen einer Person bewegt, erscheint ein Tooltip, welcher zusätzliche Informationen zu dieser Person aus der VOMS-Datenbank enthält.
Um bei sehr umfangreichen VOs den Umfang der Tabelle reduzieren zu können gibt es zwei

Mechanismen. Einerseits können die Rolleninformationen zu jeder Gruppe individuell oder für die gesamte VO zentral ein- und ausgeblendet werden. Andererseits kann man die Tabelle nach bestimmten Personen filtern. Wird in das Formularfeld eine Zeichenfolge eingegeben, werden nur noch die Personen angezeigt, welche diese Zeichenfolge enthalten. Dabei wird nicht nur der Name der Person betrachtet, sondern auch die zusätzlichen Informationen, die über das Tooltip angezeigt werden. Auf diese Weise ist es möglich nur Personen anzuzeigen, die z.B. aus einer bestimmten Organisation sind oder die in einem bestimmten Land leben.

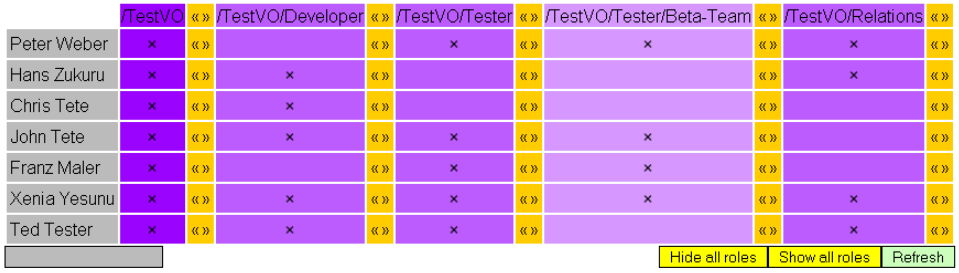



Abbildung 5.1: Gruppen- und Rollen-Verwaltung mit ausgeblendeten Rollen

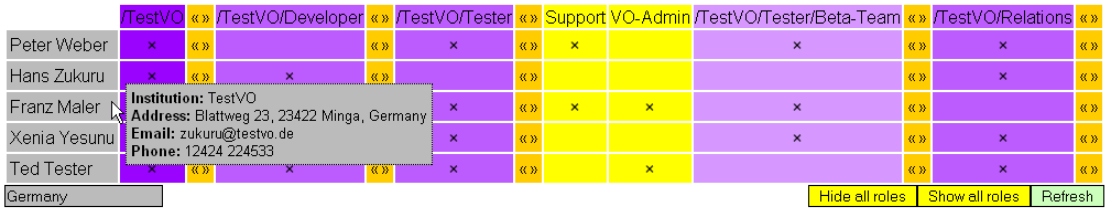
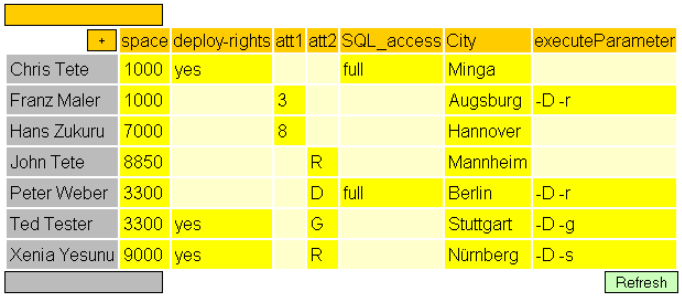



Abbildung 5.2: Gruppen- und Rollen-Verwaltung mit eingeblendeten Rollen einer Gruppe, Tooltip mit Mitgliederinformationen und aktiviertem Personenfilter

Abbildungen 5.3 und 5.4 zeigen das Bedienelement für die Attribut-Verwaltung. Identisch mit der Gruppen- und Rollenverwaltung ist die Anzeige der Personen auf der linken Seite. Auch hier können Tooltips mit zusätzlichen Informationen angezeigt werden und die Tabelle kann mit Hilfe des Personenfilters gekürzt werden.
Statt Gruppen und Rollen repräsentieren hier aber die Spalten verschiedene generische Attribute. Am Schnittpunkt eines Attributnamens und einer bestimmten Person wird der Wert angezeigt, der dieser Person zugeordnet ist. Ist das Attribut nicht gesetzt erscheint die entsprechende Tabellenzelle in einer blasseren Farbe. Um einen Wert zu bearbeiten, klickt man diesen an. Anstelle des einfachen Textes erscheint dadurch ein Formular, ein so genannter InPlace-Editor, mit welchem der Wert verändert werden kann. Klickt man auf „Submit“ verschwindet, dass Formular wieder, und es erscheint in der Tabelle der neue Wert. Auf die gleiche Weise ist es möglich die Namen der Attribute zu verändern. Durch Löschen des Namens eines Attributs kann man es entfernen. In der VOMS-Datenbank werden dann alle Wertzuordnungen gelöscht und die dazugehörige Spalte in der GUI entfernt. Durch das „+“-Zeichen ist es möglich ein neues Attribut anzulegen. Füllt man das in diesem Fall erscheinende Formular aus und schickt es ab, wird die Tabelle sogleich um eine neue Spalte ergänzt.

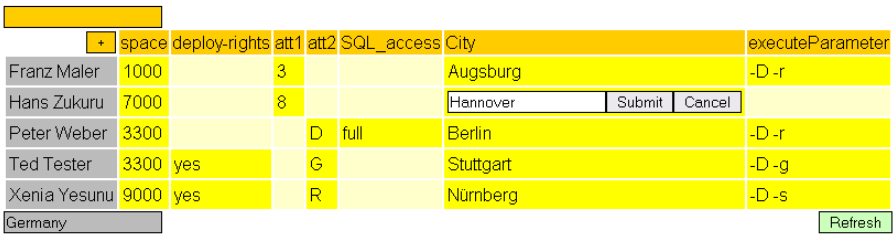
Zusätzlich zur Filtermöglichkeit nach Personen, kann man auch nach Attributnamen filtern. Durch die Kombination von Personen- und Attributfilter, kann man die Tabelle bis auf einen speziellen Attributwert, den man bearbeiten will, zusammenschrumpfen lassen.



| + | space | deploy-rights | att1 | att2 | SQL_access | City | executeParameter |
|---|---|---|---|---|---|---|---|
| Chris Tete | 1000 | yes | | | full | Minga | |
| Franz Maler | 1000 | | 3 | | | Augsburg | -D -r |
| Hans Zukuru | 7000 | | 8 | | | Hannover | |
| John Tete | 8850 | | | R | | Mannheim | |
| Peter Weber | 3300 | | | D | full | Berlin | -D -r |
| Ted Tester | 3300 | yes | | G | | Stuttgart | -D -g |
| Xenia Yesunu | 9000 | yes | | R | | Nürnberg | -D -s |

Refresh

Abbildung 5.3: Attribut-Verwaltung



| + | space | deploy-rights | att1 | att2 | SQL_access | City | executeParameter |
|---|---|---|---|---|---|---|---|
| Franz Maler | 1000 | | 3 | | | Augsburg | -D -r |
| Hans Zukuru | 7000 | | 8 | | | Hannover Submit Cancel | |
| Peter Weber | 3300 | | | D | full | Berlin | -D -r |
| Ted Tester | 3300 | yes | | G | | Stuttgart | -D -g |
| Xenia Yesunu | 9000 | yes | | R | | Nürnberg | -D -s |

Germany

Refresh

Abbildung 5.4: Attribut-Verwaltung mit aktiviertem InPlace-Editor und Personen-Filter

## 5.2 Konzeptionelle Unterschiede zwischen VOMS Admin und VOMS-AJAX-GUI

VOMS Admin und VOMS-AJAX-GUI unterscheiden sich konzeptuell in zwei wesentlichen Merkmalen. Das erste wurde bereits im letzten Kapitel sichtbar. In VOMS Admin werden Gruppen, Rollen und Attribute in Listenform angezeigt. Einerseits kann man sich eine Liste anzeigen lassen, in der die Gruppen und Rollen aufgezählt sind, denen das Mitglied angehört. Andererseits gibt es eine Liste mit den Mitgliedern einer bestimmten Gruppe oder den Mitgliedern, welche über ein bestimmtes Attribut verfügen. Diese Listendarstellung ermöglicht aber keinen gesamtheitlichen Blick auf die VO. In VOMS-AJAX-GUI wurde deshalb die Darstellung in Form einer Tabelle bzw. Matrix gewählt. Horizontal sind die Gruppen und Rollen bzw. die Attribute eingetragen, vertikal die Mitglieder. Auf diese Weise wird ein Überblick über die gesamte VO geschaffen. Man sieht z.B. auf einen Blick, sowohl welche Mitglieder in bestimmten Gruppen sind, als auch in welchen Gruppen bestimmte Mitglieder sind. Zudem sind sämtliche Änderungen mit einer geringeren Anzahl an Klicks durchführbar. In VOMS Admin würde man z.B. für das Zuweisen einer Rolle in einer Gruppe, in der die Person noch nicht Mitglied ist, bis zu sechs Klicks benötigen - in VOMS-AJAX-GUI genügt bei aktivierter Anzeige der Rollen ein einziger Klick, ansonsten zwei.

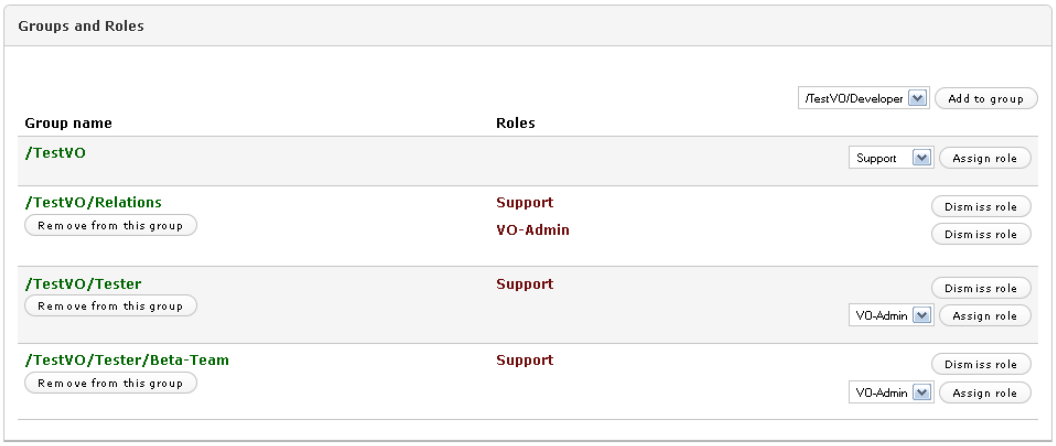



Abbildung 5.5: Gruppen- und Rollen-Verwaltung eines einzelnen Benutzers in VOMS Admin

Untersucht man die Funktionsweise von VOMS Admin, so kann man feststellen, dass diese Anwendung bereits an einigen Stellen Ajax nutzt. So wird z.B. beim Hinzufügen eines Benutzers zu einer Gruppe die Benutzer-ID und Gruppen-ID über einen Ajax-Request an den Server geschickt, dieser liefert ein HTML-Fragment, in diesem Fall die Aufzählung von Gruppen und Rollen des Mitglieds, zurück. Dieses HTML-Fragment ersetzt die veraltete Information auf der Seite. Ajax-Requests in VOMS Admin dienen somit lediglich dazu, das Neuladen der gesamten Seite zu verhindern. Die Darstellung der Ergebnisse und somit das Arbeiten wird dadurch beschleunigt.
In VOMS-AJAX-GUI wird das System dagegen in zwei Komponenten geteilt. Es gibt einerseits das Frontend, welches JavaScript-basiert ist und im Browser läuft und andererseits das Backend, deren Hauptaufgabe es ist, Informationen aus der VOMS-Datenbank abzufragen und darin Veränderungen vorzunehmen. Ajax-Requests bilden das Bindeglied zwischen diesen beiden Teilen. Nur über diese können Informationen aus der VOMS-Datenbank abgerufen oder darin verändert werden. Das zuerst normal vom Server abgerufene Dokument enthält keinerlei VO-spezifische Informationen. Des Weiteren werden nie HTML-Fragmente ausgetauscht, sondern lediglich einzelne Parameter oder XML-Daten. Um die Visualisierung dieser Daten, also z.B. die Umwandlung in die im vorherigen Kapitel gezeigten Tabellen, kümmert sich das Frontend auf Clientseite.

## 5.3 Funktionsprinzip

Im Folgenden wird das im letzten Kapitel angesprochene Funktionsprinzip vertieft und detaillierter erklärt - ohne jedoch dieses im weiteren mit VOMS Admin zu vergleichen. Die Funktionsweise wird hier aber auf einer eher abstrakten Ebene betrachtet, mit einzelnen Implementierungsdetails und Problemen wird sich das nächste Kapitel beschäftigen.
Wie bereits erwähnt lässt sich die VOMS-AJAX-GUI in ein JavaScript-basiertes Frontend und ein JSP-basiertes Backend aufteilen, wobei die Kommunikation zwischen beiden über Ajax-Requests realisiert ist. So gibt es sowohl für die Gruppen- und Rollenverwaltung, als auch für das Attributmanagement jeweils zwei Dateien auf dem Server: *groups.jsp* und *groups_ajax.jsp*, sowie *attributes.jsp* und *attributes_ajax.jsp*. Die Dateien ohne „ajax“-Zusatz

enthalten keinen JSP-Code (vom Setzen von HTML-Headern abgesehen) und werden somit jedes mal in gleicher Form und ohne VO-spezifische Informationen vom Client abgerufen. Sie beinhalten den benötigten anwendungsspezifischen JavaScript-Code (Frameworks und gemeinsam verwendete Methoden sind in externe Dateien ausgelagert) sowie ein einfaches HTML-Gerüst, welches unter anderem die Navigation enthält. Die Dateien mit „ajax“-Zusatz bilden das Backend, sie beinhalten den eigentlichen serverseitigen Code welcher für die Verarbeitung von Abfragen und Änderungswünschen und somit auch für die Kommunikation mit der VOMS-Datenbank verantwortlich ist.

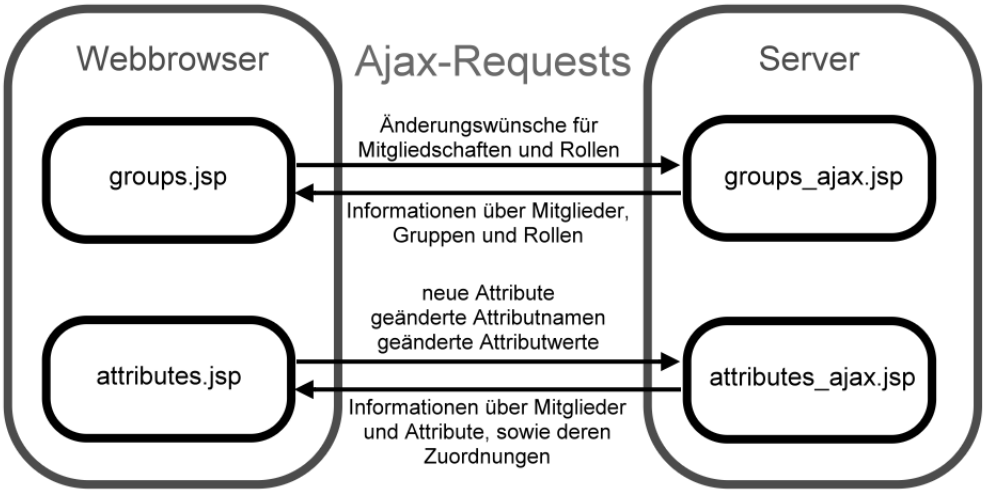



Abbildung 5.6: Ajax-Kommunikation zwischen Client und Server

Zunächst soll das Gruppen- und Rollenmanagement betrachtet werden. Zuerst ruft der Browser die Datei *groups.jsp* ab. Nach Erhalt werden sofort die ersten beiden XMLHttpRequests durchgeführt. Es werden dabei zwei XML-Dokumente abgerufen: das Erste enthält die Hierarchie der Gruppen und Rollen, auf deren Grundlage die verschieden gefärbten Spalten erzeugt werden. Das Zweite enthält Informationenen zu den einzelnen Personen - neben den persönlichen Informationen, welche über das Tooltip angezeigt werden, sind dies die Mitgliedschaften und Rollen.
Beim Erzeugen der Tabelle wird den Zellen eine ID zugewiesen die sich aus Zellenart (Gruppe oder Rolle), Personen-ID, Gruppen-ID und gegebenenfalls Rollen-ID zusammensetzt. Anschließend werden für die Zellen Event-Listener initialisiert, welche bei einem Klick auf eine Zelle einen Ajax-Request ausführen, bei dem eben diese Zellen-ID übertragen wird. Das Backend teilt die zusammengesetzte ID in ihre Bestandteile auf und führt in der VOMS-Datenbank in Abhängigkeit von den aktuell gesetzten Werten rekursiv die benötigten Änderungen aus. Anschließend wird eine XML-Datei erzeugt, welche die neuen Gruppen und Rollen der betroffenen Person enthält. Diese wird als Antwort auf den Ajax-Request zurückgeliefert, vom Frontend daraus eine neue Tabellenzeile erzeugt und mit dieser die Alte ersetzt. Der Personenfilter, welcher mit Regular Expressions realisiert ist, und das Auf- und Zuklappen der Rollen, bei welchen Tabellenzellen unsichtbar gesetzt werden, arbeiten rein lokal und erfordern keine Kommunikation mit dem Backend.

Das Attributmanagement funktioniert sehr ähnlich, wobei hier die Tabellenzellen den InPlace-Editor enthalten, welcher vom Scriptaculous-Framework zur Verfügung gestellt wird. Zusätzlich zu der kombinierten ID wird der neue Wert an den Server übertragen. Es wird allerdings kein neuer Datensatz über die Person zurückgeliefert, sondern lediglich der neue Wert als Bestätigung. Das Einfügen des neuen Wertes in die Tabelle ist Aufgabe des InPlace-Editors. Ähnlich funktioniert das Ändern des Attributnamens und das Hinzufügen neuer Attribute. Das Übertragen eines leeren Attributnamens ist ein Sonderfall, da dies das Löschen des Attributs bedeutet. In diesem Fall werden vom Backend in der Datenbank sowohl das Attribut als auch alle dazugehörigen Werte gelöscht, das Frontend entfernt die Tabellenspalte.

Das Backend greift direkt auf die VOMS-Datenbank zu, so dass der Zugriff unabhängig vom VOMS Admin oder von auch einem gestartetem VOMS-Server ist. Die VOMS-AJAX-GUI nutzt und schreibt nur einen Teil der Informationen, die in der VOMS-Datenbank enthalten sind. Die wichtigsten Tabellen der VOMS-Datenbank sind „m“, welche die Zuordnungen von Benutzern, Gruppen und Rollen enthält, und „usr_attrs“, welche die einzelnen, den Personen zugeordneten, Attributwerte enthält. Um benötigte Informationen zu ermitteln müssen meistens diese Tabellen mit anderen gejoint werden. Ein SQL-Statement wie es vom Backend verwendet wird sieht z.B. folgendermaßen aus:

```
SELECT usr.userid, name, surname, institution, address,
email_address, phone_number, gid, rid
FROM usr LEFT OUTER JOIN m ON usr.userid=m.userid;
```

## 5.4 Beschreibung der einzelnen Funktionen

Im Folgenden werden kurz die Aufgaben der einzelnen Methoden und ihr Zusammenspiel erläutert. Dabei wird auf erwähnenswerte Details besonders eingegangen. Die Erläuterungen sind nach den Dateien gegliedert, in welchen sie vorkommen. Zuerst werden die JavaScript-Funktionen des Frontends, anschließend der Java-Code des Backends erläutert.
Zu den Backend-Dateien ist noch Folgendes anzumerken: bei der Verarbeitung von JSP-Dateien fügt der Server häufig Leerzeilen oder sonstigen Whitespace in die Antwort ein. Jede page-Direktive, welche am Anfang der Datei steht wird z.B. durch eine Leerzeile ersetzt. Neuere Tomcat-Versionen bieten eine Direktive um dieser Verhalten zu unterbinden, in der eingesetzten Version ist diese aber noch nicht vorhanden. Bei normalen HTML-Datei ist dieser zusätzliche Whitespace nicht störend, bei der Übertragung von XML-Daten aber umso mehr. Die JavaScript-Engines der Browser verweigern nämlich das Verarbeiten der Daten, wenn vor dem XML-Prolog eine Leerzeile vorhanden ist. Um dieses Problem zu umgehen, werden die Direktiven am Anfang jeder Datei mitten im Tag umgebrochen, so dass nach Abarbeitung einer Direktive die Zeile nicht leer ist und so keine Leerzeilen eingefügt werden. Sollte nach irgendwelchen Veränderungen an Quelltext plötzlich die Anzeige der Tabellen nicht mehr funktionieren, hat sich sehr wahrscheinlich eine Leerzeile an den Anfang der Ausgabe eingeschlichen.

**groups.jsp**

`initTable()`
Diese Funktion initialisiert die Tabelle. Dazu wird zunächst mit Hilfe des Prototype-Frameworks ein Ajax-Request durchgeführt, bei welchem die Gruppen und Rollen in Form von XML-Daten abgefragt werden und daraus die Kopfzeile der Tabelle und parallel ein Template für die weiteren Tabellenzeilen erstellt. Genutzt wird hierbei die Template-Klasse von Prototype. Die Werte für die verschiedenen Variablen in diesem Template werden mit dem nächsten Ajax-Request abgerufen, nämlich die Informationen über die einzelnen Mitglieder mit ihren Gruppen und Rollen. Mit diesen Daten werden mit Hilfe des Templates alle Tabellenzeilen erstellt und die Tabelle abschließend ins Dokument eingefügt.

`toggler(part)`
Diese Funktion erstellt Event-Listener für die Tabellenzellen, durch deren Anklicken die Rollen ein- und ausgeblendet werden. Die Auswahl dieser Zellen erfolgt über CSS-Klassen und das name-Attribut (sämtliche Rollenzellen einer Gruppe haben das gleiche name-Attribut).
@param `part` Selektor für den Teil der Tabelle, auf welchem die Funktion ausgeführt werden soll

`toggleRecover(part, source)`
Nach dem Erzeugen einer neuen Tabellenzeile mit aktualisierten Personendaten sind in dieser zunächst alle Rollen ausgeblendet. Diese Funktion überträgt diese Eigenschaften aus einer anderen Tabellenzeile.
@param `part` Selektor der Tabellenzeile bei der die Rollenanzeige wiederhergestellt werden soll
@param `source` Selektor der Tabellenzeile aus der die Rollenanzeige übernommen werden soll (in der Regel die Kopfzeile)

`toggleAllRoles(action)`
Durch diese Funktion können sämtliche Rollen in der Tabelle ein- und ausgeblendet werden.
@param `action` „hide“ oder „show“ je nach gewünschter Aktion

`createDatasetTR(subject)`
Diese Funktion erzeugt mit Hilfe des in der Funktion initTable() erstellten Tabellenzeilen-Templates eine einzelne Zeile.
@param `subject` XML-Element mit Informationen über die Person

`setter(part)`
Diese Funktion erstellt Event-Listener für die Tabellen-Zellen mit den Kreuzen. Bei Anklicken einer solchen Zelle wird die zusammengesetzte ID dieser Zelle per Ajax an den Server übertragen und mit den erhaltenen neuen Daten die Tabelle aktualisiert. Die Funktion ruft sich rekursiv selbst auf, um für die neu erstellte Tabellenzeile neue Event-Listener zu erstellen.
@param `part` Selektor für den Teil der Tabelle, auf welchem die Funktion ausgeführt werden soll

`refresh()`
Hierbei wird die Tabelle neu initialisiert und sämtliche Filter zurückgesetzt.

**attributes.jsp**

`initTable()`
Diese Funktion initialisiert die Tabelle. Sie funktioniert nach dem gleichen Prinzip wie bei der Gruppen- und Rollenverwaltung.

`editing()`
Diese Funktion erstellt InPlace-Editoren in der Tabelle. Dabei wird unterschieden, ob es sich um einen Attributnamen, einen Attributwert oder um das Eingabefeld für neue Attribute handelt. Der InPlace-Editor wird vom Scriptaculous-Framework zur Verfügung gestellt. Die Optionen `onEnterHover` und `onLeaveHover` werden in der Scriptaculous-Dokumentation nicht aufgeführt, sie dienen dazu die standardmäßig auftretenden Highlight-Effekte zu verhindern, die in diesem Fall stören würden.

`changeBackground(transport, ele)`
`adjustTable(transport, ele)`
Diese Funktion werden vom InPlace-Editor entsprechend der onComplete-Option nach Erhalt der Antwort vom Server automatisch aufgerufen und die Parameter passend gesetzt.
Erstere verändert die Hintergrundfarbe einer Tabellenzelle, je nach dem ob der neue Wert ein leerer String ist oder nicht. Letztere entfernt eine Tabellenspalte wenn der neue Attributname ein leerer String ist.
@param `transport` Das Transport-Objekt des durch den InPlace-Editor durchgeführten Ajax-Requests
@param `ele` HTML-Element des InPlace-Editors

`attributeFilter()`
Diese Funktion erstellt einen Event-Listener für das Attributfilter-Eingabefeld. Die Attributnamen werden mit Hilfe eines regulären Ausdrucks mit der Eingabe verglichen und die Spalten entsprechend ein- oder ausgeblendet. Bei einem leeren Eingabefeld werden alle Spalten angezeigt.

`refresh()`
siehe Gruppen- und Rollenverwaltung

**shared.js**
Die Datei shared.js enthält Funktionen, die sowohl von der Gruppen- und Rollen-, als auch von der Attributverwaltung benötigt werden.

`tooltip(part)`
Diese Funktion erstellt einen Event-Listener, der bei einem mousemove-Event über der Zelle mit dem Mitgliedernamen ein der Maus folgendes Tooltip anzeigt, welches aus einem div-Element in dieser Zelle erzeugt wird.
@param `part` Selektor für den Teil der Tabelle, auf welchem die Funktion ausgeführt werden soll

`nameFilter()`
Diese Funktion erstellt einen Event-Listener für das Personenfilter-Eingabefeld. Es wird mit Hilfe von regulären Ausdrücken geprüft, ob der eingegebene Wert im Namen oder in den zusätzlichen Informationen, die im Tooltip angezeigt werden, vorkommt, und daraufhin die Tabellenzellen entsprechend ein- oder ausgeblendet. Bei einem leeren Eingabefeld werden alle Zeilen angezeigt.

**groups_ajax.jsp**
Für JSP typisch ist der Code nicht in mehrere Methoden unterteilt. Bei Abruf dieser Datei durch einen Ajax-Request muss einer der folgenden GET-Paramter übertragen werden, diese geben an welcher Code-Block ausgeführt werden soll.

`groups`
Dieser Block ruft die existierenden Gruppen und Rollen aus der VOMS-Datenbank ab, erstellt aus diesen Informationen ein XML-Dokument und sendet dieses als Antwort an den Client. Die Untergruppen-Elemente werden dabei innerhalb des Elements der übergeordneten Gruppe platziert. Für eine einfachere Verarbeitung auf Clientseite wird aber die Ebene, auf welcher sich die Gruppe befindet, zusätzlich in Form eines Attributs angegeben.

`subjectsWithGroups`
Diese Block ruft sämtliche Personen und ihre Gruppen und Rollen aus der VOMS-Datenbank ab und erstellt auf Grundlage dieser Informationen ein XML-Dokument, welches als Antwort an den Client gesendet wird.

`grouproleEditValue`
Bei Aufruf dieses Blocks muss zusätzlich eine Zellen-ID als Parameter übertragen werden. Die zusammengesetzte Zellen-ID wird in ihre Bestandteile gesplittet und passend dazu, ob es sich um eine Gruppen- oder eine Rollenzelle handelt, die Gruppe bzw. Rolle je nach Ursprungszustand gesetzt oder entfernt. Des Weiteren müssen, um sämtliche Abhängigkeiten wie unter- und übergeordnete Gruppen und Rollen passend zu setzen, rekursiv mehrere Datenbankabrufe und Datenbank-Updates durchgeführt werden. Es ist von besonderer Wichtigkeit, dass dies korrekt funktioniert, da anders als diverse andere Abhängigkeiten, die in der VOMS-Datenbank bestehen, diese nicht von der Datenbank, z.B. über Fremdschlüssel, sichergestellt werden. Wurden alle notwendigen Änderungen an der Datenbank durchgeführt, werden die neuen Gruppenmitgliedschaften und Rollen der betreffenden Person erneut von der Datenbank abgefragt und im gleichen XML-Format übertragen wie bei „subjectsWithGroups“ - nur mit dem Unterschied, dass in diesem Fall nur die Informationen über diese eine Person enthalten sind.

**attributes_ajax.jsp**
Auch bei Abruf dieser Datei durch einen Ajax-Request muss ein passender GET-Paramter übertragen werden, der angibt welcher Block ausgeführt werden soll.

`attributes`
Dieser Block ruft sämtliche Attribute aus der Datenbank ab und sendet diese in XML-Form an den Client.

`subjectsWithAttributes`
Ähnlich wie bei „subjectsWithGroups“ werden hier Informationen über die Personen abgerufen, nur dass statt Informationen über Gruppen und Rollen, hier die gesetzten Attribute aufgeführt werden. Ist ein Attribut für eine Person nicht gesetzt, so wird dieses nicht aufgezählt.

`attributeEditValue`
Bei Aufruf dieses Blocks muss zusätzlich die zusammengesetzte ID des InPlace-Editors übertragen werden, welche Personen-ID und Attribut-ID enthält, sowie der neue Wert des Attributs. Je nach Ursprungszustand und je nach dem, ob der neue Wert ein leerer String ist, muss unter Umständen ein neuer Datensatz angelegt oder ein alter gelöscht werden.

`attributeEditName`
Bei Aufruf dieses Blocks werden die gleichen Parameter gefordert wie beim vorherigen. Hier wird der Name eines Attributs geändert oder das Attribut gelöscht, falls ein leerer String als neuer Wert gesetzt wird. Im Falle einer Löschung müssen zusätzlich alle Wertzuordnungen in der Datenbank gelöscht werden.

`attributeNew`
Bei Aufruf dieses Blocks wird ein neues Attribut angelegt, dessen Name in einem zusätzlichem Parameter übertragen werden muss.

# 6 Neue Möglichkeiten durch den Einsatz von Ajax am Beispiel des Abrufs von SAML Assertions

Im Folgenden geht es um den zweiten Schwerpunkt dieser Arbeit: den Abruf von SAML-Assertions mit Hilfe von Ajax. Demonstriert wird dies wieder an Hand der VOMS-AJAX-GUI, wobei sie in diesem Fall als Endbenutzer-Oberfläche dient und nicht primär für den Administrator gedacht ist.

Wie bereits am Anfang dieser Arbeit erläutert, ist es Aufgabe eines VOMS Servers Proxy-Zertifikate auszustellen, die für die Autorisierung von Grid-Jobs benötigt werden. Mit dem Tool voms-proxy-init kann man beim Abruf eines solches Zertifikats festlegen, welche Gruppen und Rollen im Proxy-Zertifikat aufgeführt werden sollen. Grids werden multiinstutional betrieben und so kann es natürlich auch gegenseitige Begehrlichkeiten geben. So kann es sein, dass ein VO-Mitglied bei der Anmeldung an einer Ressource nicht sichtbar machen will, dass es Mitglied einer bestimmten Untergruppe ist, da der Resource-Provider möglicherweise aus dieser Angabe nicht für ihn bestimmte Informationen ableiten könnte. Ebenfalls denkbar ist es, dass ein Mitglied in Abhängigkeit davon mit welcher Rolle er sich bei einem Resource-Provider anmeldet, unterschiedliche Dienstgüten oder Kapazitäten zur Verfügung gestellt bekommt.
Der neuere Ansatz ist nun an Stelle von Proxy-Zertifikaten SAML-Assertions zu verwenden, wie sie von der SAML-Schnittstelle des VOMS Admin zur Verfügung gestellt werden. Auch hier möchte man Festlegen welche Angaben die Assertion enthält. Und da es sich bei SAML-Requests und SAML-Assertions schließlich um XML-Daten handelt, soll in dieser Arbeit gezeigt werden, das man sowohl die Zusammenstellung der gewünschten Assertion als auch den Abruf komfortabel mit Ajax erledigen kann.
Zunächst sei eine kurze Erklärung der etwas verwirrenden doppelten Verwendung des Begriffs Attribut eingeschoben. Bei SAML sind alle Eigenschaften des Subjekts Attribute, dies können zwar die generischen Attribute aus der VOMS-Datenbank sein, aber auch Gruppen oder Rollen. Um diese beiden Arten von Attributen zu unterscheiden, wird im Folgenden immer die Rede von generischen Attributen sein, wenn eben diese gemeint sind. Die SAML-Attribute werden einfach Attribute genannt.

## 6.1 Beschreibung der GUI

Zunächst wird wieder beschrieben, wie die entwickelte GUI aussieht und funktioniert, bevor die technischen Hintergründe erläutert werden.
Wie in Abbildung 6.1 zu sehen, sieht der Nutzer beim Aufruf des SAML Assertion Requesters eine Tabelle mit den Parametern mit welchen die SAML Query erstellt wird - somit

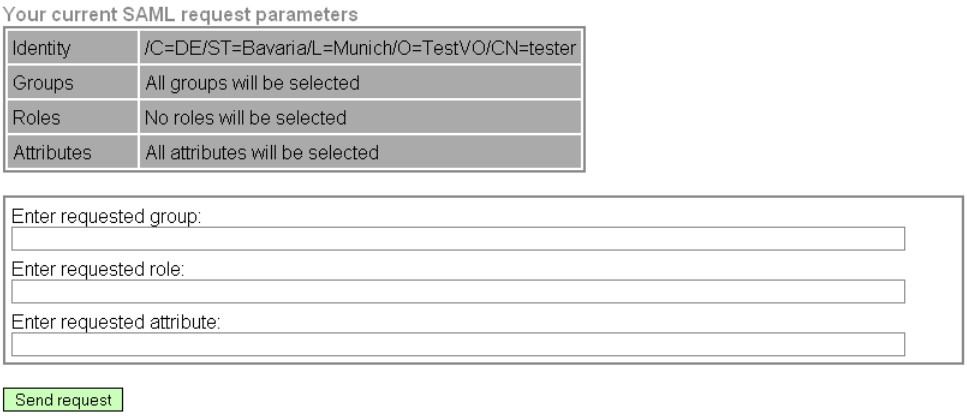



Abbildung 6.1: SAML Assertion Requester beim Abruf

sind dies die Attribute die mit der Assertion zugesichert werden. Die Tabelle enthält zur Bestätigung seinen Distinguished Name und die geforderten Gruppen, Rollen und generischen Attribute. Standardmäßig werden alle Gruppen und generischen Attribute, aber keine Rollen in die Assertion aufgenommen. Unterhalb befinden sich drei Formularfelder, über welche konkrete Werte für die Gruppen, Rollen und generischen Attribute ausgewählt werden können. Hierzu wird der Name der gewünschten Eigenschaften eingegeben und bereits nach dem ersten Buchstaben wird eine Autovervollständigung aktiv und dem Benutzer werden sämtliche Eingabemöglichkeiten, die ihm zur Verfügung stehen, angezeigt (siehe Abbildung 6.2). Der Nutzer kann die Eingabe zu Ende schreiben oder einen der Vorschläge auswählen. Hat er dies gemacht, erscheint sogleich seine Wahl in der Tabelle mit den Parametern. Diesen Vorgang kann er beliebig oft wiederholen und auch bereits ausgewählte Parameter wieder löschen. Wählt er eine Rolle aus, wird die dazugehörige Gruppe automatisch in die Tabelle übernommen. Löscht er eine Gruppe, werden sämtlich dazugehörigen Rollen gelöscht.
Ist die Auswahl abgeschlossen, kann schließlich die Assertion abgerufen werden, die unterhalb der bisherigen Bedienelemente erscheint. Die Gültigkeitsdauer, die in der Assertion enthalten ist, wird zudem separat, in lokale Zeit umgerechnet, angezeigt. Diesen unteren Teil der Seite zeigt Abbildung 6.3.

## 6.2 Funktionsprinzip

Die Architektur des SAML Attribute Requesters (im Folgenden nur SAML Requester genannt) ist vergleichbar mit der der bereits erläuterten administrativen Funktionen. Es gibt zwei Dateien, *saml.jsp* und *saml_ajax.jsp*, welche wieder das Frontend und das Backend bilden. Die Kommunikation zwischen Frontend und Backend ist hier aber auf den Abruf von Werten für die Autovervollständigung beschränkt. Als dritte Komponente kommt eine SOAP-Schnittstelle hinzu, über welche man SAML Assertions abfragen kann. Diese Schnittstelle wird VOMSSaml genannt. Abbildung 6.4 zeigt die Kommunikation zwischen Client und Server. Zu beachten ist die Trennung des Servers in zwei Teile. Der SAML Requester ist Teil der VOMS-AJAX-GUI, die VOMSSaml-Schnittstelle ist Bestandteil von VOMS Admin - beides sind voneinander unabhängige Systeme. Beide müssen aber auf dem selben Server

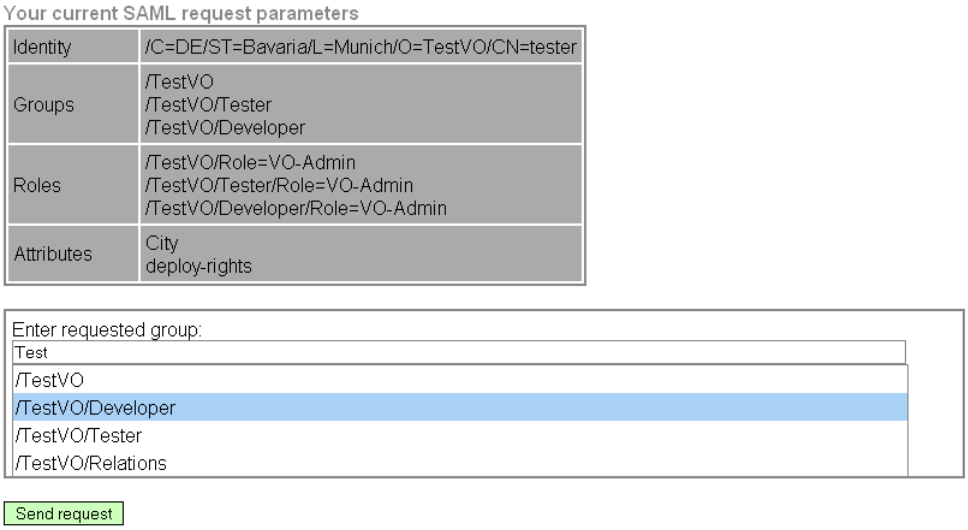



Abbildung 6.2: SAML Assertion Requester mit diversen bereits getätigten Auswahlen und aktivierter Autovervollständigung

unter der selben Domain laufen, da ansonsten der Abruf der Assertion auf Grund der Same Origin Policy, welche Ajax-Request aus Sicherheitsgründen an eine andere Domain, als die der eigentlichen Website verhindert, scheitern würde.
Die Authentifizierung an beiden Serverteilen erfolgt separat über die SSL-Session. Ein Problem ist allerdings, dass der DN aus der Session im üblichen durch Kommas getrenntem Format vorliegt, dieser aber in der VOMS-Datenbank im Slashed-Format gespeichert ist. Aus diesem Grund enthält bereits die Datei *saml.jsp* etwas JSP-Code, welcher den DN aus der SSL-Session einer Identität in der VOMS-Datenbank zuordnet. Da die DNs dem JavaScript-Code auf Clientseite bekannt sein müssen, werden diese Werte in den JavaScript-Code eingefügt, bevor dieser zum Client gesendet wird.
Auf Clientseite werden sämtliche Parameter, die für den Request benötigt werden, in einer prototypen-basierten Datenstruktur abgespeichert. Prototypen beziehen sich hier nicht auf das verwendete JavaScript-Framework, sondern auf die klassenlose Objektorientierung, welche das Programmierparadigma von JavaScript ist. Diese Datenstruktur enthält Informationen zur Identität, sowie die gewünschten Gruppen, Rollen und generischen Attribute. In ihr wird auch die Beziehung von Rollen und Gruppen abgebildet. Die Autovervollständigung, mit deren Hilfe neue Werte hinzugefügt werden können, wird von Scriptaculous zur Verfügung gestellt und arbeitet ajax-basiert. Die einzige Aufgabe des Backends beim SAML Requester ist auf die Anfragen dieses Autocompleters zu antworten. Dabei werden die Namen der Gruppen, Rollen und generischen Attribute zurückgeliefert, auf die die Eingabe im Formularfeld und die auf den Benutzer zutreffen. Bei generischen Attributen wird auch der Wert des Attributs übermittelt.
Hat der Nutzer seine Eingaben getätigt, wird an den VOMSSaml mittels Ajax-Request eine SOAP-Nachricht mit einer SAML Query gesendet. VOMSSaml liefert die dazu passende Response und im Erfolgsfall wird dem Nutzer die SAML Assertion angezeigt. Clientseitig wird abschließend noch die Gültigkeitsdauer der Assertion, die im Element `Conditions` abgelegt ist, ausgelesen und in die lokale Zeit umgerechnet und angezeigt.


Abbildung 6.3: Anzeige der Assertion und des Gültigkeitszeitraums in lokaler Zeit

Ein beispielhafter SAML Request, wie er von der VOMS-AJAX-GUI erzeugt wird, ist im Folgenden aufgeführt. Beim Senden an die SAML-Schnittstelle ist dieses XML-Fragment noch in eine SOAP-Nachricht eingebettet.

```
<samlp:AttributeQuery
xmlns:samlp="urn:oasis:names:tc:SAML:2.0:protocol"
ID="94434471.84763865" IssueInstant="2010-08-09T12:53:53"
Version="2.0">
  <saml:Issuer
  xmlns:saml="urn:oasis:names:tc:SAML:2.0:assertion"
  Format="urn:oasis:names:tc:SAML:1.1:nameid-format:X509SubjectName">
    CN=tester, O=TestVO, L=Munich, ST=Bavaria, C=DE
  </saml:Issuer>
  <saml:Subject
  xmlns:saml="urn:oasis:names:tc:SAML:2.0:assertion">
    <saml:NameID
    Format="urn:oasis:names:tc:SAML:1.1:nameid-format:X509SubjectName">
        CN=tester, O=TestVO, L=Munich, ST=Bavaria, C=DE
    </saml:NameID>
  </saml:Subject>
  <saml:Attribute
  xmlns:saml="urn:oasis:names:tc:SAML:2.0:assertion"
  Name="http://voms.forge.cnaf.infn.it/fqan"
  NameFormat="urn:oasis:names:tc:SAML:2.0:attrname-format:uri">
    <saml:AttributeValue
    xmlns:xs="http://www.w3.org/2001/XMLSchema"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:type="xs:string">
        /TestVO
    </saml:AttributeValue>
  </saml:Attribute>
</samlp:AttributeQuery>
```

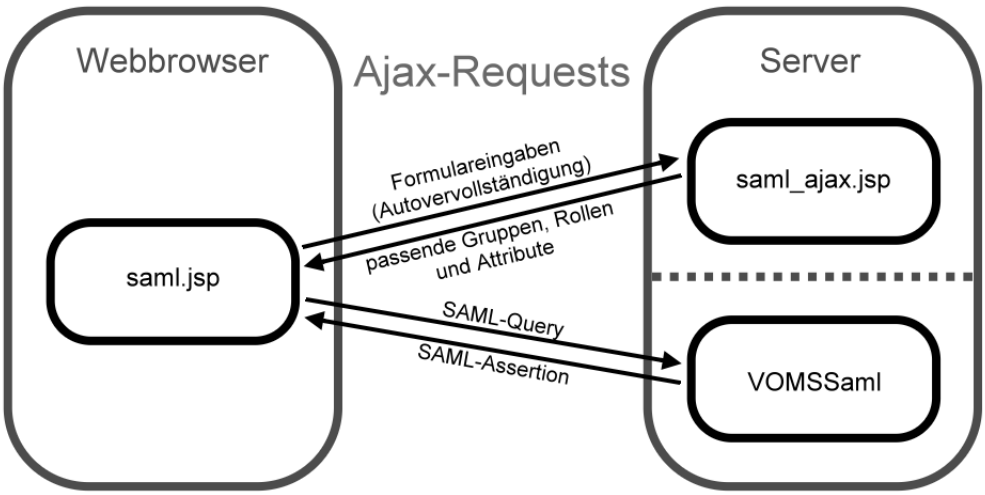



Abbildung 6.4: Ajax-Kommunikation zwischen Client und VOMS-AJAX-GUI sowie VOMS-Saml auf Serverseite

An diesem Beispiel sieht man einiges, um das sich das erzeugende JavaScript kümmern muss. Als Attribute des Elements `AttributeQuery` müssen eine eindeutige ID und die aktuelle Zeit (`IssueInstant`) übertragen werden. `Issuer` und `Subject` müssen identisch sein und im gleichen Format angegeben werden, wie in der SSL-Session. Das Slashed-Format aus der VOMS-Datenbank wird nicht akzeptiert. Die Angabe des Elements `Attribute` ist optional: erfolgt sie nicht, liefert der VOMS Server alle Gruppen und generischen Attribute, aber keine Rollen. Möglich ist es nur den Namen eines Attributs anzugeben, um alle Werte dieses Attributs zu erhalten oder diesen mit einem oder mehreren bestimmten Werten (`AttributeValue`) zu kombinieren, um nur diese zu erhalten. Die Gruppen und Rollen sind in der FQAN-Syntax (also z.B. /TestVO/untergruppe/Role=VO-Admin) anzugeben. Die verwendete Version der SAML-Schnittstelle funktioniert aber nicht richtig und liefert zu den Queries unpassende Antworten. Hierzu später mehr in einem weiteren Kapitel.
Auf das Abdrucken einer SAML Assertion wird hier verzichtet, da eine solche sehr lang ist und selbst bei einer Kürzung der darin enthaltenen Zertifikate mehrere Seiten einnehmen würde.

## 6.3 Einsatzmöglichkeiten

Es stellt sich die Frage, wie man eine solche mit Hilfe von Ajax abgerufene Assertion nutzen kann. Natürlich kann man diese in einer Datei speichern und dann im Unfeld einer anderen Anwendung nutzen. Eine andere Möglichkeit ist, diese direkt für einen Login zu verwenden. Die Grafik 6.5 zeigt wie dies funktionieren kann.
Der Nutzer ruft einen Grid-Dienst auf und wird von diesem zur VOMS-AJAX-GUI seiner VO umgeleitet. Die vom Grid-Dienst angegebene URL zum SAML Requester enthält hierbei einen zusätzlichen GET-Parameter, durch welchen dem SAML Requester mitgeteilt wird, wohin die fertige Assertion zu schicken ist. Im SAML Assertion Requester kann er sich seine Assertion nach seinen Wünschen zusammenstellen, worin er durch den Autocompleter

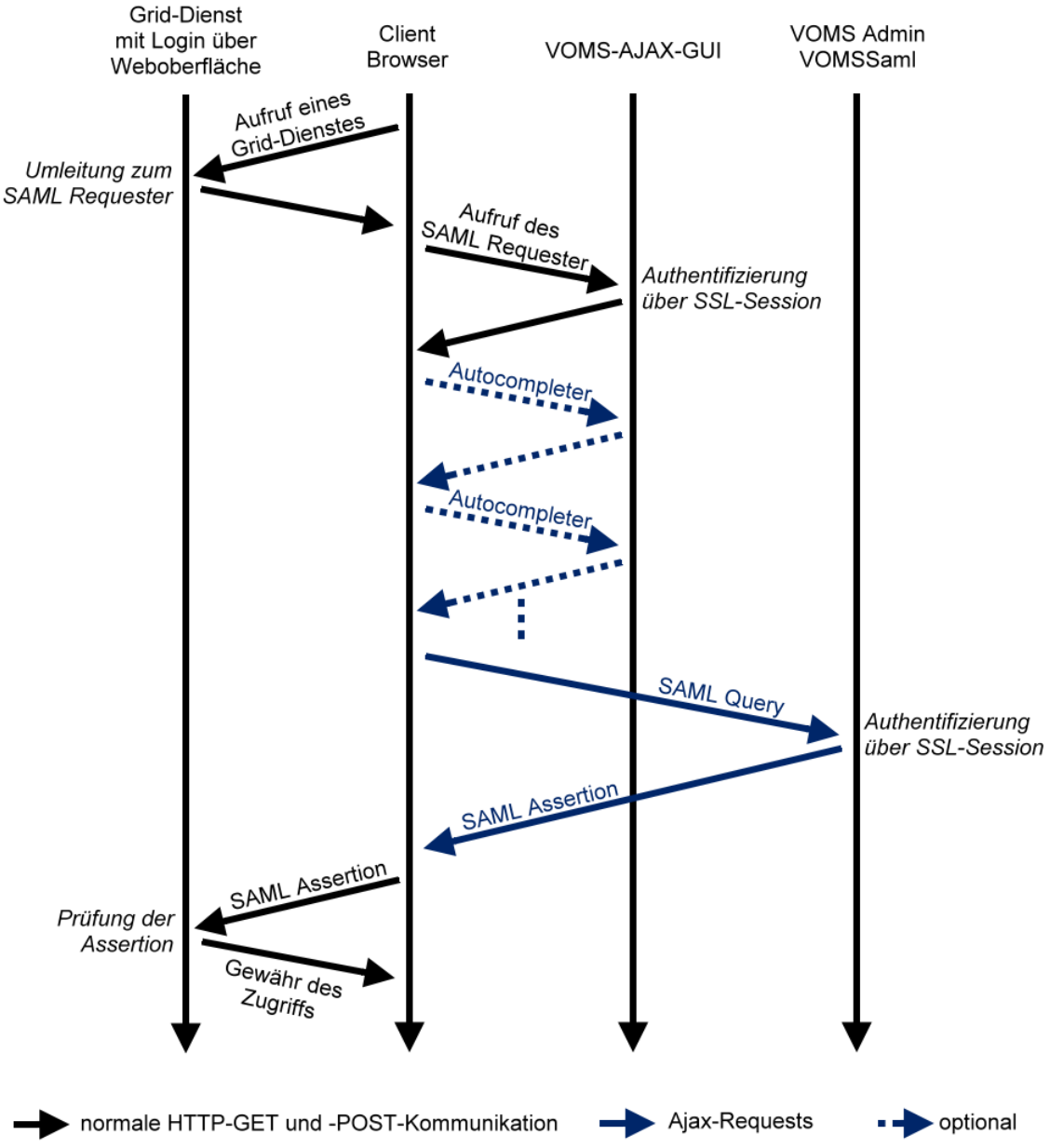



Abbildung 6.5: Ablauf des Logins bei einem Grid-Dienst unter Nutzung des SAML Assertion Requesters

unterstützt wird. Schließlich ruft er die Assertion ab, welche ihm anschließend zur Prüfung zusammen mit der Gültigkeitsdauer angezeigt wird. Ist er mit der Assertion zufrieden, sendet er sie mittels eines gewöhnlich HTTP-POST-Requests an den Grid-Dienst.
Im Verzeichnis der VOMS-AJAX-GUI befindet sich eine Datei namens *exampleservice.jsp*, welche stellvertretend für einen Grid-Dienst steht. Durch den Aufruf dieser Datei kann die Funktionsweise des hier beschriebenen Login-Verfahrens getestet werden.

## 6.4 Beschreibung der einzelnen Funktionen

**saml.jsp**
Am Anfang dieser Datei befindet sich JSP-Code, welcher die SSL-Identität einer Identität in der VOMS-Datenbank zuordnet und die ermittelten DNs in den JavaScript-Code einfügt. Der Einfachheit halber wird hier die Zurordnung nur auf Grundlage des Common Name (CN) durchgeführt, in einer produktiven Implementierung muss dies aus Sicherheitsgründen natürlich gründlicher erfolgen. Außerdem wird an den JavaScript-Code ggf. der login-Parameter für das Ziel der SAML Assertion übergeben.
Direkt nach dem Abruf der Datei werden zunächst die Autocompleter für die Formulare zum Eingeben der Gruppen, Rollen und genrischen Attribute erzeugt. Anschließend werden einige Templates erzeugt, die der Erstellung der SAML-Queries dienen. Zum Schluß wird die zentrale Datenstruktur Config initialisiert, wofür die vom JSP-Code eingefügten Werte verwendet werden.

```
function Config()
function Group()
...
```

Hierbei handelt es sich um die prototypen-basierte Datenstruktur, in welcher alle Attribute, sowie weitere Informationen abgelegt werden. Einem Nutzer können z.B. mehrere Groups zugerodnet werden, die Groups wiederum enthalten ausgewählte Rollen. Durch passende Funktionen zum Setzen, Abfragen und Entfernen der Werte wird auch sichergestellt, dass die Abhängigkeiten erhalten bleiben.

`chosen(input, li)`
Diese Funktion wird, sobald eine Auswahl getroffen wurde, vom Autocompleter entsprechend der afterUpdateElement-Option aufgerufen und die Parameter automatisch gesetzt. Sie fügt die Angaben an der passenden Stelle in die Datenstruktur ein.
@param `input` ID des beroffenen Eingabefeldes
@param `li` der ausgewählte Eintrag der Liste möglicher Autovervollständigungen

`addParams(input, query)`
Dies ist eine einfache Hilfsfunktion die vom Autocompleter aufgerufen wird und die Parameter des Ajax-Request um die Identität ergänzt. Die Parameter werden automatisch vom Autocompleter gesetzt.
@param `input` ID des beroffenen Eingabefeldes
@param `query` standardmäßiger Query-String

`updateTable()`
Diese Funktion wird aufgerufen, sobald Änderungen an der Datenstruktur vorgenommen wurden. Sie liest sämtliche benötigten Informationen aus der Datenstruktur aus und aktualisiert mit diesen Daten die Übersichtstabelle mit den gewählten Query-Parametern.

`createAttributeTag()`
Diese Funktion erstellt mit Hilfe der beim Aufrufen der Seite erstellten Templates und auf Grundlage der Informationen in der Datenstruktur die Attribute-Tags für die SAML Query. Die gewählten generischen Attribute werden wegen des fehlerhaften VOMSSaml ignoriert - mehr dazu im nächsten Kapitel.

`requester()`
Dies ist die zentrale Funktion in dieser Datei. Sie erstellt einen Event-Listener für den Assertion-Abfrage-Button. Wird dieser geklickt, werden zunächst einige für die Query benötigten Angaben, wie die Zeit im benötigten Format und eine eindeutige ID, erzeugt. Diese Angaben werden genutzt, um mit den entsprechenden Templates eine SOAP-Nachricht mit einer SAML Query zu erzeugen, die mittels eines Ajax-Requests an VOMSSaml gesendet wird. Wird eine SAML Assertion zurückgeliefert, so wird aus der Assertion die Gültigkeitsdauer ermittelt und diese mit Hilfe der JavaScript-Funktion `toLocaleString()` in lokale (Betriebssystem-)Zeit umgerechnet und angezeigt. Außerdem wird die Assertion serialisiert und ebenfalls angezeigt.
Wurde beim Aufruf des SAML Requesters der GET-Parameter login übertragen, wird die darin angegebene URL genutzt um ein Formular zu erzeugen, über welches die Assertion an eben jene URL geschickt werden kann.

**saml_ajax.jsp**
Der Backend-Teil des SAML Requesters hat nur eine Aufgabe und ist deshalb an die Funktionsweise des Autocompleters von Scriptaculous angepasst. Von diesem erhält er einen Parameter mit der Eingabe des Formulars und die Identität im Slashed-Format. Auf Grundlage des Namens des ersten Attributs wird ermittelt, um welches Formularfeld es sich handelt und passend dazu mit einer SQL-Query die möglichen Antworten ermittelt. Da der Autocompleter eine HTML-Liste erwartet, wird ausnahmsweise diese und keine XML-Datei zurückgeschickt. Würde man einen eigenen Autocompleter implementieren, könnte man auch hier auf XML-Daten setzen.

## 6.5 Probleme mit der aktuellen VOMSSaml-Schnittstelle

Es hat sich herausgestellt, dass die VOMSSaml-Schnittstelle mehrere gravierende Fehler aufweist. Aus diesem Grund dient der SAML Requester nur der Demonstration der Möglichkeiten. Die durch ihn abgerufenen Assertions entsprechen zum Teil nicht den gewünschten Eigenschaften. Die größten Mängel werden im Folgenden dargestellt.
Zunächst gibt es keine dokumentierte Möglichkeit spezifische generische Attribute abzurufen. Die Assertions enthalten immer alle generischen Attribute eines Nutzers. Versucht man ein Attribut mit dem Namen auszuwählen, welcher in der Assertion angegeben wird, liefert VOMSSaml einen Fehler zurück. Die VOMS-AJAX-GUI ignoriert deshalb sämtliche Angaben zu den gewünschten generischen Attributen und fügt diese nicht der Query hinzu.

Ein weiteres Problem ist, dass die Namen der Attribute die für die Anfrage verwendet werden müssen, nicht den Namen in den Antworten entsprechen. So muss für die Anfrage einer spezifischen Gruppe oder einer Rolle der Name „http://voms.forge.cnaf.infn.it/fqan“ verwendet werden, in der Antwort heißt das Attribut aber „http://authz-interop.org/xacml/subject/voms-fqan“. Laut einem Bericht über die Funktionsweise der ersten Version der Implementierung [Hen09] waren die Antworten in der letzten Version von gLite noch korrekt. Aus dem gleichen Dokument stammen letztlich auch die verwendeten Namen für die Anfrage. Möglicherweise hängt dieser Fehler mit der Entwicklung eines SAML Profils für VOMS zusammen.
Werden keine expliziten Angaben zu den gewünschten Rollen und Gruppen getätigt, liefert VOMSSaml sämtliche Gruppen, aber keine Rollen. Bei einer expliziten Anfrage nach einer Gruppe oder Rolle sollte nur diese und alle übergeordneten Gruppen in der Assertion enthalten sein. Doch auch dies funktioniert nicht. So bestätigt VOMSSaml zwar eine Rolle nur dann, wenn diese explizit angefragt wird, doch die Gruppen werden immer alle aufgezählt - auch dann wenn nur eine bestimmte gewünscht ist.
Fragt man übrigens eine Gruppe oder Rolle an, die für diesen Nutzer nicht gesetzt ist, wird dies ignoriert. Eigentlich wäre es sinnvoll in so einem Fall eine Meldung, dass die gewünschte Assertion nicht ausgestellt werden kann, zu erhalten.
Es zeigt sich also, dass die SAML-Schnittstelle einer größeren Überarbeitung und Fehlerkorrektur bedarf, bevor diese eingesetzt werden kann. Eventuell werden dabei getätigte Anpassungen, z.B. an den Namen, auch Änderungen im SAML Requester erfordern.

# 7 Vorschläge für eine produktive Implementierung

Für den Fall, dass die Ideen und Konzepte dieser Arbeit in eine produktive Implementierung einfließen sollten, werden hier einige Vorschläge gemacht, die in der VOMS-AJAX-GUI nicht berücksichtigt werden konnten.

## 7.1 Alternative JavaScript-Frameworks

Für die VOMS-AJAX-GUI wurde Prototype mit dem Addon Scriptaculous als JavaScript-Framework verwendet. Zwar ist dieses Framework durchaus geeignet, doch sollte der Einsatz alternativer Frameworks dennoch geprüft werden. Im Besonderen bietet sich hier jQuery (jquery.com) aus mehreren Gründen an. Es handelt sich hierbei um ein von John Resig entwickeltes und 2006 vorgestelltes Framework, welches unter anderem viel kleiner als Prototype ist (24KB). Diverse GUI-Elemente, wie eine Autovervollständigung, stehen über die Erweiterung jQuery UI zur Verfügung. Außerdem gibt es eine große Datenbank mit Plugins von Drittentwicklern.
Ein wichtiges Argument für jQuery ist, dass dies das von VOMS Admin verwendete Framework ist. Man kann zwar die VOMS-AJAX-GUI zu einem produktiven System weiterentwickeln, doch erscheint dieses wenig sinnvoll, da VOMS Admin trotz einiger Fehler ein durchaus ausgereiftes System ist und es sich somit anbietet die neuen Konzepte und Bedienoberflächen in die Weiterentwicklung von VOMS Admin einfließen zu lassen. In so einem Fall sollte natürlich auch auf das bereits in VOMS Admin verwendete JavaScript-Framework gesetzt werden.
Der wohl größte Vorteil von jQuery liegt aber darin, dass die jQuery-Funktionen zur Selektion und Traversierung auch auf XML-Dokumente angewendet werden können, die durch einen Ajax-Request abgerufen wurden. In Prototype ließen sich z.B. Selektoren nur auf Elemente des Dokumentenbaumes anwenden, für die Traversierung des XML-Baumes mussten native JavaScript-Funktionen genutzt werden. Wenn man sich die Verarbeitung der XML-Dokumente in der VOMS-AJAX-GUI und zum Vergleich das folgende Beispiel ansieht, zeigt sich wie sehr jQuery in dieser Hinsicht die Implementierung vereinfachen kann.

```
$.ajax({
  url: 'groups_ajax.jsp?subjectsWithGroups',
  dataType: 'xml',
  success: function(transport) {
    $(transport).find("subject[institution='TestVO']").each( function() {
      alert($(this).attr("name"));
    });
  }
});
```

In diesem Code-Fragment wird ein Ajax-Request getätigt, der ein XML-Element mit allen Mitgliedern und deren Gruppen und Rollen abruft. In diesem Dokument werden alle Personen selektiert, die der Institution „TestVO“ angehören und deren Namen nacheinander ausgegeben.

## 7.2 Verbessertes Tabellenlayout mit CSS3

Betrachtet man die Tabelle der Gruppen- und Rollenverwaltung, stellt man fest, dass diese eine sehr starke horizontale Ausbreitung hat. Bedingt wird dies durch die teilweise sehr langen Gruppennamen, was zu der Gefahr führt, dass bei einer etwas umfangreicheren VO mit vielen Untergruppen die Übersichtlichkeit verloren geht, weil der Nutzer zum horizontalen Scrollen gezwungen wird. Zwar wird der Effekt etwas dadurch abgemildert, dass die Gruppennamen an den Slashes von einigen Browsern automatisch umgebrochen werden sobald 100% der Fensterbreite ausgenutzt sind, doch andererseits würden die Kreuze selbst sehr wenig Platz in Anspruch nehmen. Man könnte das Tabellenlayout optimieren indem man die Gruppen- und Rollennamen um 90° dreht oder schräg hinschreibt, wie man dies aus gedruckter Literatur gewohnt ist. Diese Form der Darstellung würde nicht nur Platz sparen, sondern wäre Dank gleich breiter Spalten zudem auch noch übersichtlicher. Mit den derzeitigen Mitteln, die einem mit HTML- und CSS zur Verfügung stehen ist dies jedoch kaum oder nur sehr aufwendig zu realisieren. Wie eine solche kompaktere Form der Darstellung aussehen könnte, zeigt Abbildung 7.1.

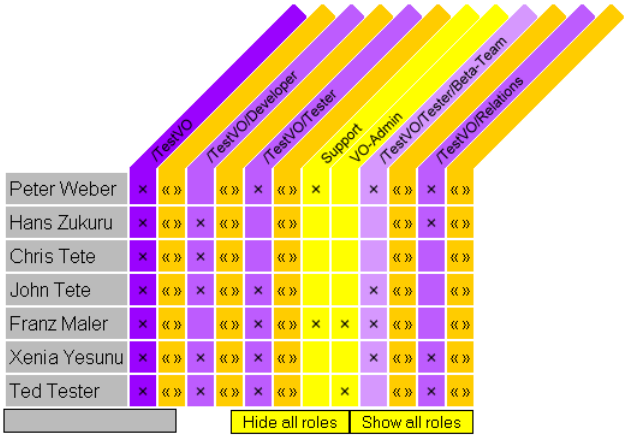



Abbildung 7.1: Optimiertes kompakteres Tabellen-Layout, erstellt mit einem Grafikprogramm

Möglicherweise stehen aber zum Zeitpunkt einer produktiven Implementierung bessere Mittel zur Verfügung, um ein solches Layout zu realisieren. Sehr aussichtsreich in der Hinsicht ist CSS3, welches zur Zeit noch in der Entwurfsphase ist. Nur einige Funktionen werden von wenigen Browsern testweise zur Verfügung gestellt. Betrachtet man den Entwurf „CSS 2D Transforms Module Level 3“ [CTR09] findet man dort eine Möglichkeit HTML-Elemente zu rotieren. In Mozilla-Browsern läßt sich so etwas über die zu Testzwecken eingeführte Eigenschaft `-moz-transform` realisieren - allerdings mit optisch unschönem Ergebnis. Eine

andere CSS3-Eigenschaft, die allerdings aus den neusten Entwürfen wieder entfernt wurde, ist der `writing-mode`, welche einen um 90° gedrehten Text ermöglicht [CTE01]. Dieser wird interessanter Weise schon seit Version 5.5 vom Internet Explorer unterstützt, findet sich aber zur Zeit noch in keinem anderen Browser.
Sobald solche CSS3-Techniken endgültig standardisiert und in den Browsern weiter verbreitet sind, sobald z.B. die Rotation in den meisten Browsern in zufriedenstellender Qualität verfügbar ist, könnte man diese nutzen, um das gewünschte kompakte Layout zu erstellen.

# 8 Fazit

In dieser Arbeit wurde an Hand der VOMS-AJAX-GUI dargestellt, dass ein verstärkter Einsatz von Ajax im Umfeld von Grid-Computing, hier im speziellen in Kombination mit einem VOMS-Server, durchaus gewinnbringend sein kann. Einerseits wurde gezeigt, wie man durch Ajax und eine neu gestaltete Oberfläche die Verwaltung einer VO übersichtlicher gestalten kann. Ohne Ajax wären zwar Tabellen, wie sie für die Gruppen-, Rollen- und Attributverwaltung genutzt werden, auch möglich, doch wäre die Benutzung dieser durch das ständige Neuladen bei jeder Änderung nicht wirklich benutzerfreundlich. Andererseits wurde gezeigt, wie man sich mit Hilfe einer ajax-basierten Oberfläche SAML Assertions nach Wunsch zusammenstellen kann und diese dann auch noch mittels Ajax abrufen kann. Es wurde ebenfalls aufgezeigt wie man dies für den Login auf einen Grid-Dienst nutzen kann.
Die Architektur der entworfenen VOMS-AJAX-GUI wurde gezielt daraufhin ausgelegt, dass Ajax-Requests einen elementaren Bestandteil des Systems bilden und somit oft zum Einsatz kommen. Natürlich kann man bei einer produktiven Implementierung dieses Konzept abschwächen. Die Autovervollständigung des SAML Assertion Requesters ließe sich z.B. genauso gut mittels vorab übertragenen Listen der möglichen Werte realisieren.
Auch stellt sich die Frage, ob es sinnvoll ist die VOMS-AJAX-GUI als eigenständiges System weiterzuentwickeln oder doch eher die Ideen in VOMS Admin einfließen zu lassen. Da die VOMS-AJAX-GUI sehr rudimentär ist und z.B. noch keinerlei Form von Rechtemanagement beinhaltet, dieses in VOMS Admin dagegen sehr weit ausgereift ist, bietet es sich wohl eher an VOMS Admin entsprechend anzupassen. Denkbar wäre es aber auch eine GUI zu entwerfen, die nach einem ähnlichen Prinzip arbeitet, wie die hier vorgestellte, aber als Backend die SOAP-Schnittstellen von VOMS Admin nutzt. Würde man sich doch für eine komplette Neuimplementierung entscheiden, so müsste die SAML-Schnittstelle des VOMS Admin aus diesem herausgelöst werden - doch diese bedarf ohnehin einer Überarbeitung.

# 9 Anhang

## 9.1 Virtuelle Maschine

Auf dem Datenträger zu dieser Arbeit befindet sich neben der VOMS-AJAX-GUI auch ein gzip-komprimiertes Image einer Festplatte für eine virtuelle Maschine der Virtualisierungslösung VirtualBox (www.virtualbox.org). Entpackt man dieses Image und läßt es unter einer standardmäßigen VM laufen, erhält man ein fertig eingerichtetes System, welches nach der Anleitung in dieser Arbeit installiert wurde. Nach dem Booten der VM muss man sich als root mit dem Passwort „rootpass“ einloggen und zunächst alle Dienste starten.

```
/etc/init.d/mysql start
/etc/init.d/tomcat start
/opt/glite/etc/init.d/voms start
```

Zu Testzwecken ist eine CA namens „TestCA“ (Passwort: „TestCApass“) eingerichtet, welche mittels *tinyca2* verwaltet werden kann. Sämtliche bereits erstellten Zertifikate sind einige Jahrzehnte gültig. Zertifikate für den Nutzer „Ted Tester“ findet man im root-Verzeichnis. Mit diesem Zertifikat (Passwort: „testerpass“) hat man sämtliche Rechte im VOMS Admin, im Firefox ist dieses Zertifikat bereits hinzugefügt. In der VOMS-Datenbank ist eine VO namens „TestVO“ mit mehreren Nutzern eingerichtet. Der VOMS-Datenbanknutzer heißt „voms“ (Passwort: „vomspass“).

## 9.2 Troubleshooting

Im folgenden werden Lösungen für einige gängige Fehler aufgezeigt.

**Beim Abruf der VOMS-AJAX-GUI erscheinen die Tabellen nicht.**
Dieses Problem kann verschiedene Ursachen haben. Ein möglicher Grund kann die Verwendung eines nicht unterstützten Browsers sein. In Firefox, Chrome und Safari sollte es keine Probleme geben, aktuelle Versionen des Internet Explorer funktionieren nicht.
Eine weitere Ursache für eine solche Fehlfunktion, die dann vorliegt, wenn zusätzlich der SAML Requester einen Fehler ausgibt, können falsche MySQL-Zugangsdaten sein.
Wurden am Quelltext der VOSM-AJAX-GUI Änderungen durchgeführt, so kann es auch sein, dass sich an den Anfang der XML-Daten eine Leerzeile eingeschlichen hat, welche eine Verarbeitung verhindert.

**Kein Zugriff auf die Dienste von einem anderen Rechner aus.**
Wurde bei der Installation von Scientific Linux die Firewall nicht deaktiviert, sind iptables-Regeln aktiv die den Zugriff verhindern. Mit folgendem Kommando kann man dieses Problem lösen:

```
iptables -D INPUT 1
```

# Abbildungsverzeichnis


2.1 Ajax-Aufruf . . . . . 5

5.1 Gruppen- und Rollen-Verwaltung mit ausgeblendeten Rollen . . . . . 14
5.2 Gruppen- und Rollen-Verwaltung mit eingeblendeten Rollen einer Gruppe, Tooltip mit Mitgliederinformationen und aktiviertem Personenfilter . . . . . 14
5.3 Attribut-Verwaltung . . . . . 15
5.4 Attribut-Verwaltung mit aktiviertem InPlace-Editor und Personen-Filter . . . 15
5.5 Gruppen- und Rollen-Verwaltung eines einzelnen Benutzers in VOMS Admin 16
5.6 Ajax-Kommunikation zwischen Client und Server . . . . . 17

6.1 SAML Assertion Requester beim Abruf . . . . . 24
6.2 SAML Assertion Requester mit diversen bereits getätigten Auswahlen und aktivierter Autovervollständigung . . . . . 25
6.3 Anzeige der Assertion und des Gültigkeitszeitraums in lokaler Zeit . . . . . 26
6.4 Ajax-Kommunikation zwischen Client und VOMS-AJAX-GUI sowie VOMS-Saml auf Serverseite . . . . . 27
6.5 Ablauf des Logins bei einem Grid-Dienst unter Nutzung des SAML Assertion Requesters . . . . . 28

7.1 Optimiertes kompakteres Tabellen-Layout, erstellt mit einem Grafikprogramm 34

# Literaturverzeichnis


[CTE01] *CSS3 module: text*, Mai 2001. http://www.w3.org/TR/2001/WD-css3-text-20010517/.

[CTR09] *CSS 2D Transforms Module Level 3*, Dezember 2009. http://www.w3.org/TR/2009/WD-css3-2d-transforms-20091201/.

[FKT01] Foster, Ian, Carl Kesselman und Steven Tuecke: *The Anatomy of the Grid*. International Journal of Supercomputer Applications, 15(3), 2001. http://www.globus.org/alliance/publications/papers/anatomy.pdf.

[Hen09] Henne, Benjamin: *Test des VOMS-SAML-Services des DGrid VOMS*, Juni 2009. http://dgi.d-grid.de/fileadmin/user_upload/documents/DGI2-FG3/FG3-2/DGI-2_FG-3.2_Test_VOMS-SAML-Service.pdf.

[Jäg08] Jäger, Kai: *Ajax in der Praxis*. Springer.verlag, Berlin Heidelberg, 2008.

[KK07a] Kain, Michael und Guido Keller: *SAML 2.0, ein Tutorium - Teil 1: Theorie*. JAVASpektrum, 05, 2007. http://www.acando.de/Global/GER/fachartikel_ger/kain_keller_JS_05_07.pdf.

[KK07b] Kain, Michael und Guido Keller: *SAML 2.0, ein Tutorium - Teil 2: Praxis*. JAVASpektrum, 06, 2007. http://www.acando.de/Global/GER/fachartikel_ger/kain_keller_JS_06_07.pdf.

[ML07] Mintert, Stefan und Christoph Leisegang: *Ajax : Grundlagen, Frameworks und Praxislösungen*. dpunkt.verlag, Heidelberg, 2007.

[SAM05] *Assertions and Protocols for the OASIS Security Assertion Markup Language (SAML) V2.0*, März 2005. http://docs.oasis-open.org/security/saml/v2.0/saml-core-2.0-os.pdf.

[VA08] *VOMS Admin User's Guide*, April 2008. https://edms.cern.ch/file/974094/1/voms-admin-user-guide.pdf.