Master Thesis

im Rahmen des
Universitätslehrganges „Geographical Information Science & Systems“
(UNIGIS MSc) am Zentrum für GeoInformatik (Z_GIS)
der Paris Lodron-Universität Salzburg

zum Thema

„GeoXACML im J2EE Environment“

vorgelegt von

DI Christian Müller

u1422, UNIGIS MSc Jahrgang 2008

Zur Erlangung des Grades
„Master of Science (Geographical Information Science & Systems) – MSc(GIS)”

Gutachter:
Ao. Univ. Prof. Dr. Josef Strobl

Wien, 01.09.2009

Danksagung

Bedanken möchte ich mich hier in erster Linie bei meiner Frau Henriette für ihre mentale Unterstützung und für die vielen entbehrungsreichen Wochenenden und Feiertage.

Des weiteren bei meinen zwei Söhnen, welche mich immer wieder motiviert haben obwohl sie es gar nicht so genau gewusst haben.

Auch die Mitarbeit an den Open Source Projekten GeoTools (www.geotools.org) und GeoServer (www.geoserver.org) war wichtig um die in dieser Arbeit dargestellten Fakten auch im Einsatz zu testen.

Auch nicht zu vergessen ist, dass die praktische Arbeit im Rahmen des GSOC (Google Summer of Code) 2009 durchgeführt wurde.

Namentlich möchte ich hier Andrea Aime erwähnen, der als mein Mentor im GSOC viele Ideen eingebracht und viele Diskussionen mit mir geführt hat.

Schlussendlich ist auch noch mein größter Kunde, die Statistik Austria zu erwähnen, dessen Anforderungen mich überhaupt zu einem UNIGIS Studium veranlasst haben.

Eigenständigkeistserklärung

"Ich versichere, diese Master Thesis ohne fremde Hilfe und ohne Verwendung anderer als der angeführten Quellen angefertigt zu haben, und dass die Arbeit in gleicher oder ähnlicher Form noch keiner anderen Prüfungsbehörde vorgelegen hat. Alle Ausführungen der Arbeit die wörtlich oder sinngemäß übernommen wurden sind entsprechend gekennzeichnet."

Ort und Datum eigenhändige Unterschrift

Zusammenfassung

Diese Arbeit erläutert Aspekte der geographischen Zugriffskontrolle. Basierend auf XACML, einem von OASIS genormten Standard, wurde seitens OGC eine Erweiterung namens GeoXACML für die Zusatzanforderungen geographischer Daten spezifiziert.

In den ersten Kapiteln wird in die Funktionsweise von XACML eingeführt. Da XACML sehr umfangreich ist, werden nur die für diese Arbeit wichtigen Konzepte und Konstrukte besprochen.

Zugriffskontrolle basierend auf einem Rollenkonzept ist ein weiterer Bestandteil. Auch die Integration in einen J2EE Container wird erläutert.

Danach werden die Eigenschaften und Möglichkeiten von GeoXACML besprochen, der Datentyp Geometrie wird eingeführt und die Zusatzfunktionalität wird aufgelistet.

Alle als XML Dokumente bzw. Fragmente dargestellten Beispiele wurden im Rahmen eines Open Source Projekts (www.geotools.org, www.geoserver.org) erarbeitet. Diese Projekt läuft in einem J2EE Container, erwähnenswerte Erfahrungen sind in die Arbeit eingeflossen.

Eine Fallstudie mit einem WMS GetMap Request zeigt die Problemstellungen und mögliche Lösungen. Besonders das XACML Element <Obligation> eröffnet ganz neue Möglichkeiten.

Zusätzlich werden weitere Möglichkeiten durch die Verwendung von CQL und OGC kodierten Filtern dargestellt.

Auch der Zugriffskontrolle auf Basis von URLs ist ein Kapitel gewidmet.

Abgeschlossen wird die Arbeit mit einer kurzen Abgrenzung zu GeoDRM, welches noch kein Standard ist.

Abstract

This work is about access control in geographic information systems. Based on XACML, specified by OASIS, the OGC consortium extended this XML dialect for the needs of spatial data. This extension is called GeoXACML.

The first chapters are an introduction to the concepts of XACML. Since XACML is a rather complex standard, this work explains only the constructs needed for later chapters, omitting some details.

Another discussion point is access control based on roles and how an integration with a J2EE Container could be established.

Next is the introduction to GeoXACML, its new data type geometry and the additional spatial functions.

All XML examples (complete documents or fragments) have been developed during an open source project (www.geotools.org, www.geoserver.org). This project runs in a J2EE container, the results of this work are described in this paper.

A case study using a WMS GetMap request is used to show problems and solutions. Especially the XAML element <Obligation> offers new and powerful possibilities.

Additional possibilities by using CQL and OGC filters are demonstrated.

URL based access control has it's own chapter.

Finally, there is a chapter pointing out the difference between GeoXACML and GeoDRM.

Inhaltsverzeichnis

1 . Abbildungsverzeichnis

Abbildungsverzeichnis

2 . Tabellenverzeichnis

Tabellenverzeichnis

3 . Abkürzungsverzeichnis

BBOX	Bounding Box
CQL	Catalog Query Language
DRM	Digital Rights Management
EJB	Enterprise Java Beans
EPSG	European Petroleum Survey Group
GeoDRM	DRM für geographische Daten.
GeoXACML	XACML für geographische Daten
GDI	Geodateninfrastruktur
GIS	Geografisches Informationssystem
GML	Geographic Markup Language
HTTP	Hyper Text Transfer Protocol
INSPIRE	Infrastructure for Spatial Information in Europe
IT	Informationstechnologie
JACC	Java Authorization Contract for Containers
J2EE	Java 2 Enterprise Environment
OASIS	Organisation for the Advancement of Structured Information Standards
OGC	Open Geospatial Consortium
OS	Open Source
OWS	OpenGIS Web Services
PAP	Policy Administration Point
PDP	Policy Decision Point
PIP	Policy Information Point
PEP	Policy Enforcement Point
PPS	Permission Policy Set
RPS	Role Policy Set
SAML	Security Assertion Markup Language
SOAP	Simple Object Access Protocol
SQL	Structured Query Language
URI	Uniform Resource Identifier

3 . Abkürzungsverzeichnis

URL	Uniform Resource Locator
URN	Uniform Resource Name
WCS	Web Coverage Service
WKB	Well Known Binary (Kodierung einer Geometrie)
WKT	Well Known Text (Textdarstellung einer Geometrie)
WMS	Web Map Service
WFS	Web Feature Service
XACML	Extensible Access Control Markup Language
XML	Extensible Markup Language

4 . Notation

XML Elemente werden immer in Spitzklammern dargestellt:

Bsp: <xmlElement>

XML Attribute immer Kursiv:

Bsp: *attributeId*

Werte im XML Dokument/Fragment immer unter Hochkomma

Bsp: „Alice“

Begriffsdefinitionen werden unterstrichen

Bsp: Authentifizierung

Security Rollen sind immer in Großbuchstaben und beginnen mit ROLE_

Bsp: ROLE_ANONYMOUS

Auswertungen eines Prädikates immer TRUE oder FALSE

Ergebnis einer XACML Evaluierung immer eine von den 4 Möglichkeiten PERMIT,DENY,NOT_APPLICABLE und INDETERMINATE

XML Dokumente und Fragmente werden durch Farbgebung übersichtlicher gestaltet.

Bsp:

```
<Attribute
 AttributeId="urn:oasis:names:tc:xacml:1.0:subject:subject-id"
 DataType="http://www.w3.org/2001/XMLSchema#string">
       <AttributeValue>Alice</AttributeValue>
</Attribute>
```

5 . Einführung

5.1 Ausgangslage

Der Markt für Geodaten gewinnt immer mehr an Bedeutung. Im Europäischen Raum wird dieser Trend noch durch die EU Richtlinie INSPIRE[1] verstärkt.

Dies bedingt den vermehrten Aufbau von GDIs (Geodateninfrastruktur). Da es sich hier um verteilte Applikationen handelt, kommt standardisierten Protokollen und Formaten eine besondere Bedeutung zu.Seitens OGC[2] gibt es hierfür eine Reihe von Spezifikationen für den Datenaustausch von GIS (Geographisches Informationssystem) Daten.

Hervorzuheben sind hier folgende OGC Spezifikationen:

1) GML (Geographic Markup Language)
   Beschreibt ein XML Format für Vektordaten inklusive Raumbezug
2) WMS (Web Map Service)
   Beschreibt ein Service mit dem Kartenausschnitte eines Map Servers angefordert werden können.
3) WCS (Web Coverage Service)
   Beschreibt ein Service mit dem Rasterdaten angefordert werden können
4) WFS (Web Feature Service)
   Beschreibt ein Service mit dem Vektordaten angefordert werden können, welche in GML dargestellt sind. Auch die Modifikation von Daten ist möglich. Zu diesem Zwecke existiert auch ein Konzept für Transaktionen im GIS Umfeld

Während 1) ein Datenformat beschreibt, definieren 2), 3) und 4) ein Web Service. Diese Services sind, wie der Name schon sagt, für das World Wide Web ausgelegt und bedienen sich des HTTP (Hyper Text Transfer Protocoll). Hierbei werden Parameter als Key/Value Paare übergeben oder im HTTP Body in XML kodierter Form übermittelt. Mittlerweile wird auch schon an einer Möglichkeit gearbeitet, um über SOAP (Simple Object Access Protocol) zu kommunizieren.

Keine dieser Spezifikationen berücksichtigt irgendeine Art der Zugriffskontrolle. (eng. Authorization), welche aber unbedingt notwendig ist.

Für klassische Webservices gibt es aber seitens OASIS[3] sehr wohl Standards. Zum Zwecke der Authentifizierung kann SAML (Security Assertion Markup Language) verwendet werden, zum Zwecke der Zugriffskontrolle ist XACML (Extensible Access Control Markup Language) gedacht.

Aufbauend auf XACML wurde seitens OGC eine Spezifikation mit der Bezeichnung GeoXACML erstellt, welche XACML für die Anforderungen von GIS Daten erweitert.

Um diesen Standard geht es nun in dieser Arbeit, untersucht werden soll wie mit GeoXACML Zugriffskontrolle umzusetzen ist. Zur Evaluierung der Erkenntnisse wurde ein OS (Open Source) Java Projekt gestartet, welches in einem J2EE Container einsetzbar ist.

5.2 Motivation

IT Security war immer schon ein sehr interessantes Thema. Wie eigentlich überall im EDV Bereich stellen sich auch hier komplexe Anforderungen. Security speziell im GIS Bereich war allerdings eine neue „Herausforderung“.

Abgesehen von meinem Interesse hat dieser Know How Aufbau auch einen Nutzen für meine Kunden (EU INSPIRE Richtlinie).

Des weiteren wollte ich auch etwas zu einem Open Source Projekt beitragen, da ich jetzt schon sehr lange fast ausschließlich mit OS Softwarekomponenten arbeite und es für mich moralisch an der Zeit ist, auch etwas „zurückzugeben“

Weiters denke ich, dass auch für das Privatleben GIS und Security eine immer größere Bedeutung gewinnt, man denke hier nur z. Bsp. an die steigende Zahl von Überwachungskameras und an den stetig wachsenden Zugang zu Satellitenaufnahmen und Kartenmaterial. Hier muss man sich natürlich schon auch überlegen, wie man Missbrauch unterbindet.

6 . XACML

6.1 Einleitendes Szenario

Extensible Access Controll Markup Language (XACML) ist eine XML Spezifikation für das Problem der Zugriffskontrolle. (eng. Access). Hier wird ein XML Standard vorgegeben, wie die Regeln der Zugriffskontrolle zu spezifizieren sind, welche Komponenten benötigt werden und welche Funktionalität erfüllt werden muss.

Anhand eines einfachen Beispiels wird nun schrittweise das Konzept hinter XACML erörtert. Ausgangspunkt ist ein WMS Request, welcher ein Bild mit Straßen aus einem Teil Österreichs erzeugt.

```
http://<hostname>/<deploy_name>?
SERVICE=WMS&VERSION=1.1.1&REQUEST=GetMap&LAYERS=Straßen&STYLES=&SRS
=EPSG:31287&BBOX=400000,400000,420000,410000&WIDTH=800&HEIGHT=400&FORM
AT=image/png
```

Parameter:

- <hostname> Der Name des Servers
- <deployname> Context Root des WMS Services
- SERVICE Name des Services
- VERSION WMS Versionsnummer
- REQUEST Art des Requests
- LAYERS Welche GIS Layer werden angefordert
- STYLES Layout
- BBOX Kartenausschnitt
- WIDTH Pixelbreite
- HEIGHT Pixelhöhe
- FORMAT Datenformat des Ergebnisses

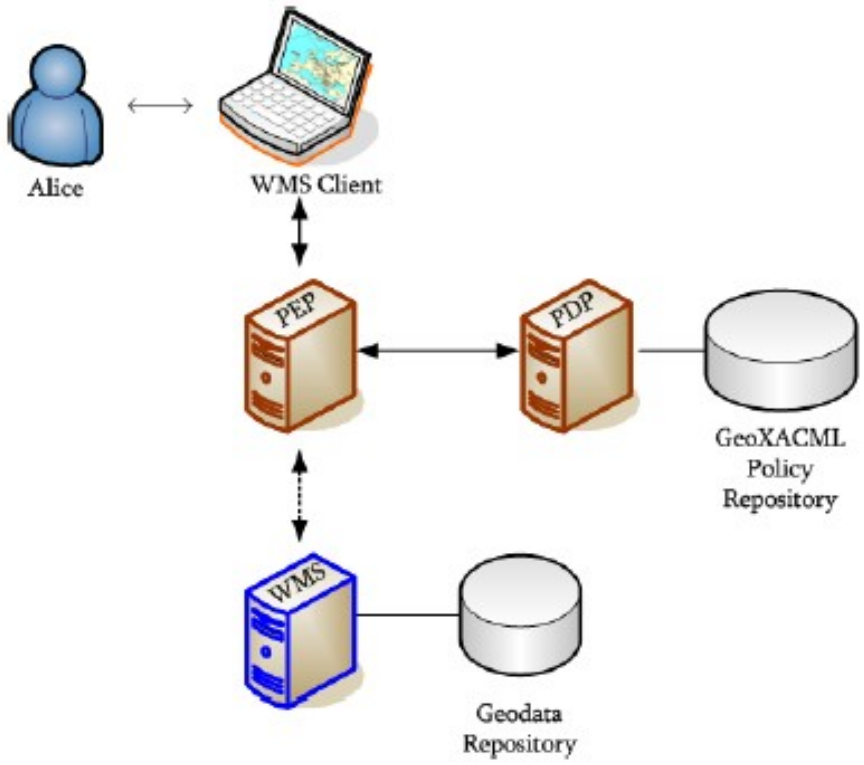
Dieser Request wird nun einer Zugriffskontrolle unterworfen, wobei diese sinnvollerweise vor dem Abarbeiten des eigentlichen Services durchgeführt wird.

Es gibt aber auch Szenarien wo die Zugriffskontrolle erst nach der Beendigung des angeforderten Services durchgeführt wird. Als Beispiel kann ein GetMap Request genommen werden, bei dem das Ergebnis (das Bild) mittels Clipping (Beschneiden) auf eine erlaubte Region eingeschränkt wird.

Auch parallele Verfahren sind denkbar, bei dem Zugriffskontrolle und Service gleichzeitig ausgeführt werden und erst zum Schluss das Ergebnis aufgrund der Berechtigungen modifiziert wird.

Die grundlegende Architektur von XACML ist in folgender Abbildung sehr vereinfacht dargestellt.

Abbildung 1: Architektur XACML



(entnommen aus der GeoXACML Spezifikation)

Hier passiert die Zugriffskontrolle vor dem Aufruf des eigentlichen Services.

Alice sendet einen WMS Request. Vor dem eigentlichen WMS Server ist ein PEP (Policy Enforcement Point). Dieser erstellt aus dem WMS Request einen passenden XACML Request und sendet diesen an den PDP (Policy Decision Point).

Der PDP verfügt über ein Repository von XACML Policies. Er versucht nun eine passende Policy zu finden und die Zugriffsberechtigung zu überprüfen. Das Ergebnis wird zum PEP in Form einer XACML Response übermittelt.

Ist der Zugriff erlaubt (PERMIT), dann wird der WMS Request zum WMS Service weitergeleitet und von diesem beantwortet. Alice selbst merkt von der Zugriffskontrolle nichts.

Wird der Zugriff aber verweigert(DENY), so formuliert der PEP eine entsprechende abweisende WMS Antwort.

XACML definiert folgende Komponenten:

PDP (Policy Decision Point)

Erhält einen XACML Request und evaluiert diesen gegen seine Policies im Repository. Das Ergebnis wird in eine XACML Response verpackt und dem PEP zurückgegeben. Diese Komponente arbeitet ausschließlich mit standardisierten XACML Formaten und ist damit universell einsetzbar.

PEP (Policy Enforcement Point)

Übernimmt den ursprünglichen (hier WMS) Request und formuliert den XACML Request. Dieser wird zum PDP geschickt und auf die Antwort gewartet. Aufgrund dieser lässt der PEP den Zugriff zu oder nicht. PEPs sind nicht universell einsetzbar, diese müssen für diverse Systeme extra entwickelt werden.

PAP (Policy Administration Point)

Komponente zum administrieren der Policies. Universell einsetzbar, da hier nur mit XAMCL Policies gearbeitet wird. Die Art der Speicherung des Policy Repositories ist nicht vorgegeben, File basierend ist genauso möglich wie speichern in einer SQL DB oder auf einem HTTP Server.

PIP (Policy Information Point)

Ein Policy Information Point dient dem PEP dazu, Umgebungsinformation abzufragen. Klassische Beispiele sind hier Datum und Uhrzeit welche von einem Internet Time Server geholt werden. Genauso kann ein J2EE Web Container als PIP gelten, man kann hier z. Bsp. die IP Adresse des Clients abfragen.

Der Kommunikationsmechanismus zwischen den Komponenten ist nicht vorgegeben. „In Memory“ ist sicher das schnellste, genauso kann man aber die Komponente über HTTP oder eine andere Transporttechnologie kommunizieren lassen. Das ist leicht möglich, da man ja alle benötigten Elemente (XACML Request,Response und Policy) in XML serialisieren kann.

6.2 Kommunikation und Verhalten von PEP und PDP

Ein PDP hat folgende Möglichkeiten zur Beantwortung eines XACML Requests

1. PERMIT
   Zugriffsberechtigung vorhandenen

2. DENY
   Zugriff verweigert

3. INDETERMINATE
   Bei der Verarbeitung ist ein Fehler aufgetreten, Ergebnis unbestimmt

4. NOT APPLICABLE
   Es wurde keine XACML Policy gefunden, mit der der Request evaluiert werden kann

Weiters unterscheidet man 3 Typen von PEPs, welche in Abhängigkeit obiger Antworten verschieden reagieren.

1. BASE PEP
   PDP Antwort PERMIT, dann Zugriff erlaubt
   PDP Antwort DENY, dann Zugriff verboten
   Für alle anderen Antworten ist das Verhalten nicht vorgegeben

2. DENY biased PEP
   PDP Antwort PERMIT, dann Zugriff erlaubt
   Alle anderen PDP Antworten verhindern den Zugriff.

3. PERMIT biased PEP
   PDP Antwort DENY, dann Zugriff verboten
   Alle anderen Antworten erlauben den Zugriff.

6.3 XACML Request

Der PEP erzeugt nun aus dem spezifischen (WMS,WFS,..) Request einen XACML Request, welchen er zum PDP sendet. XACML definiert hier folgende Begriffe:

Subject

Wer will zugreifen ? Eine eindeutige Identifikation (Mensch Computer,...).wird benötigt

Resource

Auf was soll der Zugriff erfolgen.

Action

Art des Zugriffs.

Environment

Umgebungsvariablen welche bei der Entscheidung helfen sollen.

Aus dem beispielhaften WMS Request (siehe Einleitendes Szenario) ergibt sich nun

Resource:	Straßen
Action:	GetMap
Subject:	?
Environment	?

Das Environment kann durchaus leer sein, hier könnte beispielsweise die IP Adresse von Alice eingefügt werden.

Das Subject, in unserem Fall Alice, kann nicht aus dem WMS Request extrahiert werden. Nachfolgend nun einige Begriffsdefinitionen.

Authentisierung

Der Vorgang des Identifizierens eines Subjektes an einem System. Das einfache Login mit Benutzername und Password ist ein solcher Vorgang. Es gibt natürlich weitere Möglichkeiten, Identifikation mittels Biometrie (Iris, Fingerabdruck) oder mittels Besitz und Wissen (Bankomatkarte mit Geheimzahl).

Authentifizierung

Der Vorgang auf der Gegenseite, das System überprüft das sich authentisierende Subjekt. (Prüfung des Passwortes, der Geheimzahl, eines Zertifikates,usw.)

Autorisierung

Autorisierung (Zugriffskontrolle) kann erst nach erfolgreicher Authentifizierung erfolgen. Hier wird überprüft ob das Subjekt die Berechtigung für die von ihm gewünschten Handlungen hat.

Authentifizierung ist **nicht** Bestandteil von XACML. Wir gehen hier davon aus dass wir eine authentifizierte Benutzerkennung zur Verfügung haben. In einem J2EE Container ist das über die API sehr einfach ermittelbar, nämlich

a) im Web Container durch die Methode **getRemoteUser** im **HttpServletContext** Objekt.

b) im EJB (Enterprise Java Beans) Container durch die Methode **getCallerPrincipal** im **EJBContext** Objekt.

Die Benutzerkennung könnte auch im WMS URL als Parameter mitgegeben werden. Natürlich nicht in plain (reinem) Text, sondern z. Bsp. als SAML Artefakt. Ein WMS Request mit SAML Artefakt

http://<hostname>/<deploy_name>?SERVICE=WMS&VERSION=1.1.1&REQUEST=GetMap&LAYERS=Straßen&STYLES=&SRS=EPSG:31287&BBOX=400000,400000,420000,410000&WIDTH=800&HEIGHT=400&FORMAT=image/png&SAMLart=00040000c878f3fd685c833eb03a3b0e1daa329d47338205e436913660e3e917549a59709fd8c91f2120222f

Mit Hilfe des zusätzlichen Parameter *SAMLArt* kann nun bei einem SAML Artifact Provider die Benutzerkennung abgefragt werden.

Wie auch immer, man geht davon aus dass meine eine authentifizierte Benutzerkennung zur Verfügung hat und damit ergibt sich

Subject: „Alice“

Ein XACML <Request> Element besteht nun aus

- Einem oder mehreren <Subject> Elementen
- Einer oder mehreren <Resource> Elementen
- Einem <Action> Element
- Einem <Environment> Element

Die Daten selbst (Alice,GetMap und Straßen) werden als XACML Attribute übergeben.

Ein XACML <Attribute> hat ein oder mehrere Elemente vom Typ. <AttributeValue> (eng. Multivalued Attribute). Der Name (Identifikation) eines Attributes wird festgelegt durch die *AttributeId* und zusätzlich durch einen optionalen *Issuer*(Herausgeber). Auch der Datentyp des Attributes muss festgelegt werden.

Ein Beispiel:

```
<Attribute
 AttributeId="urn:oasis:names:tc:xacml:1.0:subject:subject-id"
 DataType="http://www.w3.org/2001/XMLSchema#string">
      <AttributeValue>Alice</AttributeValue>
</Attribute>
```

Hier handelt es sich um ein <Attribute> mit den Namen (*AttributeId*)

urn:oasis:names:tc:xacml:1.0:subject:subject-id

Der Name selbst ist als URN (Unified Resource Identifier) kodiert. Der Teil „urn:oasis:names:tc:xacml“ ist für XACML reserviert. Der darauf folgende Namensteil bestimmt die Versionsnummer „1.0“ oder „2.0“. „subject“ bedeutet dass dieses Attribut zu der Kategorie „Subject“ gehört und „subject-id“ besagt, dass das nachfolgende <AttributeValue> Element den Namen eines Benutzers beinhaltet.

Der Datentyp ist

http://www.w3.org/2001/XMLSchema#string

Mögliche Datentypen sind in der XACML Spezifikation aufgelistet. Zusätzlich kann man weitere Datentypen hinzufügen (GeoXACML fügt den

Datentyp Geometrie hinzu).

Der Wert des <AttributeValue> Element ist

```
Alice
```

Das optionale *Issuer* Attribut ist nicht angegeben. Dieses komplettiert den Namen eigentlich, es kann also verschiedene Attribute mit gleichem Namen und unterschiedlichem *Issuer* geben. Hier wird der Übersichtlichkeit halber darauf verzichtet.

Der ganze Request nun als vollständiges XML Dokument:

```
<?xml version="1.0" encoding="UTF-8"?>
<Request xmlns="urn:oasis:names:tc:xacml:2.0:context:schema:os"
        xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
        <Subject>
                <Attribute
                  AttributeId="urn:oasis:names:tc:xacml:1.0:subject:subject-id"
                  DataType="http://www.w3.org/2001/XMLSchema#string">
                        <AttributeValue>Alice</AttributeValue>
                </Attribute>
        </Subject>
        <Resource>
                <Attribute
                  AttributeId="urn:oasis:names:tc:xacml:1.0:resource:resource-id"
                  DataType="http://www.w3.org/2001/XMLSchema#string">
                        <AttributeValue>Strassen</AttributeValue>
                </Attribute>
        </Resource>
        <Action>
                <Attribute
                  AttributeId="urn:oasis:names:tc:xacml:1.0:action:action-id"
                  DataType="http://www.w3.org/2001/XMLSchema#string">
                        <AttributeValue>read</AttributeValue>
                </Attribute>
        </Action>
        <Environment/>
</Request>
```

XACML Request und Response verwenden als XML Namensraum (eng. Namespace)

```
urn:oasis:names:tc:xacml:2.0:context:schema:os
```

6.4 XACML Response

Der PDP sendet nach der Evaluierung eines XACML Requests gegen seine Policies eine XACML Response. Dieses <Response> Element besteht aus mehreren <Result> Elementen, nämlich ein <Result> Element für jedes <Resource> Element aus dem XACML Request.

Anmerkung: Für die Verwendung von mehreren Resourcen im Request und den entsprechenden <Result> Elementen gibt es eine eigene Spezifikation[4]

Ein <Result> Element wiederum besteht aus einem <Decision> Element. Optional kann auch noch ein <Status> Element und ein <Obligations> Element vorhanden sein.

Eine Zugriffsberechtigung kann so aussehen.

```
<?xml version="1.0" encoding="UTF-8"?>
<Response
    xmlns="urn:oasis:names:tc:xacml:2.0:context:schema:os"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
  <Result ResourceId="urn:oasis:names:tc:xacml:1.0:resource:resource-id“>
    <Decision>Permit</Decision>
    <Status>
      <StatusCode
          Value="urn:oasis:names:tc:xacml:1.0:status:ok"/>
    </Status>
  </Result>
</Response>
```

Der Wert des Attributes *ResourceId* ist die Verbindung zur *AttributeId* im Resource Teil des Requests. Es handelt sich hier um die Resource „Straßen“. Das <Decision> Element hat den Wert „Permit“, weitere Möglichkeiten sind „Deny“,“Indeterminate“ und „NotApplicable“.

Das <Status> Element gibt zusätzliche Information an. Neben <StatusCode> kann es optional auch noch <StatusDetail> und ein <StatusMessage> Element angeben. Die Interpretation des <Status> Elements ist aber implementationsabhängig und deswegen wird hier nicht länger darauf eingegangen.

Eine Zugriffsberechtigung mit <Obligations> (Verpflichtungen, Verbindlichkeiten) sieht folgendermaßen aus.

```
<?xml version="1.0" encoding="UTF-8"?>
<Response
    xmlns="urn:oasis:names:tc:xacml:2.0:context:schema:os"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
  <Result ResourceId="urn:oasis:names:tc:xacml:1.0:resource:resource-id“>
    <Decision>Permit</Decision>
    <Status>
      <StatusCode
          Value="urn:oasis:names:tc:xacml:1.0:status:ok"/>
    </Status>
    </Obligations>
      <Obligation ObligationId="urn:oasis:names:tc:xacml:1.0:obligation-mail“
                  FullfillOn="Permit“.>
        <AttributeAssignment  AttributeId="urn:oasis:names:tc:xacml:mail“
                              DataType="http://www.w3.org/2001/XMLSchema:string“>
            unigis@sbg.ac.at
        </AttributeAssignment>
      .<Obligation>
    </Obligations>
  </Result>
</Response>
```

In einer Obligation werden zusätzliche Informationen vom PDP zum PEP übertragen. Jede <Policy> (siehe nächstes Kapitel) kann eine Menge von <Obligation> Elemente an den PEP zurückliefern.

Eine Obligation wird eindeutig durch ihr *ObligationId* Attribute identifiziert. Das Attribut *FullFillOn* zeigt die Gültigkeit dieser Verpflichtung an, „Permit“ und „Deny“ sind die Möglichkeiten.

Eine Obligation beinhaltet wiederum eine Menge von <AttributeAssignment> Elementen. In diesen ist dann schließlich die Zusatzinformation enthalten, wiederum identifiziert durch *AttributeId* und *DataType*.

Obiges Beispiel enthält ein <Obligation> Element

```
urn:oasis:names:tc:xacml:1.0:obligation-mail
```

welches im Falle einer „Permit“ Entscheidung zu verarbeiten ist. Diese Verpflichtung enthält wiederum ein <AttributeAssignment> mit

der *AttributeId*

urn:oasis:names:tc:xacml:mail

und dessen Inhalt ist nunmehr eine Mail Adresse.

Was der PEP nun zu tun hat, hängt von der konkreten Implementierung/ Konfiguration ab. In diesem Falle könnte der PEP eine Mail an die angegebene Adresse senden und den Zugriff dadurch protokollieren.

Wenn ein PEP eine Obligation nicht kennt, so muss er das Ergebnis auf INDETERMINATE umwandeln um einen Fehler im System zu signalisieren. Keinesfalls dürfen <Obligation> Elemente ignoriert werden.

Dieses Konstrukt ist für Zugriffskontrolle im GIS Bereich besonders wertvoll. Als einfache Variante kann man sich ein <Obligation> Element mit der *attributeId* „geometryRestriction“ vorstellen, welches ein <AttributeAssignment> Element enthält, dessen Wert eine in GML kodierte Geometrie ist. Der PEP wäre dann verpflichtet ein georeferenziertes Ergebnis (Karte, Vektordaten) mit einer „Intersection“ Operation zu bearbeiten. (Nicht berechtigte Regionen werden weggeschnitten)

Der Nachteil an dieser Vorgangsweise ist natürlich, dass sie in keinster Weise portabel ist. Ein XACML Repository mit <Obligation> Elementen kann nicht einfach mit einem anderen PEP zusammenarbeiten. Allerdings kreiert der PEP auch die XACML Requests welche gegen das XACML Repository geprüft werden müssen. Diese Requests sind auch nicht standardisiert (inhaltlich) und somit ist auch ein XACML Repository ohne <Obligation> Elemente nicht einfach für einen anderen PEP zu verwenden. Daher relativiert sich dieser Nachteil von <Obligation> Elementen als eher vernachlässigbar.

6.5 XACML Policy

6.5.1 Einfache Policies

Der PDP erhält einen XACML Request vom PEP, evaluiert diesen gegen die Policies aus seinem Repository und sendet eine XACML Response. Als einleitendes Beispiel eine Policy die die jedem alles erlaubt (und damit praktisch sinnlos ist, mit Ausnahme für einen Administrator).

```
<?xml version="1.0" encoding="UTF-8"?>
<Policy xmlns="urn:oasis:names:tc:xacml:2.0:policy:schema:os"
        xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
        PolicyId="urn:oasis:names:tc:xacml:2.0:gis:PermitAll"
        RuleCombiningAlgId="urn:oasis:names:tc:xacml:1.0:rule-combining-algorithm:deny-
overrides">
        <Target />
        <Rule Effect="Permit"
                RuleId="urn:oasis:names:tc:xacml:2.0:rule:PermitAll" >
        </Rule>
</Policy>
```

Der XML Namensraum ist nicht derselbe wie für Request und Response. Jede Policy braucht zwingend ein Attribute *PolicyId*. Der Wert dieser muss eindeutig sein, und zwar mindestens innerhalb des Repositories.

Die Kernelemente eines <Policy> Elements sind <Target> und <Rule>.

Das <Target> selbst wiederum besteht aus den Elementen <Subjects>,<Resources>,<Actions> und <Environments>. Die Elementnamen verraten hier schon die Beziehung zum XACML Request welcher die gleichen Elemente enthält. Die Aufgabe des <Target> Elements ist zu beschreiben, für welche XACML Requests diese Policy zuständig ist.

Wichtig ist hier noch, dass nicht alle Policies gegen Requests evaluiert werden. Man unterscheidet **Policies by Request** und **Policies by Reference**. Letztere werden niemals gegen XACML Requests evaluiert, sondern sind nur „Bausteine“ zum kombinieren von Policies. (Dazu später mehr)

Da ein PDP Zugriff auf eine Menge von Policies hat, gelten folgende Regeln für das Ermitteln der entsprechenden Policy.

1. Der PDP findet genau eine passende Policy für den Request, diese Policy wird evaluiert und das Ergebnis an den PEP zurückgesendet. Die entsprechende Entscheidung kann PERMIT oder DENY sein.
2. Der PDP findet keine Policy, dann ist das Ergebnis NOT_APPLICABLE.
3. Der PDP findet mehr als eine Policy, dann ist das Ergebnis INDETERMINATE.

In unserem Beispiel ist das <Target> Element leer, dass bedeutet dass die Policy für jeden XACML Request zuständig ist. Dies impliziert weiterhin, dass es nur genau diese eine Policy by Request im Repository geben darf.

Das <Rule> Element ist in unserem Beispiel ebenfalls leer. Dies bedeutet dass die Rule zu TRUE evaluiert. Das <Rule> Attribute *Effect* kann die Werte „Permit“ und „Deny“ haben. Evaluiert eine Rule zu TRUE, so bestimmt der Wert des „Effect“ Attributes ob eine XMACL PERMIT oder DENY Response geschickt wird.

Ein <Rule> Element hat wiederum zwingend ein Attribut *RuleId*. Der Hintergedanke hierbei ist, dass <Rule> Elemente in verschiedenen Policies verwendet werden können und man sie damit nur einmal erstellen muss. Allerdings müssen die <Rule> Elemente wirklich textuell in die Policy eingefügt werden, sie sind damit keine eigenständige Einheiten die referenziert werden können. Die *RuleId* Attribute sind für den PAP (Policy Administration Point) gedacht, um hier eine Unterstützung beim Erstellen von komplexen Policies zur Verfügung zu stellen.

Das Attribute *RuleCombiningAlgID* des <Policy> Elements wird später erläutert.

Würde man im obigen Beispiel den Werte des Attributes *Effect* von „Permit“ auf „Deny“ verändern, so würde jeder XACML Request mit einer Deny Response beantwortet. (Auch das hat natürlich wenig Sinn).

6.5.2 Das Target Element

Mit dem <Target> Element wird evaluiert, ob eine Policy für den Request zuständig ist. Das einleitendes Beispiel wird nun insofern modifiziert, dass die Policy nur für den Benutzer „Alice“ vom PDP verwendet wird.

```
<?xml version="1.0" encoding="UTF-8"?>
<Policy xmlns="urn:oasis:names:tc:xacml:2.0:policy:schema:os"
        xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
        PolicyId="urn:oasis:names:tc:xacml:2.0:gis:PermitAlice"
        RuleCombiningAlgId=
        "urn:oasis:names:tc:xacml:1.0:rule-combining-algorithm:deny-overrides">
        <Target>
          <Subjects>
            <Subject>
                <SubjectMatch
                MatchId="urn:oasis:names:tc:xacml:1.0:function:string-equal">
                        <AttributeValue
                        DataType="http://www.w3.org/2001/XMLSchema#string">
                                Alice
                        </AttributeValue>
                        <SubjectAttributeDesignator
                                DataType="http://www.w3.org/2001/XMLSchema#string"
                        AttributeId="urn:oasis:names:tc:xacml:1.0:subject:subject-id" />
                </SubjectMatch>
            </Subject>
          </Subjects>
        </Target>
        <Rule Effect="Permit"
                RuleId="urn:oasis:names:tc:xacml:2.0:rule:PermitAll" >
        </Rule>
</Policy>
```

Diese Policy erlaubt Alice den Zugriff auf jede Ressource mit jeder Aktion ohne etwaiger Berücksichtigung des Environments.

Das <Target> Element hat ein optionales Element <Subjects> welches wieder aus mehreren (mindestens eines) <Subject> Elementen bestehen kann. Jedes <Subject> Element hat wiederum 1 bis N <SubjectMatch> Elemente.

Das <SubjectMatch> Element hat ein Attribute *MatchId* welches angibt,

mit welcher XACML Funktion verglichen werden soll. In unserem Beispiel

```
urn:oasis:names:tc:xacml:1.0:function:string-equal
```

Dies entspricht einem normalen Vergleich von Strings. Mögliche Funktionen sind in der XACML 2.0 Spezifikation, Punkt7.5, „Match Evaluation“ angegeben. Eine alternative wäre

```
urn:oasis:names:tc:xacml:1.0:function:string-regexp-match
```

Dies würde bedeuten, dass das erste Argument als Regulärer Ausdruck zu interpretieren ist, in unserem Beispiel macht das mit „Alice“ natürlich keinen Sinn.
Das erste Argument für die Match Funktion wird als <Attribute> angegeben. Für das zweite Argument benötigen wir ein Element <AttributeDesignator>.

Ein AttributeDesignator ist ein Verweis auf ein Attribut, welches im XACML Request steht. Man benötigt ja eine Möglichkeit in der Policy zu spezifizieren, auf welche Attribute im Request zugegriffen werden soll. Wie schon schon beschrieben, hat ein Attribut im Request eine *AttributeId*, einen *Datentyp* und optional einen *Issuer*. Genau diese drei XACML Attribute finden wir in einem AttributeDesignator wieder, nämlich *Datatype*, *AttributeId* und optional *Issuer*.

In unserem Beispiel wird das Element <SubjectAttributeDesignator> verwendet. Dies bedeutet, dass das Attribut nur im <Subjects> Teilbaum des Requests gesucht wird.

Wichtig ist noch, dass ein AttributeDesignator immer zu einem Bag von Attributwerten evaluiert. Ein Bag ist eine Sammlung von Attributwerten, in dem im Gengensatz zu einem Set (Menge) Elemente mehrmals vorkommen dürfen. Dieses Bag Konstrukt ist erforderlich, da es mehr als ein <Attribute> Element mit der selben *AttributeId* im Request geben kann.

Für das <SubjectMatch> Element führt der XACML Prozessor nun folgende Operationen aus. Er sucht alle passenden Attribute im <Subjects> Element des XACML Requests und schaut dann nach , ob einer dieser Attributwerte gleich „Alice“ ist.

Obige Policy ist nun für jeden Request zuständig, welcher von Alice initiiert wird. Möchte man nun weiter einschränken, so benötigt man

für die WMS Aktion „GetMap“ ein <Actions> Element, für die Resource „Strassen“ ein <Resources> Element. Für etwaige Environment Attribute gibt es auch ein <Environments> Element.

Ein <Subjects> Element besteht aus 1..N <Subjects>, diese wiederum aus 1..N <SubjectMatch> und letzteres aus einem Attribute und einem <SubjectAttributeDesignator> Element.

Wichtig ist hier noch zu erwähnen, dass mehrere <SubjectMatch> Elemente mittels logischem UND verknüpft werden , während mehrere <Subject> Elemente ODER verknüpft werden.

Beispiel für ein <Subjects> Element welches für Alice oder Bob zuständig ist

```
<Subjects>
 <Subject>
    <SubjectMatch
    MatchId="urn:oasis:names:tc:xacml:1.0:function:string-equal">
        <AttributeValue
        DataType="http://www.w3.org/2001/XMLSchema#string">
            Alice
        </AttributeValue>
        <SubjectAttributeDesignator
            DataType="http://www.w3.org/2001/XMLSchema#string"
        AttributeId="urn:oasis:names:tc:xacml:1.0:subject:subject-id" />
    </SubjectMatch>
 </Subject>
 <Subject>
    <SubjectMatch
    MatchId="urn:oasis:names:tc:xacml:1.0:function:string-equal">
        <AttributeValue
        DataType="http://www.w3.org/2001/XMLSchema#string">
            Bob
        </AttributeValue>
        <SubjectAttributeDesignator
            DataType="http://www.w3.org/2001/XMLSchema#string"
        AttributeId="urn:oasis:names:tc:xacml:1.0:subject:subject-id" />
    </SubjectMatch>
 </Subject>
</Subjects>
```

Als Gengensatz dazu dasselbe Element, wenn im Request „Alice“ und „Bob“ vorkommen müssen.

```
<Subjects>
 <Subject>
    <SubjectMatch
    MatchId="urn:oasis:names:tc:xacml:1.0:function:string-equal">
        <AttributeValue
        DataType="http://www.w3.org/2001/XMLSchema#string">
            Alice
        </AttributeValue>
        <SubjectAttributeDesignator
            DataType="http://www.w3.org/2001/XMLSchema#string"
        AttributeId="urn:oasis:names:tc:xacml:1.0:subject:subject-id" />
    </SubjectMatch>
    <SubjectMatch
    MatchId="urn:oasis:names:tc:xacml:1.0:function:string-equal">
        <AttributeValue
        DataType="http://www.w3.org/2001/XMLSchema#string">
            Bob
        </AttributeValue>
        <SubjectAttributeDesignator
            DataType="http://www.w3.org/2001/XMLSchema#string"
        AttributeId="urn:oasis:names:tc:xacml:1.0:subject:subject-id" />
    </SubjectMatch>
 </Subject>
</Subjects>
```

Analog dazu sind <Actions>,<Action><ActionMatch> und <ActionAttributeDesignator>.

Für Resourcen dann <Resources>,<Resource><ResourceMatch> und <ResourceAttributeDesignator>.

Für Umgebungsvariablen schlussendlich <Environments>,<Environment>,<EnvironmentMatch> und <EnvironmentAttributeDesignator>

6.5.3 Rules

Eine Policy besteht (vereinfacht, nicht alle Elemente werden hier aufgelistet) aus einem <Target>, 1 oder mehreren <Rule> Elementen und einem optionalen <Obligations> Element, welches im Abschnitt XACML Response bereits erklärt wurde.

Das <Target> Element filtert die passenden XACML Requests. Passt ein XACML Request zum <Target> Element, wird die Policy evaluiert, das heißt, ihre <Rule> Elemente werden ausgewertet. In obigem Beispiel ist ein einfaches <Rule> Element gegeben, dessen *Effect* Attribut den Wert „Permit“ hat.

Eine <Rule> hat zwei optionale Bestandteile.

1) Ein eigenes <Target> Element

   Dieses funktioniert genauso wie im <Policy> Element. Hier kann man nun einschränken, ob diese <Rule> auf den XACML Request angewendet werden soll. Dieses <Target> Element ist also eine Verfeinerung des <Target> Elements aus der Policy. Fehlt es gänzlich, so wird die <Rule> angewendet.

2) Ein <Condition> Element

   Das <Condition> Element evaluiert zu TRUE oder FALSE. Wenn es zu TRUE evaluiert, so ist das Ergebnis des <Rule> Elements der Wert seines *Effect* Attributes. („Permit“ oder „Deny“). Bei FALSE wird das ganze <Rule> Element im Entscheidungsprozess nicht berücksichtigt.

Beispiel für ein <Rule> Element

```
<Rule
  RuleId="urn:oasis:names:tc:xacml:2.0:conformance-test:wildcard:rule"
  Effect="Permit">
 <Condition>
  <Apply FunctionId="urn:oasis:names:tc:xacml:1.0:function:all-of-all">
    <Function FunctionId="org:geotools:function:string-wildcard-match"/>
    <Apply FunctionId="urn:oasis:names:tc:xacml:1.0:function:string-bag">
      <AttributeValue DataType="http://www.w3.org/2001/XMLSchema#string">
          prefix*infix*suffix
      </AttributeValue>
      <AttributeValue DataType="http://www.w3.org/2001/XMLSchema#string">
          prefix?infix?suffix
      </AttributeValue>
    </Apply>
    <SubjectAttributeDesignator
      AttributeId="urn:oasis:names:tc:xacml:2.0:conformance-test:test-attr"
      DataType="http://www.w3.org/2001/XMLSchema#string"/>
  </Apply>
 </Condition>
</Rule>
```

Obiges Beispiel hat nun viele Neuerungen, welcher die Möglichkeiten von XACML verdeutlichen sollen. Eine tiefer greifende Beschreibung des <Condition> Elements würde den Rahmen dieser Arbeit sprengen, der Leser sei hier auf die Spezifikationen in [3] verwiesen.

Dieses XML Fragment ist Bestandteil einer Test Policy, welche zum Überprüfen einer neuen Funktion

```
org:geotools:function:string-wildcard-match
```

eingesetzt wird. XACML Funktionen wurden schon im <Target> Elementbaum eingesetzt, und zwar als Wert des Attributes *MatchId*. Es ist also möglich, einem XACML Prozessor neue Funktionen hinzuzufügen. Dies ist wichtig im Hinblick auf GeoXACML, welches eine Menge von neuen graphischen Funktionen benötigt.

Das Element <Apply> steht für Auswertung oder Anwendung. Das innere <Apply> hat ein Attribut *FunctionId* mit dem Wert

```
urn:oasis:names:tc:xacml:1.0:function:string-bag
```

Damit wird aus den nachfolgenden <AttributeValue> Elementen ein Bag erzeugt. Das äußere <Apply> hat eine *FunctionId*

```
urn:oasis:names:tc:xacml:1.0:function:all-of-all
```

Dies ist ein Beispiel für eine ***High Order Bag Function.***

Diese Funktion verlangt 3 Argumente:

1) Ein <Function> Element.

2) Einen Bag von <AttributeValue> Elementen, hier durch das innere <Apply> Element erzeugt.

3) Einen weiteren Bag von <AttributeValue> Elementen, hier durch das <SubjectAttributeDesignator> Element erzeugt.

Nun wird jeder Wert von 2) mit jedem Wert von 3) mittels der Funktion in 1) ausgewertet, wenn alle Auswertungen TRUE ergeben, ist das Gesamtergebnis auch TRUE, andernfalls FALSE. Das *Effect* Attribute des <Rule> Elements hat den Wert „Permit“, ein TRUE würde also ein PERMIT ergeben.

Eine Policy hat nun möglicherweise mehrere <Rule> Elemente. Im <Policy> Element hatten wir folgendes Attribut.

```
RuleCombiningAlgId=
      "urn:oasis:names:tc:xacml:1.0:rule-combining-algorithm:deny-overrides">
```

Dieses Attribut beschreibt wie die Auswertungen der einzelnen <Rule> Elemente zu einem Gesamtergebnis kombiniert werden.

Aus dem Namen des Rule Combining Algorithmus kann man schon ableiten, dass ein einziges DENY eines <Rule> Elements zum DENY der ganzen Policy führt.

Folgende Kombinationsmöglichkeiten sind vorhanden:

XACML URN
urn:oasis:names:tc:xacml:1.0:rule-combining-algorithm:deny-overrides
urn:oasis:names:tc:xacml:1.0:rule-combining-algorithm:permit-overrides
urn:oasis:names:tc:xacml:1.0:rule-combining-algorithm:first-applicable
urn:oasis:names:tc:xacml:1.0:rule-combining-algorithm:only-one-applicable
urn:oasis:names:tc:xacml:1.1:rule-combining-algorithm:ordered-deny-overrides
urn:oasis:names:tc:xacml:1.1:rule-combining-algorithm:ordered-permit-overrides

Tabelle 1: Rule Combining Algorithmen

Erläuterungen:

deny-overrides:
Evaluiert zu DENY, wenn nur eine einziges <Rule> Element den *Effect* „Deny“ hat.

Permit-overrides:
Evaluiert zu PERMIT, wenn nur eine einziges <Rule> Element den *Effect* „Permit“ hat.

First-applicable:
Evaluiert zu DENY oder PERMIT, die erste zuständige <Rule> in der Policy bestimmt das Ergebnis.

Only-one-applicable:
Evaluiert zu DENY oder PERMIT, es darf aber nur ein einziges <Rule> Element zuständig sein

ordered-deny-overrides:
Wie deny-overrides, die Reihenfolge der Evaluierung muss gleich der Reihenfolge der <Rule> Elemente in der Policy sein.

Ordered-permit-overrides:
Wie permit-overrides, die Reihenfolge der Evaluierung muss gleich der Reihenfolge der <Rule> Elemente in der Policy sein.

6.5.4 PolicySets

Ein PDP welcher die Policy aus vorigem Abschnitt zur Verfügung hat, würde Alice jeden Zugriff erlauben (Antwort PERMIT). Für jeden anderen Benutzer wird keine Policy gefunden und die Antwort wäre „NOT_APPLICABLE“. Möchte man ein DENY wenn der Benutzer nicht Alice ist, so kann man dies erreichen indem man Policies kombiniert.

Hierzu dienen PolicySets. Ein PolicySet besteht aus Policies und wiederum PolicySets. Damit erreicht man eine hierarchische Baumstruktur. Die Blätter des Baums sind immer Policies, alle anderen Knoten immer PolicySets.

Ein PolicySet wird mit dem Tag <PolicySet> erzeugt. Analog zum Attribute *PolicyId*, welches einer Policy einen eindeutigen Namen zuordnet, gibt es für das <PolicySet> Element ein Attribut *PolicySetId*.

Da diese Ids eindeutig sein müssen, kann man nun aus einem PolicySet andere Policies/PolicySets referenzieren. Durch diesen Mechanismus wird Wiederverwendbarkeit sichergestellt und Redundanz vermieden.

Die entsprechenden Tags sind <PolicyIdReference> und <PolicySetIdReference>. Der Wert ist die referenzierte Id.

Gegeben sein eine Policy mit der PolicyId

```
"urn:oasis:names:tc:xacml:2.0:gis:DenyAll"
```

welche jeglichen Zugriff untersagt und eine Policy

```
"urn:oasis:names:tc:xacml:2.0:gis:PermitAlice"
```

welche Alice einen WMS „GetMap“ Request auf das Feature „Straßen“ erlaubt. Folgendes PolicySet kombiniert diese zwei Policies.

```
<?xml version="1.0" encoding="UTF-8"?>
<PolicySet xmlns="urn:oasis:names:tc:xacml:2.0:policy:schema:os"
        xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
        PolicySetId="urn:oasis:names:tc:xacml:2.0:gis:PolicySetForAlice"
        PolicyCombiningAlgId=
        "urn:oasis:names:tc:xacml:1.0:policy-combining-algorithm:permit-overrides">
        <Target></Target>
        <PolicyIdReference>
                urn:oasis:names:tc:xacml:2.0:gis:PermitAlice
        </PolicyIdReference>
        <PolicyIdReference>
                urn:oasis:names:tc:xacml:2.0:gis:DenyAll
        </PolicyIdReference>
</PolicySet>
```

Auch ein <PolicySet> Element benötigt immer ein <Target> Element. In diesem Fall ist dieses leer, damit ist das PolicySet für jeden XACML Request zuständig.

Dieses Beispiel hat zwei Konfliktsituationen. Einerseits hat der PDP nun zwei Policies und ein PolicySet im Repository. Egal welcher XACML Request zu verarbeiten ist, der PDP findet mindestens zwei (wenn das Subject Alice ist sogar drei) Startpunkte für seine Auswertung. Damit ist das Ergebnis automatisch INDETERMINATE, da ja immer nur eine Policy oder ein PolicySet als Startpunkt zuständig sein darf.

Aus diesem Grund unterscheidet man Root (by request) und NonRoot (by reference) Policies/PolicySets. Für unser Beispiel muss das PolicySet ein Root Element sein, die beiden Policies aber nicht. Mit anderen Worten gesagt, dürfen die beiden Policies nur über <PolicyIdReference> Elemente gefunden und verarbeitet werden.

Ein weiteres wichtiges Attribute des <PolicySet> ist „PolicyCombiningAlgId“, in diesem Fall mit dem Wert

```
urn:oasis:names:tc:xacml:1.0:policy-combining-algorithm:permit-overrides
```

Bei einem XAMCL Request mit Subject ungleich „Alice“, würde die erste referenzierte Policy mit NOT_APPLICABLE, die zweite mit DENY evaluieren. Ein XAMCL Request mit Subject „Alice“ würde PERMIT gefolgt von DENY ergeben. Der Policy Combining Algorithmus

entscheidet nun, wie die Auswertungsresultate kombiniert werden. In diesem Fall (man kann das an dem Namen schon erahnen), genügt ein PERMIT und der Zugriff ist gewährt. (Bei Alice). In allen anderen Fällen ist die XACML Response DENY.

Die Kombinationsmöglichkeiten sind die gleichen wie bei den <Rule> Elementen. Man muss lediglich im URN „rule“ gegen „policy“ tauschen. Die Funktionalität ist gleichwertig.

XACML URN
rn:oasis:names:tc:xacml:1.0:policy-combining-algorithm:deny-overrides
urn:oasis:names:tc:xacml:1.0:policy-combining-algorithm:permit-overrides
urn:oasis:names:tc:xacml:1.0:policy-combining-algorithm:first-applicable
urn:oasis:names:tc:xacml:1.0:policy-combining-algorithm:only-one-applicable
urn:oasis:names:tc:xacml:1.1:policy-combining-algorithm:ordered-deny-overrides
urn:oasis:names:tc:xacml:1.1:policy-combining-algorithm:ordered-permit-overrides

Tabelle 2: Policy Combining Algorithmen

7 . XACML und RBAC

7.1 Einführung Rollenkonzept

In den bisherigen Beispielen wurde immer mit konkreten Subjekten gearbeitet (in unserem Falle die Benutzerkennungen „Bob“ und „Alice“, welche in der Security Literatur immer als Beispielbenutzer herhalten müssen)

Zum Zeitpunkt der Entwicklung einer Software weiß man natürlich nicht wie zukünftige Benutzer heißen, geschweige denn welchen Benutzergruppen sie angehören. (Ein Benutzer kann in mehreren Gruppen sein, es geht auch noch komplexer, indem Gruppen selbst wieder Mitglieder von anderen Gruppen sind)

Benutzer und Gruppen sind somit eine organisatorische bzw. administrative Einteilung beim Endbenutzer.

Es gibt nun mehrere Möglichkeiten, Authentifizierung und Zugriffskontrolle unabhängig von Benutzerkennungen zu implementieren. Ein auf Rollen basierender Ansatz, welcher auch im J2EE Environment verwendet wird, soll nun kurz erläutert werden.

Im Zuge der Entwicklung einer Applikation werden Rollen definiert. Beispiele:

- Interessent
- Kunde
- Sachbearbeiter
- Supervisor
- Administrator

Diese Rollen sind während der Entwicklung bekannt, J2EE sieht eine Zuordnung von Rollen zu URLs und zu EJB Methoden (Fachprozessen) vor, wobei einer Ressource mehreren Rollen zugeordnet werden können.

Während der Installation der Applikation (eng. Deployment) beim Endkunden werden nun den konkreten Benutzern und Gruppen die Rollen zugeordnet. Somit ist die Applikation universell einsetzbar.

Die Zugriffskontrolle läuft folgendermaßen ab:

1) Ein Benutzer wird authentifiziert.
2) Es werden alle seine Gruppen ermittelt
3) Für den Benutzer und seine Gruppen werden alle Rollen ermittelt, die ihm direkt oder indirekt über seine Gruppen zugewiesen sind.
4) Bei Zugriff auf eine geschützte Ressource werden die Rollen ermittelt, welche dieser Ressource zugeordnet sind.
5) Ist der Durchschnitt der Rollen ermittelt aus 3) mit den Rollen ermittelt aus 4) nicht leer, so ist der Zugriff gewährleistet.

In der Regel gibt es drei spezielle Rollen:

1) Rolle Administrator
   Darf immer auf alles zugreifen
2) Rolle Authentifiziert
   Alle Benutzer die sich authentifiziert haben
3) Rolle Anonym
   Alle nicht authentifizierten Benutzer.

Rollen können auch hierarchisch aufgebaut werden. So ist es intuitiv zu sagen, dass eine eingeloggter Benutzer, welche die Rolle „Sachbearbeiter“ hat, auch automatisch die Rolle „Authentifiziert“ hat Analog für „Supervisor“ und Sachbearbeiter. Mann muss hier allerdings folgendes beachten:

Nicht die Rollen sind hierarchisch, sondern die Zugriffsrechte !

Auf diesen Unterschied wird nachfolgend noch eingegangen.

Abschließend noch zwei Definitionen

Role Assignment Authority

Eine Komponente (Autorität), die einem Benutzer seine Rollen zuordnet, meistens eine Softwarekomponente.

Role Enablement Authority

Eine Komponente die prüft, ob eine Rolle zu einem gewissen Zeitpunkt in einem gegebenen Environment gültig ist.

7.2 RBAC

RBAC steht für „Role Based Access Control“. Die vollständige Spezifikation findet man hier.[5]

Diese Spezifikation beschreibt ein hierarchisches Rollenkonzept mit den Mitteln von XACML. Für den XACML Request ändert sich nicht all zu viel, man schickt anstelle der Benutzerkennung einfach die Rolle.

Bsp:

```
<?xml version="1.0" encoding="UTF-8"?>
<Request xmlns="urn:oasis:names:tc:xacml:2.0:context:schema:os"
        xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
        <Subject>
                <Attribute
                  AttributeId="urn:oasis:names:tc:xacml:2.0:subject:role"
                  DataType="http://www.w3.org/2001/XMLSchema#string">
                        <AttributeValue>ROLE_ADMINISTRATOR</AttributeValue>
                </Attribute>
        </Subject>
</Request>
```

AttributeId hat nun den Wert

```
urn:oasis:names:tc:xacml:2.0:subject:role
```

und anstelle von „Alice“ wird eine Rolle „ROLE_ADMINISTRATOR“ als <AttributeValue> verwendet.

Hat ein Benutzer mehrere Rollen, so wird für jede Rolle ein XACML Request erzeugt. Eine einzige PERMIT Response genügt damit der Zugriff gewährleistet ist. Diese Logik ist aber nach [5] im PEP zu implementieren.

RBAC gibt des weiteren Richtlinien zum Aufbau des XACML Repositories.

Role Policy Set (RPS)

Für jede Rolle gibt es genau ein RPS.

```
<?xml version="1.0" encoding="UTF-8"?>
<PolicySet xmlns="urn:oasis:names:tc:xacml:2.0:policy:schema:os"
 xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
 PolicySetId="org:geoserver:policySet:role:anonymous"
 PolicyCombiningAlgId=
 "urn:oasis:names:tc:xacml:1.0:policy-combining-algorithm:permit-overrides">
 <Target>
  <Subjects>
   <Subject>
    <SubjectMatch MatchId="urn:oasis:names:tc:xacml:1.0:function:anyURI-equal">
     <AttributeValue DataType="http://www.w3.org/2001/XMLSchema#anyURI">
      ROLE_ANONYMOUS
     </AttributeValue>
     <SubjectAttributeDesignator
      AttributeId="urn:oasis:names:tc:xacml:2.0:subject:role"
      DataType="http://www.w3.org/2001/XMLSchema#anyURI" />
    </SubjectMatch>
   </Subject>
  </Subjects>
 </Target>
 <PolicySetIdReference>
  org:geoserver:policySet:permission:anonymous
 </PolicySetIdReference>
</PolicySet>
```

Dieses RPS ist für die Rolle „ROLE_ANONYMOUS“ zuständig. Das Element <PolicySetIdReference> referenziert ein <PolicySet> mit der *PolicySetId*

```
  org:geoserver:policySet:permission:anonymous
```

Dieses referenzierte PolicySet enthält alle Zugriffsberechtigungen für die Rolle ROLE_ANONYMOUS und führt zur Definition von

Permission PolicySet (PPS)

Ein PPS enthält die Zugriffsberechtigung einer Rolle. Im Unterschied zu einem RPS darf dieses niemals gegen einen XACML Request evaluiert werden sondern ist nur über <policySetIdReference> Elemente erreichbar.

```
<?xml version="1.0" encoding="UTF-8"?>
<PolicySet xmlns="urn:oasis:names:tc:xacml:2.0:policy:schema:os"
 xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
 PolicySetId="org:geoserver:policySet:permission:anonymous"
 PolicyCombiningAlgId="
  urn:oasis:names:tc:xacml:1.0:policy-combining-algorithm:first-applicable">
 <Target></Target>
 <Policy xmlns="urn:oasis:names:tc:xacml:2.0:policy:schema:os"
       xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
       PolicyId="urn:oasis:names:tc:xacml:2.0:gis:DenyAlll"
      RuleCombiningAlgId=
          "urn:oasis:names:tc:xacml:1.0:rule-combining-algorithm:deny-overrides">
    <Rule Effect="Deny"
      RuleId="urn:oasis:names:tc:xacml:2.0:rule:DenyAll" >
..   </Rule>
 </Policy>
</PolicySet>
```

Dieses PPS beinhaltet nun eine einziges <Policy> Element, welchen jeglichen Zugriff für die Rolle ROLE_ANONYMOUS verbietet.

Hierarchische Rollen werden nun folgendermaßen abgebildet. Für eine neue Rolle wird standardmäßig ein RPS erzeugt. Das zugehörige PPS hat dann ein <PolicySetIdReference> Element, welches das „Parent“ PPS Element referenziert. Als Beispiel „erbt“ hier die Rolle ROLE_AUTHENTICTED von ROLE_ANONYMOUS. Zunächst das RPS.

RPS für ROLE_AUTHENTICATED

```
<?xml version="1.0" encoding="UTF-8"?>
<PolicySet xmlns="urn:oasis:names:tc:xacml:2.0:policy:schema:os"
 xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
 PolicySetId="org:geoserver:policySet:role:authenticated"
 PolicyCombiningAlgId=
 "urn:oasis:names:tc:xacml:1.0:policy-combining-algorithm:permit-overrides">
 <Target>
  <Subjects>
   <Subject>
    <SubjectMatch MatchId="urn:oasis:names:tc:xacml:1.0:function:anyURI-equal">
     <AttributeValue DataType="http://www.w3.org/2001/XMLSchema#anyURI">
      ROLE_AUTHENTICATED
     </AttributeValue>
     <SubjectAttributeDesignator
       AttributeId="urn:oasis:names:tc:xacml:2.0:subject:role"
       DataType="http://www.w3.org/2001/XMLSchema#anyURI" />
    </SubjectMatch>
   </Subject>
  </Subjects>
 </Target>
 <PolicySetIdReference>
  org:geoserver:policySet:permission:authenticated
 </PolicySetIdReference>
</PolicySet>
```

Das zugehörige PPS für ROLE_AUTHENTICATED

```
<?xml version="1.0" encoding="UTF-8"?>
<PolicySet xmlns="urn:oasis:names:tc:xacml:2.0:policy:schema:os"
 xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
 PolicySetId="org:geoserver:policySet:permission:authenticated"
  PolicyCombiningAlgId="
    urn:oasis:names:tc:xacml:1.0:policy-combining-algorithm:first-applicable">
  <Target></Target>


   <!-- EINFÜGEN Policy/PolicySet fuer ROLE_AUTHENTICATED -->

  <PolicySetIdReference>
      org:geoserver:policySet:permission:anonymous
   </PolicySetIdReference>
</PolicySet>
```

Hierarchische Strukturen werden also mittels <PolicyIdReference> und <PolicySetIdReference> Elementen zwischen den PPS abgebildet. Nun ist auch klar warum nicht Rollen hierarchisch miteinander verknüpft sind, sondern nur deren Zugriffsberechtigungen.

Weiters gibt es noch das Konzept des

Role Assignment PolicySet

Hier werden die Rollen selbst als Resource betrachtet. Eine Anwendung wäre z. Bsp die Einschränkung einer Rolle auf eine bestimmte Tageszeit.

Der Vollständigkeit halber soll hier noch das Konzept des

Has Permissions of Role PolicySet

erwähnt werden. Dabei handelt es sich um Policies, welche die Frage beantworten, ob ein Benutzer eine Rolle hat oder nicht (PERMIT oder DENY).

7.3 RBAC und J2EE

J2EE Security beinhaltet Mechanismen für die Authentifizierung und Zugriffskontrolle auf Basis von Rollen.

Geschützt werden können

1) URLs
2) EJB Methoden (Fachprozesse)

Die Authentifizierung sollte man der J2EE Infrastruktur auch überlassen, XACML ist ja ausschließlich für Zugriffskontrolle gedacht.

Es gibt leider keine Java API, mit der man die Rollen eines authentifizierten Benutzers abfragen kann. Zur Erinnerung, J2EE ist hier die „Role Assignment Authority“, RBAC die „Role Enablement Authority“.

Die Benutzer/Gruppen zu Rollen Zuordnung findet man in einem J2EE Konfigurationsfile (in XML Syntax). Das nützt wiederum auch nichts, da man ja nicht weiß, in welchen Gruppen der Benutzer ist.

Die beste Möglichkeit bietet JACC (Java Authorization Contract for Containers). JACC ermöglicht es, Authorization Plugins für einen J2EE Container zu installieren.

Innerhalb solch eines Plugins hat man alle Informationen die man benötigt. (Benutzer und dessen Rollen). Für jede geschützte Ressource (URL oder EJB Methode) wird das Plugin zum Zwecke der Zugriffskontrolle aufgerufen. Das Plugin selbst generiert dann den XACML Request, sendet diesen an den XACML Prozessor und evaluiert die Antwort.

In dieser Konstellation hat man aber das Problem, dass man keine Obligation zurückgeben kann. Hilfsmittel wäre hier ein „Thread Local Storage“ Objekt, welches für jeden Thread mögliche Obligationen beinhaltet.

Möchte man nur URLs schützen kann man auch mit einem Servlet Filter arbeiten, dies ist im noch folgenden Kapitel „Zugriffskontrolle auf Basis von URLs“ im Detail beschrieben

8 . GML und GeoXACML

Gemäß GeoXACML Spezifikation[6] können innerhalb einer Policy mehrere Geometrien angegeben werden. Da eine Policy als XML Dokument gespeichert wird, ist es nahe liegend, für die Darstellung von Geometrien GML zu verwenden.

Das Geometrie Modell basiert auf der Simple Features Spezifikation[7] des OGC. Es wird ein abstrakter XACML Datentyp „Geometrie“ eingeführt, welcher durch den URN

urn:ogc:def:dataType:geoxacml:1.0:geometry

eindeutig identifiziert wird. Es wird hier von der Möglichkeit Gebrauch gemacht,XACML um Datentypen zu erweitern.

Konkrete Geometrien sind nun

- Point
  Dieser kann 1, 2 oder 3 dimensional sein, Ordinaten werden mit x,y und z bezeichnet
- LineString
  Linienzug , sind Anfangs- und Endpunkt identisch, so spricht man auch von einem LinearRing. Letztere werden bei Polygonen benötigt.
- Polygon
  eine durch ein Polygon bestimmte Fläche , welche auch „Löcher“ haben kann
- MultiPoint
  Ansammlung von Punkten
- MultiLineString
  Ansammlung von Linienzügen
- MultiPolygon
  Ansammlung von Polygonen

Zu beachten ist hier das alle „Multi“ Typen auch als eine Geometrie angesehen werden.

Weiteres sollte man sich innerhalb einer Policy für GML 2 oder GML 3 entscheiden. Das Mischen von GML 2 und GML3 Geometrien innerhalb einer Policy muss nicht unterstützt werden.

8.1 GML Version 2

Der offizielle XML Namensraum (engl. Namespace) von GML (sowohl Version 2 als auch Version 3) ist

http://www.opengis.net/gml

Laut Extension A der GeoXACML Spezifikation müssen folgende GML V2 Datentypen unterstützt werden.

http://www.opengis.net/gml#Point
http://www.opengis.net/gml#LineString
http://www.opengis.net/gml#Polygon
http://www.opengis.net/gml#LinearRing
http://www.opengis.net/gml#Box
http://www.opengis.net/gml#MultiPoint
http://www.opengis.net/gml#MultiLineString
http://www.opengis.net/gml#MultiPolygon

Beispiel für einen Punkt:

```
<gml:Point xmlns:gml="http://www.opengis.net/gml" >
	<gml:coordinates >
		7.7,8.8
	</gml:coordinates>
</gml:Point>
```

oder alternativ

```
<gml:Point xmlns:gml="http://www.opengis.net/gml" >
	<gml:coord>
		<gml:X>7.7</gml:X>
		<gml:Y>8.8</gml:Y>
	</gml:coord>
</gml:Point>
```

Man beachte, dass die XML Elemente hier immer mit dem GML Namensraum angegeben werden, in einer Policy können Elemente aus unterschiedlichsten XML Dialekten vermischt sein.

Noch ein Beispiel für einen Linienzug

```
<gml:LineString xmlns:gml="http://www.opengis.net/gml">
	<gml:coordinates >
		-1.1,-2.2 -3.3,-4.4 -5.5,-6.6
	</gml:coordinates>
</gml:LineString>
```

Es fehlt jetzt allerdings noch das zugeordnete Koordinatensystem. Hierfür gibt es ein XML Attribute

srsName

Der Name leitet sich von „Spatial Reference System Name“ ab. Die Angabe des Namens von Koordinatensystemen kann leider auf verschiedene Art und Weise erfolgen.

Ein in GML V2 kodierter Punkt mit Angabe des Koordinatensystems kann so aussehen.

```
<gml:Point xmlns:gml="http://www.opengis.net/gml" srsName="EPSG:4326">
	<gml:coord>
		<gml:X>7.7</gml:X>
		<gml:Y>8.8</gml:Y>
	</gml:coord>
</gml:Point>
```

Für WGS 84 (hat bei EPSG den Code 4326) können z. Bsp. folgende Bezeichnungen verwendet.werden

1. WGS84
2. urn:ogc:def:crs:OGC:1.3:CRS84
3. EPSG:4326
4. http://www.opengis.net/gml/srs/epsg.xml#*4326*
5. urn:ogc:def:crs:EPSG:6.6:4326

Es ist also gar nicht so einfach, aus dem Wert des „srsName“ Attributes das entsprechende Koordinatensystem abzuleiten. Am besten eignen sich die URNs, in obiger Liste unter 2) und 5) angegeben. Diese URNs sind durch das OGC genormt, bestehen aus 7 Bestandteilen, welche durch einen Doppelpunkt getrennt sind. Zerlegt man

urn:ogc:def:crs:EPSG:6.6:4326

in seine Bestandteile, so erhält man folgende Komponenten:

1. urn
   Konstant, steht für Uniform Resource Name
2. ogc
   Steht für das OGC
3. def
   Gibt zum Ausdruck dass es sich um eine Definition handelt.
4. crs
   Beschreibt um welche Komponente es sich handelt, in unserem Fall ein „Coordinate Reference System".
5. EPSG
   Die Authority (ausgebende Stelle) von welcher die Definition stammt.
6. 6.6
   Die Versionsnummer der Definition
7. 4326
   Die eindeutige Bezeichnung innerhalb der Authority.

Weiters muss man noch die Lage der Achsen im Koordinatensystem berücksichtigen. Die Definition von EPSG:4326 hat als erste Achse die Breitenkreise, dies entspricht nicht dem intuitiven „xy" Denken aus einem kartesischen Koordinatensystem.

```
            <gml:X>7.7</gml:X>
            <gml:Y>8.8</gml:Y>
```

Diese Angaben entsprechen dem nördlichen Breitengrad 7,7 und dem westlichen Längengrad 8,8. Die GML Elementnamen gml:X und gml:Y sind hier etwas unglücklich.

Verwendet man allerdings „urn:ogc:def:crs:OGC:1.3:CRS84" als Koordinatensystemreferenz so stimmt die XY Intuition wieder, da diese Definition Längengrade als x, und Breitengrade als y vorgibt.

Man sieht an diesem Beispiel sehr deutlich, dass die Interpretation von x und y vom Koordinatensystem abhängig ist.

8.2 GML Version 3

Laut Extension B der GeoXACML Spezifikation müssen folgende GML V3 Datentypen unterstützt werden.

http://www.opengis.net/gml#Point
http://www.opengis.net/gml#LineString
http://www.opengis.net/gml#Polygon
http://www.opengis.net/gml#LinearRing
http://www.opengis.net/gml#Envelope
http://www.opengis.net/gml#MultiPoint
http://www.opengis.net/gml#MultiCurve
http://www.opengis.net/gml#MultiSurface

Bsp für einen Punkt

```
<gml:Point gid="Point" srsName="EPSG:4326"
        xmlns:gml="http://www.opengis.net/gml">
        <gml:pos >
                7.7 8.8
        </gml:pos>
/gml:Point>
```

oder ein Linienzug

```
<gml:LineString gid="LineString" srsName="EPSG:4326"
        xmlns:gml="http://www.opengis.net/gml">
        <gml:posList >
                -1.1 -2.2 -3.3 -4.4 -5.5 -6.6
        </gml:posList >
</gml:LineString>
```

Die in GML V2 vorhandenen Elemente „X“, „Y“ und „Z“ gibt es in GML V3 nicht mehr. Hier werden Koordinaten nur mehr als Auflistung von Gleitkommazahlen (Double) angegeben.

Somit stellt sich die Frage nach der Dimension (1,2 oder 3) und auch das Problem mit den Achsen ist wieder gegeben.

Beide Parameter ergeben sich, wenn nicht explizit angegeben, aus dem verwendeten Koordinatensystem. Um aber auch Spezialfälle (zwei dimensionale Koordinaten mit M oder Z Werten) verarbeiten zu können, gibt es in GML V3 drei zusätzliche Attribute:

- *srsDimension*
  Angabe der Dimension, 1,2 oder 3
- *axisLabels*
  Angabe der Achsenausrichtung
- *uomLabels* (Unit Of Measure)
  Die Einheit im Koordinatensystem (diese muss angegeben werden, wenn *axisLabels* angegeben ist.

Beispiel mit Verwendung obiger Attribute
:

```
<gml:LineString gid="LineString" srsName="EPSG:4326"
      srsDimension="2" axisLabels="Lat Long" uomLables="degree degree"
      xmlns:gml="http://www.opengis.net/gml">
      <gml:posList >
            -1.1 -2.2 -3.3 -4.4 -5.5 -6.6
      </gml:posList >
</gml:LineString>
```

Nun ist ist nicht mehr notwendig das Koordinatensystem zu untersuchen, alle notwendigen Angaben sind in den Attributen angegeben.

Die möglichen Werte für *axisLabel* und *uomLabels* sind seitens GML V3 vorgegeben.

9 . GeoXACML

9.1 Einführung

GeoXACML erweitert XACML um räumliche Zugriffsberechtigungen.
Nachfolgend sind 3 elementare Arten von Einschränkungen dargestellt.

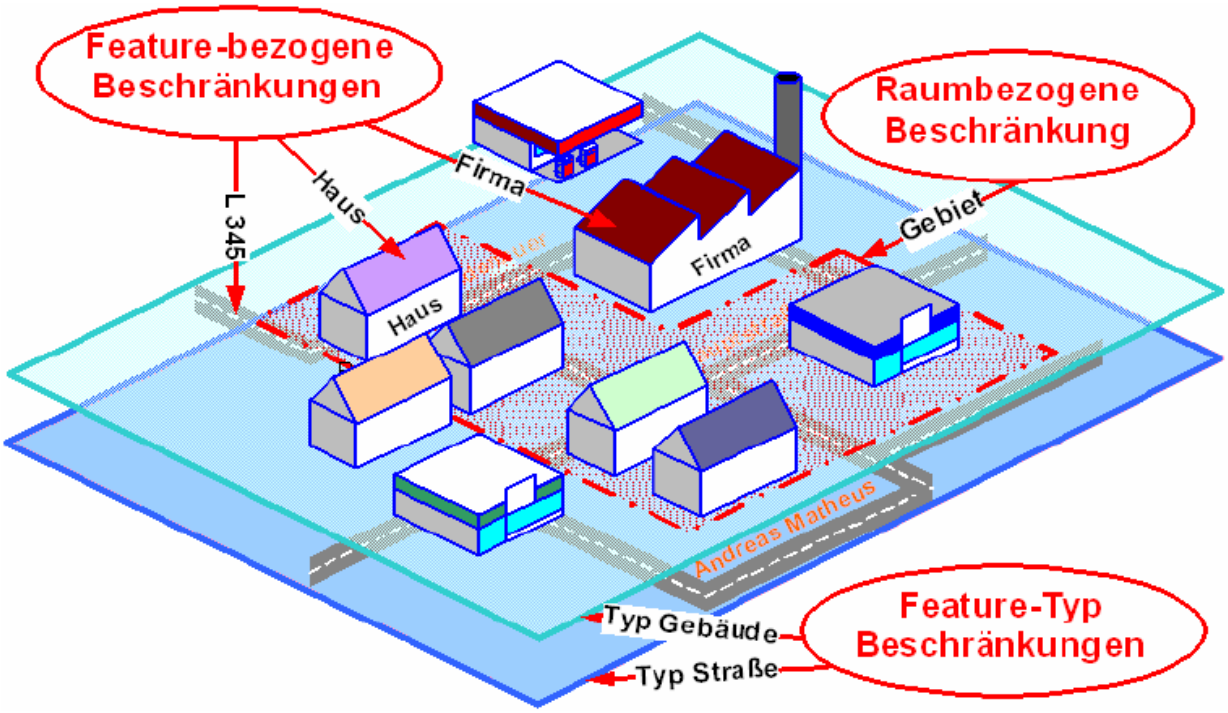


Abbildung 2: Zu schützende Resourcen

Entnommen aus [8]

1. Feature-Typ bezogene Beschränkungen.
   Der Zugriff auf einen Feature-Typ wird gewährt oder nicht.
   Feature-Typen werden oft auch als „Layer“ bezeichnet. In obiger
   Abbildung sind Gebäude und Straßen als Beispiel angegeben.
2. Feature bezogene Beschränkungen
   Hier wird die Zugriffsberechtigung für einzelne Features
   festgelegt, also für konkrete Gebäude oder Straßen

3. Raumbezogene Beschränkungen
   Diese Art der Zugriffsberechtigung ist der eigentliche Ausgangspunkt für GeoXACML. Es wird ein räumlicher Bereich festgelegt und diesem werden Berechtigungsinformationen zugeordnet.

Während Punkt 1 und 2 Problemstellungen der klassischen IT sind, ist Punkt 3 dem GIS Bereich zuzuordnen. Hier ist für die beiden ersten Anforderungen XACML zuständig und für Anforderung 3 eben GeoXACML. Da ja letzteres eine Erweiterung von XACML ist, sollte die ganze Bandbreite abgedeckt sein.

9.2 Räumliche Funktionen in GeoXACML

Das Geometriemodell wurde schon im Kapitel über GML vorgestellt. Nochmals eine kurze Zusammenfassung

Der XACML Datentyp ist

```
urn:ogc:def:dataType:geoxacml:1.0:geometry
```

Es werden Point,LineString,Polygon, Multipoint, MultiLineString und Multipolygon unterstützt

Geometrien werden in GML 2 oder GML 3 kodiert.

Räumliche Funktionen führen nun Operationen mit diesem Datentyp aus, wobei diese Funktionen Geometrien als Parameter verarbeiten aber auch neue Geometrien erzeugen können.

Alle Berechnungen basieren auf der Annahme das die Einheit Meter und folglich für Flächen Quadratmeter ist. Für die Umrechnung in andere Einheiten gibt es eine eigene Funktion. Weiters werden die Datentypen Set und Bag für Geometrien benötigt.

Geometry Set:

Eine Menge von Geometrien, jede Geometrie kann aber maximal einmal in dieser Menge vorkommen. Auch ein leeres Set ist möglich.

Geometry Bag:

Eine Menge von Geometrien, jede Geometrie kann beliebig oft vorkommen. Auch ein Bag kann leer sein.

Eine GeoXACML Implementierung kann Transformationen zwischen verschiedenen Koordinatensystemen. unterstützten. Diese Situation tritt dann auf, wenn in der Policy eine Geometrie mit einem Koordinatensystem A angegeben ist, im GeoXACML Request eine Geometrie mit einem Koordinatensystem B kodiert ist.

Unterstützt nun eine Implementierung keine Koordinatentransformationen, so muss mit einem Fehler geantwortet werden, für GeoXACML ist dies die Antwort INDETERMINATE.

Werden Transformationen unterstützt, so wird eine der Geometrien transformiert und danach die geometrische Operation durchgeführt.

Aus Gründen der Performance sollte diese Möglichkeit aber nicht überstrapaziert werden, eine Prüfung der Zugriffsberechtigung sollte schnell sein und nicht in komplexen mathematischen Berechnungen münden. Haben die Geometrien in der Policy eine überschaubare Anzahl von Stützpunkten, so spricht allerdings nichts gegen Koordinatentransformationen.

Nachfolgend werden nun die verfügbaren Funktionen logisch gruppiert besprochen und pro Gruppe eine Policy als Beispiel angegeben.

9.2.1 Topologische Funktionen

Topologische Funktion testen zwei Geometrien auf topologische Eigenschaften. All diese Funktionen haben zwei Geometrien als Parameter und retournieren TRUE oder FALSE in Abhängigkeit des jeweiligen Tests.

Contains(g1 Geometrie, g2 Geometrie)

Retourniert TRUE wenn g1 innerhalb g2 liegt

Crosses(g1 Geometrie, g2 Geometrie)

Die Geometrien haben einen gemeinsamen Bereich, keine ist vollständig in der anderen enthalten und die Dimension der Überschneidung ist kleiner als die der Ausgangsgeometrien. Als Beispiel kann man sich 2 kreuzende Linien vorstellen, die Dimension der Linien ist 1, die des Kreuzungspunktes ist 0.

Disjoint (g1 Geometrie, g2 Geometrie)

Die Geometrien haben keinen einzigen gemeinsamen Punkt

Equals (g1 Geometrie, g2 Geometrie)

Die beiden Geometrien sind punktgleich

Intersects (g1 Geometrie, g2 Geometrie)

Die beiden Geometrien haben mindestens einen Punkt gemeinsam, also das Gegenteil von Disjoint.

Overlaps (g1 Geometrie, g2 Geometrie)

Beide Geometrien haben gemeinsame Punkte und eigene Punkte. Die Dimension der Überschneidung ist gleich der Dimension von g1 und g2.

Touches (g1 Geometrie, g2 Geometrie)

Beide Geometrien haben zumindest einen Randpunkt gemeinsam, aber keine inneren Punkte.

Within (g1 Geometrie, g2 Geometrie)

Geometrie g1 ist vollständig in g2 enthalten. Siehe auch Contains.

Tabelle der Zuordnung der GeoXACML URN Namen zu den topologischen Funktionen.

URN	Funktion
urn:ogc:def:function:geoxacml:1.0:geometry-equals	Equals
urn:ogc:def:function:geoxacml:1.0:geometry-disjoint	Disjoint
urn:ogc:def:function:geoxacml:1.0:geometry-touches	Touches
urn:ogc:def:function:geoxacml:1.0:geometry-crosses	Crosses
urn:ogc:def:function:geoxacml:1.0:geometry-within	Within
urn:ogc:def:function:geoxacml:1.0:geometry-contains	Contains
urn:ogc:def:function:geoxacml:1.0:geometry-overlaps	Overlaps
urn:ogc:def:function:geoxacml:1.0:geometry-intersects	Intersects

Tabelle 3: URNs Topologie Funktionen

Die Verwendung soll nun anhand eines Beispiels gezeigt werden. Gegeben sei folgender GeoXACML Request.

```
<?xml version="1.0" encoding="UTF-8"?>
<Request xmlns="urn:oasis:names:tc:xacml:2.0:context:schema:os"
        xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
        <Subject>
                <Attribute
                        AttributeId="urn:oasis:names:tc:xacml:2.0:subject:role"
                        DataType="http://www.w3.org/2001/XMLSchema#string">
                        <AttributeValue>ROLE_ANONYMOUS</AttributeValue>
                </Attribute>
        </Subject>
        <Resource>
                <Attribute
                        AttributeId="urn:oasis:names:tc:xacml:1.0:resource:resource-id"
                        DataType="urn:ogc:def:dataType:geoxacml:1.0:geometry">
                        <AttributeValue>
                        <gml:Box gid="box1"  xmlns:gml="http://www.opengis.net/gml" >
                                <gml:coord>
                                        <gml:X>11</gml:X>
                                        <gml:Y>11</gml:Y>
                                </gml:coord>
                                <gml:coord>
                                        <gml:X>12</gml:X>
                                        <gml:Y>12</gml:Y>
                                </gml:coord>
                        </gml:Box>
                        </AttributeValue>
                </Attribute>
        </Resource>
        <Action>
                <Attribute
                        AttributeId="urn:oasis:names:tc:xacml:1.0:action:action-id"
                        DataType="http://www.w3.org/2001/XMLSchema#string">
                        <AttributeValue>read</AttributeValue>
                </Attribute>
        </Action>
        <Environment/>
</Request>
```

Der Request ist in drei Hauptelemente unterteilt. Das <Subject> und <Action> Element haben gegenüber XACML keine Besonderheiten. In das Resource Elemente ist allerdings eine GML Geometrie kodiert. Zusammenfassend steht im Request folgendes:

Die Rolle ROLE_ANONYMOUS will lesend (<Action> Element „read“) auf das Rechteck (11,11) (12,12) zugreifen. Der Einfachheit halber ist vorerst nicht einmal ein Layer Name angegeben.

Bei diesem Beispiel wird ein vollständiges XML Dokument für Request und Policy angegeben.Um die Übersichtlichkeit zu wahren, werden für weitere Beispiele nur mehr die interessanten XML Fragmente (bezüglich GeoXACML) angegeben.

Im Request ist das <Resource> Element also die Region, auf die zugegriffen werden soll. Als Benutzer (<Subject>) wird fix „die Rolle ROLE_ANONYMOUS verwendet und die angeforderte Zugriffsart ist immer „read“ (<Action> Element).

Im Policy XML interessiert uns das Element <Condition> genauer. Hier werden GeoXACML spezifische Elemente verwendet um die Zugriffsberechtigung zu überprüfen.

Zusammenfassend sagt nachfolgende Policy aus, dass die Rolle ROLE_ANONYMOUS Leseberechtigung im Rechteck (10,10) (20,20) hat, der Zugriff wird also erlaubt. Nachfolgend nun das komplette Listing und anschließend eine genauere Betrachtung des <Condition> Elements.

```
<?xml version="1.0" encoding="UTF-8"?>
<Policy xmlns="urn:oasis:names:tc:xacml:2.0:policy:schema:os"
 xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
 PolicyId="urn:oasis:names:tc:xacml:2.0:conformance-test:IIA1:policy"
 RuleCombiningAlgId=
       "urn:oasis:names:tc:xacml:1.0:rule-combining-algorithm:deny-overrides">
 <Description>TestContains</Description>
 <Target />
 <Rule Effect="Permit" RuleId="RuleXY">
  <Target>
   .<Subjects>
     <Subject>
      <SubjectMatch
         MatchId="urn:oasis:names:tc:xacml:1.0:function:string-equal">
       <AttributeValue
          DataType="http://www.w3.org/2001/XMLSchema#string">
          ROLE_ANONYMOUS
       </AttributeValue>
       <SubjectAttributeDesignator
         DataType="http://www.w3.org/2001/XMLSchema#string"
         AttributeId="urn:oasis:names:tc:xacml:2.0:subject:role" />
      </SubjectMatch>
     </Subject>
   </Subjects>
   <Actions>
    <Action>
     <ActionMatch MatchId="urn:oasis:names:tc:xacml:1.0:function:string-equal">
       <AttributeValue DataType=
        "http://www.w3.org/2001/XMLSchema#string">read</AttributeValue>
       <ActionAttributeDesignator
            AttributeId="urn:oasis:names:tc:xacml:1.0:action:action-id"
            DataType="http://www.w3.org/2001/XMLSchema#string" />
     </ActionMatch>
    </Action>
..  </Actions>
..  </Target>
.   <Condition >
    <Apply FunctionId="urn:oasis:names:tc:xacml:1.0:function:all-of">
..   <Function FunctionId="urn:ogc:def:function:geoxacml:1.0:geometry-contains"/>
    <AttributeValue DataType="urn:ogc:def:dataType:geoxacml:1.0:geometry">
      <gml:Polygon gid="Europe"
       xmlns:gml="http://www.opengis.net/gml"
         <gml:exterior>
```

```
            <gml:LinearRing>
              <gml:posList srsDimension="2">
                10 10 10 20 20 20 20 10 10 10
              </gml:posList>
            </gml:LinearRing>
          </gml:exterior>
        </gml:Polygon>
      </AttributeValue>
      <ResourceAttributeDesignator
           AttributeId="urn:oasis:names:tc:xacml:1.0:resource:resource-id"
           DataType="urn:ogc:def:dataType:geoxacml:1.0:geometry"/>
    </Apply>
  </Condition>
..</Rule>
</Policy>
```

Das <Condition> Element im Detail

```
<Condition >
 <Apply FunctionId="urn:oasis:names:tc:xacml:1.0:function:all-of">
  <Function FunctionId="urn:ogc:def:function:geoxacml:1.0:geometry-contains"/>
  <AttributeValue DataType="urn:ogc:def:dataType:geoxacml:1.0:geometry">
    <gml:Polygon gid="Europe"
     xmlns:gml="http://www.opengis.net/gml"
          <gml:exterior>
           <gml:LinearRing>
             <gml:posList srsDimension="2">
               10 10 10 20 20 20 20 10 10 10
             </gml:posList>
           </gml:LinearRing>
         </gml:exterior>
       </gml:Polygon>
  </AttributeValue>
  <ResourceAttributeDesignator
     AttributeId="urn:oasis:names:tc:xacml:1.0:resource:resource-id"
     DataType="urn:ogc:def:dataType:geoxacml:1.0:geometry"/>
 </Apply>
</Condition>
```

Das Ergebnis eines <Condition> Elements muss immer vom Typ Boolean sein.

Das Element <AttributeValue> enthält ein mittels GML beschriebenes Rechteck.

Das Element <ResourceAttributeDesignator> referenziert den den GeoXACML Request. Der GeoXACML Prozessor muss nun aus dem Request das Attribut mit der *AttributeId*

"urn:oasis:names:tc:xacml:1.0:resource:resource-id"

und dem Datentyp

"urn:ogc:def:dataType:geoxacml:1.0:geometry"

heraussuchen. Da aber Attribute mehrfach vorkommen dürfen, ist hier das Ergebnis ein Bag (Menge) von <AttributeValue> Elementen. In diesem Fall ist dies ein Bag mit einem Element, dem Rechteck aus dem Request.

Das Element <Function> hat die *FunctionId*

"urn:ogc:def:function:geoxacml:1.0:geometry-contains"

Es soll also getestet werden ob das Rechteck aus der Policy das Rechteck aus dem Request beinhaltet. Die **Contains** Funktion benötigt als Argumente aber zwei Geometrien, im Moment haben wir nur eine Geometrie aus der Policy und einen Bag der eine Geometrie aus dem Request enthält.

Aus diesem Grund verwendet man wieder eine „High Order Bag Function“. Das Element <Apply> hat die *FunctionId*

"urn:oasis:names:tc:xacml:1.0:function:all-of"

Diese Funktion benötigt 3 Argumente.

a) ein <Function> Element. welches einen Boolean Wert zurück gibt und 2 Argumente benötigt (Hier Contains)

b) Erstes Argument ist ein beliebiges Attribut.

c) Zweites Argument ist ein Bag

Das erste Argument (Box (10,10) (20,20)) wird nun mit jedem Objekt im Bag gepaart und die Funktion (Contains) wird aufgerufen. Evaluieren alle Funktionsaufrufe zu TRUE, so ist das Ergebnis TRUE, andernfalls FALSE.

9.2.2 Geometrische Funktionen

Diese Funktionen erzeugen neue Geometrien. Das Ergebnis ist entweder eine neue Geometrie oder ein Bag bestehend aus solchen.

Buffer(g Geometrie, d Double) : Bag

Die Geometrie wird mit dem Wert von d gepuffert und das Ergebnis als Bag von Geometrien zurückgegeben.

Bsp: Sei g ein Punkt, d gleich 100 m so ist das Ergebnis ein Bag mit einer Geometrie. Diese Geometrie ist ein Polygon welches einen Kreis darstellt.

Die Distanz ist in Metern anzugeben.

Boundary(g Geometrie) : Bag

Berechnet die Grenzen einer Geometrie, bei einer Linie zum Beispiel die 2 Endpunkte.

ConvexHull(g Geometrie) : Geometrie

Berechnet die konvexe Hülle der Geometrie. Die konvexe Hülle ist das kleinste konvexe Polygon das alle Punkte der Geometrie g enthält. Die konvexe Hülle eines Punktes ist der Punkt selbst

Centroid(g Geometrie)

Berechnet jenen Punkt, der das geometrische Zentrum repräsentiert.

Difference (g1 Geometrie, g2 Geometrie) : Bag

Berechnet die Geometrien der geometrischen Differenz von g1 weniger g2. Es sind alle Punkte enthalten welche in g1, nicht aber in g2 liegen.

SymDifference(g1:Geometrie, g2:Geometrie) : Bag

Berechntet die Geometrie der symmetrischen geometrischen Differenz. Enthalten sind alle Punkte welche entweder in g1 oder in g2, nicht aber in g1 und g2 liegen.

Intersection(g1:Geometrie, g2:Geometrie) : Bag

Berechntet die Geometrie des geometrischen Durchschnittes. Enthalten sind alle Punkte welche entweder in g1 und in g2 liegen

Tabelle der Zuordnung der GeoXACML URN Namen zu den geometrischen Funktionen.

URN	Funktion
urn:ogc:def:function:geoxacml:1.0:geometry-buffer	Buffer
urn:ogc:def:function:geoxacml:1.0:geometry-boundary	Boundary
urn:ogc:def:function:geoxacml:1.0:geometry-union	Union
urn:ogc:def:function:geoxacml:1.0:geometry-intersection	Intersection
urn:ogc:def:function:geoxacml:1.0:geometry-difference	Difference
urn:ogc:def:function:geoxacml:1.0:geometry-sym-difference	SymDifference
urn:ogc:def:function:geoxacml:1.0:geometry-centroid	Centroid
urn:ogc:def:function:geoxacml:1.0:geometry-convex-hull	ConvexHull

Tabelle 4: URNs Geometrie Funktionen

Als Beispiel hier wieder das <Resource> Element eines Requests

```
<Resource>
 <Attribute   AttributeId="urn:oasis:names:tc:xacml:1.0:resource:resource-id"
              DataType="urn:ogc:def:dataType:geoxacml:1.0:geometry">
       <AttributeValue>
              <gml:Envelope gid="Box"  xmlns:gml="http://www.opengis.net/gml">
                     <gml:pos>0 0</gml:pos>
                     <gml:pos>10 10</gml:pos>
              </gml:Envelope>
       </AttributeValue>
 </Attribute>
</Resource>
```

Dieser Request möchte also auf einen Bereich (0,0) (10,10) zugreifen.

Nachfolgend ein <Condition> Element einer Policy, welches kontrolliert ob der Centroid aus der Request Geometrie gleich (5,5) ist und nur dann den Zugriff erlaubt.

```
<Condition >
 <Apply FunctionId="urn:ogc:def:function:geoxacml:1.0:geometry-equals">
  <Apply  FunctionId="urn:ogc:def:function:geoxacml:1.0:geometry-centroid">
   <Apply  FunctionId="urn:ogc:def:function:geoxacml:1.0:geometry-one-and-only">
      <ResourceAttributeDesignator
        AttributeId="urn:oasis:names:tc:xacml:1.0:resource:resource-id"
        DataType="urn:ogc:def:dataType:geoxacml:1.0:geometry"/>
   </Apply>
  </Apply>
   <AttributeValue DataType="urn:ogc:def:dataType:geoxacml:1.0:geometry">
              <gml:Point gid="Point"  xmlns:gml="http://www.opengis.net/gml" >
                     <gml:pos>5 5</gml:pos>
              </gml:Point>
  </AttributeValue>
 </Apply>
</Condition>
```

Am besten liest man obiges XML Fragment von innen nach außen. Ist ist also ein Punkt (5,5) in GML Syntax als <AttributeValue> gegeben.

Das <ResourceAttributeDesignator> Element liefert wieder einen Bag von Geometrien aus dem Request. Auf diesen Bag wird die Funktion mit der *FunctionId*

urn:ogc:def:function:geoxacml:1.0:geometry-one-and-only

angewendet. Diese Funktion verlangt einen Bag mit genau einem Element und retourniert dieses. Leere Bags oder Bags mit mehr als einem Element verursachen eine Fehlersituation.

Aus dieser Geometrie wird nun mittels

urn:ogc:def:function:geoxacml:1.0:geometry-centroid

der Centroid berechnet, welcher ja als Punkt auch wieder eine Geometrie ist.

Zuletzt werden die beiden Geometrien mittels

urn:ogc:def:function:geoxacml:1.0:geometry-equals

verglichen.

In diesem Fall evaluiert das GeoXACML Ergebnis zu PERMIT.

9.2.3 Skalare Funktionen

Skalare Funktionen berechnen einen Wert bezüglich vorgegebener Geometrie(n).

Area (g: Geometrie) : Double

Berechnet die Fläche in Quadratmetern. Für Punkt und Linienzüge ist das Ergebnis 0.

Distance (g1 Geometrie, g2:Geometrie) : Double

Berechnet die kürzeste Verbindung zwischen g1 und g2 in Metern.

IsWithinDistance(Geometrie g1, Geometrie g2, d Double) :Boolean

Retourniert TRUE wenn die kürzeste Distanz zwischen den beiden Geometrien kleiner oder gleich d ist.

Length (Geometrie g) : Double

Berechnet die Länge einer Geometrie. Bei Punkten ist das Resultat 0, bei Polygonen entspricht es dem Umfang.

Tabelle der Zuordnung der GeoXACML URN Namen zu den skalaren Funktionen.

URN	Funktion
urn:ogc:def:function:geoxacml:1.0:geometry-distance	Distance
urn:ogc:def:function:geoxacml:1.0:geometry-is-within-distance	IsWithinDistance
urn:ogc:def:function:geoxacml:1.0:geometry-length	Length
urn:ogc:def:function:geoxacml:1.0:geometry-area	Area

Tabelle 5: URNs Skalare Funktionen

Für nachfolgendes Beispiel wird der Request aus dem Centroid Beispiel verwendet, es wird also Zugriff auf das Rechteck (0,0) (10,10) gefordert. Dieses Rechteck hat also eine Fläche von 100 (Einheit hier bewusst ausgelassen).

```
<Condition >
 <Apply FunctionId="urn:oasis:names:tc:xacml:1.0:function:double-equal">
   <AttributeValue DataType="http://www.w3.org/2001/XMLSchema#double">
    100
   </AttributeValue>
    <Apply FunctionId="urn:ogc:def:function:geoxacml:1.0:geometry-area">
     <Apply FunctionId=
        "urn:ogc:def:function:geoxacml:1.0:geometry-one-and-only">
      <ResourceAttributeDesignator
        AttributeId="urn:oasis:names:tc:xacml:1.0:resource:resource-id"
        DataType="urn:ogc:def:dataType:geoxacml:1.0:geometry"/>
     </Apply>
    </Apply>
 </Apply>
</Condition>
```

Im Unterschied zu vorigem Beispiel wird hier von der Geometrie aus dem Request mittels

```
urn:ogc:def:function:geoxacml:1.0:geometry-area
```

die Fläche berechnet.

Diese Fläche wird dann mittels

```
urn:oasis:names:tc:xacml:1.0:function:double-equal
```

mit 100 verglichen. Ein Zugriff ist also nur erlaubt, wenn die Fläche der Geometrie aus dem Request gleich 100 ist.

9.2.4 Funktionen zum Überprüfen räumlicher Eigenschaften

IsSimple(g: Geometrie): Boolean

Einfache Geometrien berühren sich nur an ihren Endpunkten. Kreuzt sich die Geometrie mit sich selbst oder tangiert mit sich selbst , so ist diese nicht einfach.

IsClosed (g: Geometrie) : Boolean

Bei geschlossenen Geometrien ist der Start- und Endpunkt gleich. Punkte, Punktmengen und leere Geometrien sind immer geschlossenen. Geschlossene Geometrien haben keine Grenze.

IsValid (g: Geometrie) : Boolean

Prüft ob die Geometrie gültig laut OGC Spezifikation ist.

Tabelle der Zuordnung der GeoXACML URN Namen zu den Prüffunktionen.

URN	Funktion
urn:ogc:def:function:geoxacml:1.0:geometry-is-simple	IsSimple
urn:ogc:def:function:geoxacml:1.0:geometry-is-closed	IsClosed
urn:ogc:def:function:geoxacml:1.0:geometry-is-valid	IsValid

Hier wird der Zugriff nur gewährt wenn die Geometrie aus dem Request valid gemäß OGC ist.

```
<Condition >
  <Apply FunctionId="urn:ogc:def:function:geoxacml:1.0:geometry-is-valid">
    <Apply FunctionId="urn:ogc:def:function:geoxacml:1.0:geometry-one-and-only">
    <ResourceAttributeDesignator
      AttributeId="urn:oasis:names:tc:xacml:1.0:resource:resource-id"
      DataType="urn:ogc:def:dataType:geoxacml:1.0:geometry"/>
    </Apply>
  </Apply>
</Condition>
```

9.2.5 Bag Funktionen

GeometryOneAndOnly(b Bag) : Geometrie

Retourniert die Geometrie die im Bag vorhanden ist. Ist keine Geometrie oder sind mehr als eine Geometrie vorhanden, so ist das Ergebnis ein GeoXACML Processing Error INDETERMINATE.

GeometryBasSize(b Bag) : Integer

Anzahl der Elemente im Bag

GeometryIsIn (g Geometrie, ,b Bag) : Boolean

Überprüft ob die Geometrie g im Bag b vorhanden ist. Zum Testen einzelner Geometrien muss die Funktion Equals verwendet werden.

GeometryBag(g* Geometrie): Bag

Erzeugt einen Bag bestehend aus den Geometrien g*. g* steht hier für 0 ...n Geometrien.

Tabelle der Zuordnung der GeoXACML URN Namen zu den Bag Funktionen.

URN	Funktion
urn:ogc:def:function:geoxacml:1.0:geometry-one-and-only	GeometryOneAndOnly
urn:ogc:def:function:geoxacml:1.0:geometry-bag-size	GeometryBagSize
urn:ogc:def:function:geoxacml:1.0:geometry-is-in	GeometryIsIn
urn:ogc:def:function:geoxacml:1.0:geometry-bag	GeometryBag

Tabelle 6: URNs Bag Funktionen

Für diese Funktionen ist kein Beispiel angegeben da sie ja in vorigen XML Fragmenten schon verwendet wurden.

9.2.6 Set Funktionen

GeometryBagIntersection(b1 Bag, b2 Bag): Bag

Ergebnis ist ein Bag von Geometrien, welche in b1 und b2 vorkommen. Gleichheit von 2 Geometrien wird mit Equals bestimmt, keine Geometrie kommt mehrfach vor.

GeometryBagLeastOneMemberOf(b1 Bag, b2 Bag) : Boolean

Evaluiert zu TRUE wenn mindestens eine Geometrie aus b1 auch in b2 ist.

GeometryBagUnion(b1 Bag, b2 Bag) : Bag

Erzeugt einen Bag mit allen Elementen aus b1 und b2. Keine Geometrie kommt mehrfach vor

GeometryBagSubset(b1 Bag, b2 Bag) : Boolean

Evaluiert zu TRUE wenn alle Geometrien von b1 auch in b2 vorhanden sind.

GemoetrySetEquals (Bag b1, Bag b2) : Boolean

TRUE wenn alle Elemente von b1 in b2 enthalten sind und umgekehrt.

Tabelle der Zuordnung der GeoXACML URN Namen zu den Set Funktionen.

URN	Funktion
urn:ogc:def:function:geoxacml:1.0:geometry-bag-intersection	GeometryBagIntersection
urn:ogc:def:function:geoxacml:1.0:geometry-at-least-one-member-of	GeometryAtLeastOneMemberOf
urn:ogc:def:function:geoxacml:1.0:geometry-bag-union	GeometryBagUnion
urn:ogc:def:function:geoxacml:1.0:geometry-bag-subset	GeometryBagSubset
urn:ogc:def:function:geoxacml:1.0:geometry-set-equals	GeometrySetEquals

Tabelle 7: URNs Set Funktion

Als Beispiel ein <Condition> Element das zu PERMIT evaluiert wenn die Geometrie im Request der Punkt (5,5) ist.

```
<Condition >
 <Apply FunctionId="urn:ogc:def:function:geoxacml:1.0:geometry-set-equals">
  <ResourceAttributeDesignator
    AttributeId="urn:oasis:names:tc:xacml:1.0:resource:resource-id"
    DataType="urn:ogc:def:dataType:geoxacml:1.0:geometry"/>
  <Apply FunctionId="urn:ogc:def:function:geoxacml:1.0:geometry-bag">
   <AttributeValue DataType="urn:ogc:def:dataType:geoxacml:1.0:geometry">
    <gml:Point gid="Point"  xmlns:gml="http://www.opengis.net/gml" >
     <gml:pos>5 5</gml:pos>
    </gml:Point>
   </AttributeValue>
  </Apply>
 </Apply>
Condition>
```

Die Funktion

urn:ogc:def:function:geoxacml:1.0:geometry-bag

erzeugt einen Bag mit Inhalt des Punktes (5,5).Die Funktion

urn:ogc:def:function:geoxacml:1.0:geometry-set-equals

vergleicht diesen Bag mit dem Bag aus GeoXACML Request. Letzterer wird ja durch das Element <ResourceAttributeDesignator> erzeugt.

9.2.7 Funktionen zur Konvertierung

Da GeoXACML immer von Metern ausgeht, gibt es Funktionen um von anderen Maßeinheiten in Meter (bzw. Quadratmeter) zu konvertieren.

ConvertToMetre(d Double, u String) : Double

u steht hier für „Unit of Measure“, also die Einheit. Konvertiert den Wert d in Einheit u auf Meter.

ConvertToSquareMetre(d Double, u String) : Double

Analog wie ConvertToMetre, nur für Flächen

Tabelle der Zuordnung der GeoXACML URN Namen zu den Konvertierfunktionen.

URN	Funktion
urn:ogc:def:function:geoxacml:1.0:convert-to-metre	ConvertToMetre
urn:ogc:def:function:geoxacml:1.0:convert-to-square-metre	ConvertToSquareMetre

Tabelle 8: URNs Konverter Funktionen

Nachfolgend ein <Apply> Element als Beispiel. Hier wird von der Geometrie aus dem Request die Länge (oder Umfang) berechnet. Die Einheit der Länge ist Yard und wird dann in Meter umgerechnet.

```
<Apply FunctionId="urn:ogc:def:function:geoxacml:1.0:convert-to-metre">
 <Apply FunctionId="urn:ogc:def:function:geoxacml:1.0:geometry-length">
  <Apply FunctionId="urn:ogc:def:function:geoxacml:1.0:geometry-one-and-only">
   <ResourceAttributeDesignator
    AttributeId="urn:oasis:names:tc:xacml:1.0:resource:resource-id"
    DataType="urn:ogc:def:dataType:geoxacml:1.0:geometry"/>
  </Apply>
 </Apply>
 <AttributeValue
   DataType="http://www.w3.org/2001/XMLSchema#string">yd</AttributeValue>
</Apply>
```

9.3 GeoXACML Performance

Es werden nun einige Richtlinien vorgestellt um eine vernünftige Performance der GeoXACML Komponenten zu erreichen. Zugriffskontrolle kostet Zeit, diese muss sich aber in einem gewissen Rahmen halten.

Wenn möglich keine Umrechnungsfunktionen in der Policy

Konstanten kann man im Vorhinein auf Meter (Quadratmeter) umrechnen. Bezüglich Performance ist dies allerdings das kleinste Problem.

Nicht zu viele Request Matching Policies

Bei den Policies unterscheidet man zwischen jenen die nur mittels ID referenziert werden und niemals gegen einen Request evaluiert werden und jenen, welche gegen einen Request evaluiert werden.

Letztere Gruppe sollte nicht zu zahlreich sein. Wird keine passende Policy gefunden, so ist das Ergebnis NOT_APPLICABLE. Wenn aber mehr als eine passende Policy gefunden wird ist eine Fehlersituation gegeben und das Resultat ist INDETERMINATE. Um letzteren Fall zu erkennen muss der GeoXACML Prozessor immer alle Policies gegen den Request evaluieren.

Bei einem Rollenkonzept gemäß RBAC ist die Anzahl der zu untersuchenden Policies pro Request gleich der Anzahl der vorhanden Rollen,nämlich alle Role Policy Sets. Diese sind sehr einfach aufgebaut denn sie haben nur ein <SubjectMatch> Element. Somit sind die Voraussetzungen für eine schnelles ermitteln des Permission Policy Sets gegeben.

Verwendung von einfachen Geometrien in der Policy

Es macht keinen Sinn „perfekte“ Geometrien in die Policy aufzunehmen. Die Verwendung von Geometrien mit vielen Stützpunkten macht aus vielerlei Hinsicht Probleme.

Als Illustration ein praktisches Szenario. Ausgangsbasis ist ein Feature Kontinente in dem alle Kontinente auf 100 m genau vorhanden sind. Der Zugriff soll nun auf Europa eingeschränkt werden.

Hierfür eignet sich natürlich die GeoXACML Contains Funktion. Europa selbst ist Aufgrund der Inseln ein Multipolygon.

Dies bedeutet eine sehr große Anzahl an Stützpunkten, welche in GML Form in der Policy hinterlegt werden müssen. Da die Policy aber nur einmal eingelesen wird kann man das in Kauf nehmen. (Vorausgesetzt man hat genügend viel Hauptspeicher zur Verfügung).

Zur Vereinfachung gehen wir von einem WMS Request aus, welcher einen rechteckigen Ausschnitt aus der Karte anfordert. Bei jedem dieser Requests muss nun geprüft werden, ob das Rechteck auf europäischen Land liegt

Nachfolgende Abbildung veranschaulicht das Problem, die grün schraffierte Fläche zeigt bereits ein Lösung

Abbildung 3: Europa als vereinfachtes Polygon

Man sieht hier die exakten europäischen Grenzen und ein sehr vereinfachtes Polygon, welches auch Gebiete beinhaltet, die eigentlich nicht mehr zu Europa gehören.

Das vereinfachte Polygon hat nun ganz wenige Stützpunkte, dementsprechend schnell ist auch der Containment Test. Auch hat das vereinfachte Polygon nun den Vorteil, dass man auch Meeresgebiete sehen kann, beim komplexen Polygon könnte man niemals den Ärmelkanal sehen, da er ja nicht enthalten ist.

Was würde passieren wenn ein Rechteck (z.Bsp die Bounding Box aus

einem WMS Request) zum Teil im Meer (aber innerhalb des einfachen Polygons) und auf dem Festland liegt ?. Beim einfachen Polygon ist das egal, beim komplexen Multipolygon muss die Antwort DENY sein.

Anmerkung: Strategien zur Auflösung von überlappenden Flächen folgen in einem späteren Kapitel.

Zusätzlich gibt es noch das Argument der mathematischen Unschärfe bei Vektorberechnungen, sodass ein Rechteck, welches tatsächlich im komplexen Multipolygon liegt, (und zwar am Rand) den Containment Test nicht besteht.

Sollte der GeoXACML Prozessor auch in der Lage sein Koordinatensystemtransformationen durchzuführen, kann auch dies zu Performance Problemen führen. In diesem Beispiel ist offensichtlich, das Rechteck auf das Koordinatensystem des Europa Polygons/Multipolygons zu transformieren.

Als Regel sollte gelten: Ist die Anzahl der Stützpunkte der Geometrie im Request kleiner als die Anzahl der Stützpunkte der Geometrie in der Policy, so wird die Geometrie im Request transformiert. Das ganze gilt natürlich auch umgekehrt.

Hat nun aber die Geometrie in der Policy sehr viele Stützpunkte und der Request beinhaltet eine ähnliche komplexe Geometrie, so kommt man um eine Transformation vieler Stützpunkte nicht herum. Die ganze Situation kann man entschärfen, in dem man einfache Geometrien in der Policy verwendet.

Zusammenfassend:

Die Geometrie in der Policy sollte so einfach wie möglich sein, aber komplex genug um die Anforderungen der Zugriffskontrolle vernünftig abzudecken.

10 . Fallstudie: WMS GetMap Request

10.1 Ausgangsbasis

Untersucht werden nun die Möglichkeiten, einen WMS GetMap Request auf ein gewisses Gebiet einzuschränken. Es wird wieder die Karte mit den Kontinenten verwendet und die Einschränkung auf Europa mit Hilfe des einfachen Polygons.

Ausgangsbasis ist folgender XACML Request:

```
<Request>
 <Subject
  SubjectCategory="urn:oasis:names:tc:xacml:1.0:subject-category:access-subject">
  <Attribute AttributeId="urn:oasis:names:tc:xacml:2.0:subject:role"
    DataType="http://www.w3.org/2001/XMLSchema#anyURI">
    <AttributeValue>ROLE_AUTHENTICATED</AttributeValue>
 </Attribute>
 </Subject>
 <Resource>
  <Attribute AttributeId="org:geoserver:resource:type:ows:service"
      DataType="http://www.w3.org/2001/XMLSchema#string">
      <AttributeValue>WMS</AttributeValue>
  </Attribute>
  <AttributeValue AttributeId="org:geoserver:resource:type:param:bbox"
     DataType="urn:ogc:def:dataType:geoxacml:1.0:geometry">
    <gml:Polygon gid="Europe"
    xmlns:gml="http://www.opengis.net/gml"
          <gml:exterior>
           <gml:LinearRing>
            <gml:posList srsDimension="2">
             -130 24 -66 24 -66 50 -130 50 -130 24
            </gml:posList>
           </gml:LinearRing>
         </gml:exterior>
       </gml:Polygon>
  </AttributeValue>
 </Resource>
 <Action>
  <Attribute AttributeId="urn:oasis:names:tc:xacml:1.0:action:action-id"
   DataType="http://www.w3.org/2001/XMLSchema#string">
   <AttributeValue>GetMap</AttributeValue>
  </Attribute>
 </Action>
```

```
 <Environment>
 </Environment>
</Request>
```

Die Rolle ist also „ROLE_AUTHENTICATED“, die Aktion ist „GetMap“ und als Ressource wird der Servicename „WMS“ und die Bounding Box mitgegeben. (Koordinaten sind rein zufällig).

Der Übersichtlichkeit halber ist das <Target> Element für die Policies hier nur einmal angegeben und wird später nur mehr mittels Kommentar

```
<!-- GetMap Match -->
```

referenziert.

```
 <Target>
  <Subjects>
   <Subject>
    <SubjectMatch MatchId="urn:oasis:names:tc:xacml:1.0:function:string-equal">
     <AttributeValue DataType="http://www.w3.org/2001/XMLSchema#string">
       ROLE_AUTHENTICATED
     </AttributeValue>
     <SubjectAttributeDesignator DataType="http://www.w3.org/2001/XMLSchema#string"
           AttributeId="urn:oasis:names:tc:xacml:2.0:subject:role" />
    </SubjectMatch>
  <  /Subject>
  </Subjects>
  <Resources>
   <Resource>
    <ResourceMatch MatchId="urn:oasis:names:tc:xacml:1.0:function:string-equal">
     <AttributeValue DataType="http://www.w3.org/2001/XMLSchema#string">
       WMS
     </AttributeValue>
     <ResourceAttributeDesignator
           DataType="http://www.w3.org/2001/XMLSchema#string"
          AttributeId="org:geoserver:resource:type:ows:service" />
    </ResourceMatch>
  <  /Resource>
  </Resources>
  <Actions>
   <Action>
    <ActionMatch MatchId="urn:oasis:names:tc:xacml:1.0:function:string-equal">
     <AttributeValue DataType="http://www.w3.org/2001/XMLSchema#string">
```

```
          GetMap
        </AttributeValue>
        <ActionAttributeDesignator DataType="http://www.w3.org/2001/XMLSchema#string"
              AttributeId="urn:oasis:names:tc:xacml:1.0:action:action-id" />
      </ActionMatch>
  <  /Action>
  </Actions>
</Target>
```

Ein <Policy>,<PolicySet> oder <Rule> Element mit obigen <Target> ist also für WMS GetMap Requests der Rolle ROLE_AUTHENTICATED zuständig.

10.2 Fall 1, BBox im erlaubten Bereich

Folgendes Bild illustriert die Situation.

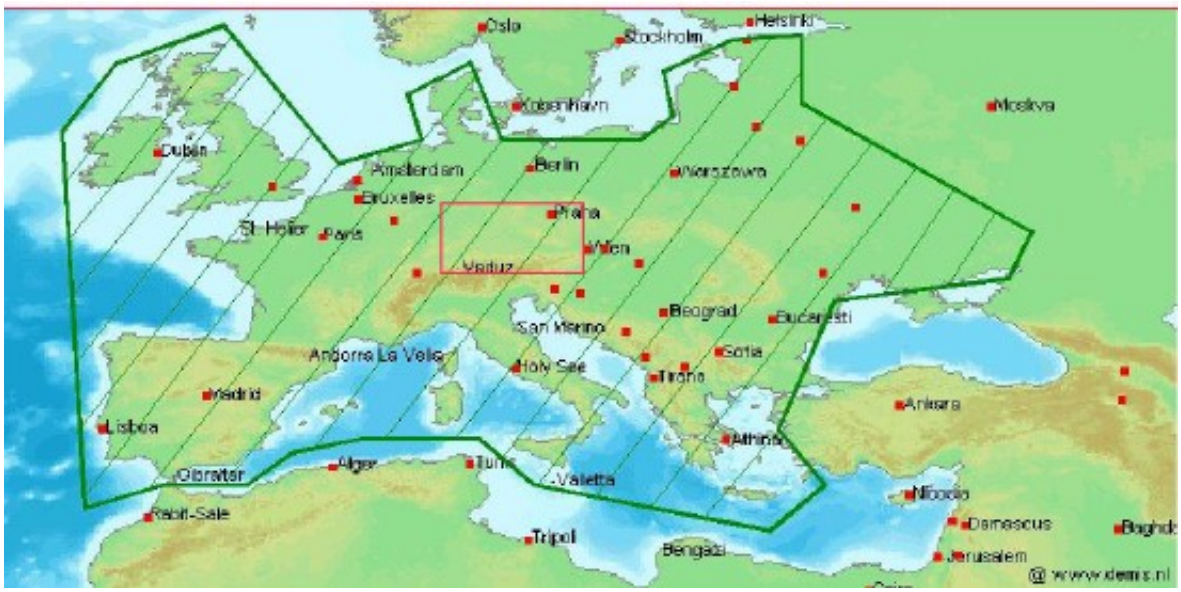


Abbildung 4: BBox im erlaubten Bereich

Dies ist die einfachste Situation. Die zugehörige Policy ist:

```
<?xml version="1.0" encoding="UTF-8"?>
<Policy xmlns="urn:oasis:names:tc:xacml:2.0:policy:schema:os"
         xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
        PolicyId="urn:oasis:names:tc:xacml:2.0:europe:contains"
        RuleCombiningAlgId=
            "urn:oasis:names:tc:xacml:1.0:rule-combining-algorithm:permit-overrides">

        <! -GetMap Match -->

    <Rule Effect="Permit" RuleId="urn:oasis:names:tc:xacml:2.0:rule:contains" >
   <Condition >
    <Apply FunctionId="urn:oasis:names:tc:xacml:1.0:function:all-of">
     <Function
          FunctionId="urn:ogc:def:function:geoxacml:1.0:geometry-contains"/>
     <AttributeValue DataType="urn:ogc:def:dataType:geoxacml:1.0:geometry">
      <gml:Polygon gid="Europe"
      xmlns:gml="http://www.opengis.net/gml"
          <gml:exterior>
           <gml:LinearRing>
            <gml:posList srsDimension="2">
             10 10 10 20 20 20 20 10 10 10
```

```
            </gml:posList>
          </gml:LinearRing>
        </gml:exterior>
      </gml:Polygon>
    </AttributeValue>
    <ResourceAttributeDesignator
        AttributeId="org:geoserver:resource:type:param:bbox"
        DataType="urn:ogc:def:dataType:geoxacml:1.0:geometry"/>
  </Apply>
 </Condition>
</Rule>
</Policy>
```

Anmerkung: Die verwendeten Koordinaten sind wiederum willkürlich. Obige Rule greift mittels des <ResourceAttributeDesignator> Elements auf die BBox im Request zu. Alle Typen von <*AttributeDesignator> Elementen (Resource,Environment,Action,Subect) liefern ja einen Bag zurück, in diesem Fall einen Bag von Geometrie Attributen.
Aus diesem Grund wird mittels einer High Order Bag Function

urn:oasis:names:tc:xacml:1.0:function:all-of

die Geometrie Funktion

"urn:ogc:def:function:geoxacml:1.0:geometry-contains"

auf alle Elemente im Bag angewendet, obwohl darin nur eine Geometrie enthalten ist.

Ist ein Containment gegeben, so evaluiert die Policy zu „Permit“, andernfalls to „NotApplicable“.

Hinweis:

Während man in einem <Condition> Element die Tatsache, dass ein <*AttributeDesignator> einen Bag zurückliefert, berücksichtigen muss, so entfällt dies im <Target>. Grund dafür ist dass im Target immer eine implizite High Order Bag Function angewendet wird, nämlich

urn:oasis:names:tc:xacml:1.0:function:any-of

10.3 Fall 2, BBox nicht im erlaubten Bereich

Illustration durch folgende Abbildung.

Abbildung 5: BBox nicht im erlaubten Bereich

Die entsprechende Policy ist leicht von Fall 1) abzuleiten.
Man ersetzt

"urn:ogc:def:function:geoxacml:1.0:geometry-contains"

durch

"urn:ogc:def:function:geoxacml:1.0:geometry-disjoint"

und ändert im <Rule> Element das Attribut *Effect* von „Permit“ auf „Deny“. Zusätzlich gibt man dem <Policy> und dem <Rule> Element noch einen anderen Namen mittels entsprechendem ID Attribut. Für diese Policy vergibt man analog zu Fall 1) die *PolicyId*

"urn:oasis:names:tc:xacml:2.0:europe:disjoint"

Diese neue Policy evaluiert zu DENY wenn die BBox außerhalb liegt, zu NOT_APPLICABLE in alle anderen Fällen.

Interessant ist hier noch, wie der PEP auf ein DENY reagiert. Grundsätzlich gibt es hier zwei Möglichkeiten:

1) Entsprechende Fehlermeldung zum Client
2) Ein leeres Image gefüllt mit Hintergrundfarbe, wie wenn keine Daten in dieser Region verfügbar wären.

10.4 Fall 3, BBox überlappt mit erlaubtem Bereich

Illustration durch folgende Abbildung.

Abbildung 6: BBox überlappt mit erlaubtem Bereich

Diese Situation bereitet die größten Probleme.

Ein PERMIT ist auf keinen Fall zielführend, da man dann praktisch überhaupt keine Zugriffskontrolle mehr hat. Die BBox braucht dann nur eine kleine Überschneidung mit dem erlaubten Bereich. Praktisch kann man dann eigentlich jede Region betrachten.

Ein DENY ist aber auch nicht gut möglich, dadurch ginge die Benutzbarkeit verloren. Man kann schwer verlangen, dass ein Benutzer darauf achtet, dass kein Teil seiner BBox aus dem erlaubten Bereich herausragt.

Das beste wäre, mit einem PERMIT zu antworten und dem PEP mitzuteilen, dass ein Clipping zu erfolgen hat. Bereiche außerhalb der erlaubten Region sind dann mit Hintergrundfarbe auszufüllen.

Die einzige Möglichkeit, dem PEP Zusatzinformation über die Entscheidung zu geben, ist das <Obligation> Element.

Eine entsprechende Policy:

```
<?xml version="1.0" encoding="UTF-8"?>
<Policy xmlns="urn:oasis:names:tc:xacml:2.0:policy:schema:os"
        xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
        PolicyId="urn:oasis:names:tc:xacml:2.0:europe:overlapps"
        RuleCombiningAlgId=
            "urn:oasis:names:tc:xacml:1.0:rule-combining-algorithm:permit-overrides">

        <! -GetMap Match -->

   <Rule Effect="Permit" RuleId="urn:oasis:names:tc:xacml:2.0:rule:contains" >
  </Rule>
  <Obligations>
     <Obligation FulfillOn="Permit" ObligationId="obligation:geometry:clip">
    <AttributeAssignment
      DataType="urn:ogc:def:dataType:geoxacml:1.0:geometry"
      AttributeId="clippGeom">
          <gml:Polygon gid="Europe" xmlns:gml="http://www.opengis.net/gml"
              <gml:exterior>
                <gml:LinearRing>
                  <gml:posList srsDimension="2">
                     10 10 10 20 20 20 20 10 10 10
                  </gml:posList>
                </gml:LinearRing>
              </gml:exterior>
            </gml:Polygon>
    </AttributeAssignment>
  </Obligation>
 </Obligations>
</Policy>
```

Obige Policy hat kein <Condition> Element in welchem eine topologische Beziehung überprüft wird. Das ist auch nicht notwendig, da die Fälle 1) und 2) eigene Policies haben und diese immer vor obiger Policy evaluiert werden. Diese Policy fordert jedenfalls den PEP zu einem Clipping mit der Geometrie aus dem <AttributeAssignment> Element auf.

Anmerkung: Die *ObligationId*

obligation:geometry:clip

und die *AttributeId*

clippGeom

müssen zwischen PEP und PDP abgestimmt sein.

Eine andere Möglichkeit wäre, nicht die GML Geometrie selbst in die Obligation zu kodieren, sondern einen Beschreibung, welche Geometrie der PEP zu verwenden hat.

```
<?xml version="1.0" encoding="UTF-8"?>
<Policy xmlns="urn:oasis:names:tc:xacml:2.0:policy:schema:os"
        xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
        PolicyId="urn:oasis:names:tc:xacml:2.0:europe:overlapps"
        RuleCombiningAlgId=
            "urn:oasis:names:tc:xacml:1.0:rule-combining-algorithm:permit-overrides">

        <! -GetMap Match -->

   <Rule Effect="Permit" RuleId="urn:oasis:names:tc:xacml:2.0:rule:contains" >
  </Rule>
  <Obligations>
     <Obligation FulfillOn="Permit" ObligationId="obligation:geometry:clip">
      <AttributeAssignment
        DataType="http://www.w3.org/2001/XMLSchema#string"
        AttributeId="Feature">
                Kontinente
      </AttributeAssignment>
      <AttributeAssignment
        DataType="http://www.w3.org/2001/XMLSchema#string"
        AttributeId="FeatureID">
                4711
      </AttributeAssignment>
  </Obligation>
 </Obligations>
</Policy>
```

Hier beinhaltet die <Obligation> zwei <AttributeAssignment> Elemente. Eines davon mit *AttributeId* „Feature“ und Wert „Kontinente“, das andere mit *AttributeId* „FeatureID“ und Wert „4711“

Der PEP wird somit veranlasst, die Geometrie mit der FeatureId 4711 aus dem Feature Kontinente als erlaubte Region zu verwenden.

10.5 Kombiniertes PolicySet

Um die Policies aus den Fällen 1-3 zu kombinieren, benötigt man ein entsprechendes PolicySet.

```
<?xml version="1.0" encoding="UTF-8"?>
<PolicySet xmlns="urn:oasis:names:tc:xacml:2.0:policy:schema:os"
        xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
        PolicySetId="urn:oasis:names:tc:xacml:2.0:gis:PolicySetSetWMSGetMap"
        PolicyCombiningAlgId=
        "urn:oasis:names:tc:xacml:1.0:policy-combining-algorithm:first-applicable">
        <Target></Target>
        <PolicyIdReference>
                urn:oasis:names:tc:xacml:2.0:europe:contains
        </PolicyIdReference>
        <PolicyIdReference>
                urn:oasis:names:tc:xacml:2.0:europe:disjoint
        </PolicyIdReference>
        <PolicyIdReference>
                urn:oasis:names:tc:xacml:2.0:europe:overlapps
        </PolicyIdReference>
</PolicySet>
```

Hier werden einfach die 3 Policies miteinander kombiniert. Man beachte dass der Policy Combining Algorithmus das erste PERMIT oder DENY als Ergebnis des PolicySets nimmt. Aus diesem Grund steht das <PolicyIdReference> Element für den überlappenden Fall (Fall 3) auch am Ende.

Optimierungen:

1) Die GML Kodierung der erlaubten Region wird mehrfach verwendet. Diese sollte zumindest als eigene XML Komponente ausgelagert werden und über XML Entity Referenzen eingebunden werden.
2) Das <Target> Element wird in jeder Policy evaluiert. Dies ist nicht notwendig und auch redundant. Man kann das <Target> Element einmal in dem PolicySet angeben, der Effekt ist der gleiche. Dann würden im Fall 1) nur ein Prädikat „contains“, im Fall 2) nur ein Prädikat „disjoint“ und im Fall 3) nur das <Obligation> Element überbleiben.
3) Unter Voraussetzung von 2) könnte man die Policy Referenzen auflösen und die 3 jetzt sehr kleinen Policies direkt im PolicySet einfügen.

11 . GeoXACML und CQL

CQL (Catalog Query Language) ist eine von der OGC genormte Abfragesprache. Die Syntax ist dabei an SQL angelehnt, allerdings erweitert um geographische Funktionalität.

Die Syntax findet man in der Catalog Service Spezifikation[9]. Es ist dadurch mögliche, bei entsprechenden OGC Service Requests auch einen Filter mitzugeben.

Ein Beispiel für einen CQL Ausdruck, „Polygon in WKT“ bedeutet ein Polygon in Well Known Text Darstellung.

```
(geometry INSIDE 'Polygon in WKT') AND (tiefe between 400 and 800)
```

Zusätzlich gibt es auch noch das OGC Filter Encoding[10]. Hier wird festgelegt, wie CQL Queries in XML darzustellen sind. Obiger CQL Ausdruck in XML:

```
<Filter>
 <And>
  <Within>
   <PropertyName>geometry</PropertyName>
   <gml:Polygon name="1" srsName="EPSG:63266405">
   ..<gml:outerBoundaryIs>
      <gml:LinearRing>
       <gml:posList>-98.5485,24.2633 ...</gml:posList>
      </gml:LinearRing>
     </gml:outerBoundaryIs>
   </gml:Polygon>
  </Within>
  <PropertyIsBetween>
   <PropertyName>tiefe</PropertyName>
   <LowerBoundary>
    <Literal>400</Literal>
   </LowerBoundary>
   <UpperBoundary>
    <Literal>800</Literal>
   </UpperBoundary>
  </PropertyIsBetween>
 </And>
</Filter>
```

Diese Filtermöglichkeiten kann man sich nun für die Zugriffskontrolle

zu Nutze machen. Voraussetzung hierfür ist, dass der PEP CQL und/oder das Filter Encoding handhaben kann, der PEP muss den Filter parsen und anwenden können.

Ein <Obligation> Element beinhaltet ja <AttributeAssignment> Elemente und gemäß der XML Schema Definition kann der Inhalt beliebig sein, also auch ein String. Dadurch hat eine Policy die Möglichkeit , OGC Filter an den PEP zu übermitteln.

Angewendet auf unser Fallbeispiel bieten sich hier 2 Möglichkeiten.

1) Für den der Überlappung (Fall 3) wird im <Obligation> Element anstatt einer Geometrie gleich ein Filter angegeben.
2) Man verzichtet komplett auf die GeoXACML Erweiterungen, hat nur eine Policy die immer zu „Permit“ evaluiert und ein <Obligation> Element mit dem Filter hat.

Auf jeden Fall eröffnet die Kombination von OGC Filtern und GeoXACML vielfältige Möglichkeiten.

12 . Zugriffskontrolle auf Basis von URLs

OWS Services können unter Verwendung von HTTP URLs genutzt werden. Hierbei unterscheidet man einen reinen HTTP GET Aufruf mit entsprechenden URL Parametern oder einem HTTP POST Aufruf, bei dem die Parameter XML kodiert im HTTP Body mitgesendet werden.

In einem J2EE Container kann man vor ein HttpServlet (welches den HTTP Request empfängt) sogenannte Servlet Filter schalten. Solche Filter sind der ideale Platz um Zugriffsberechtigungen im voraus zu prüfen. Der Entwickler des Filters hat Zugriff auf alle Informationen aus dem HTTP Request.

Die Idee ist nun, aus diesen Informationen einen entsprechenden XACML Request zu erzeugen und evaluieren zu lassen.

Als illustrierendes Beispiel ein konkreter WMS GetMap Request.

```
http://localhost:8080/geoserver/wms?bbox=-130,24,-66,50&styles=population&Format=image/png&request=GetMap&layers=topp:states&width=550&height=250&srs=EPSG:4326
```

Die Parameter wurden schon unter 6.1 ,Einleitendes Szenario, beschrieben. Ein daraus erstellter XACML Request könnte so aussehen.

```
<Request>
 <Subject
  SubjectCategory="urn:oasis:names:tc:xacml:1.0:subject-category:access-subject">
  <Attribute AttributeId="urn:oasis:names:tc:xacml:2.0:subject:role"
    DataType="http://www.w3.org/2001/XMLSchema#anyURI">
    <AttributeValue>ROLE_AUTHENTICATED</AttributeValue>
 </Attribute>
 </Subject>
 <Resource>
  <Attribute AttributeId="org:geoserver:resource:type:url:param:srs"
     DataType="http://www.w3.org/2001/XMLSchema#string">
     <AttributeValue>EPSG:4326</AttributeValue>
  </Attribute>
  <Attribute AttributeId="org:geoserver:resource:type:url:param:layers"
    DataType="http://www.w3.org/2001/XMLSchema#string">
    <AttributeValue>topp:states</AttributeValue>
  </Attribute>
  <Attribute AttributeId="org:geoserver:resource:type:url:param:styles"
```

```
        DataType="http://www.w3.org/2001/XMLSchema#string">
        <AttributeValue>population</AttributeValue>
    </Attribute>
    <Attribute AttributeId="org:geoserver:resource:type:url:param:request"
      DataType="http://www.w3.org/2001/XMLSchema#string">
     <AttributeValue>GetMap</AttributeValue>
    </Attribute>
    <Attribute AttributeId="org:geoserver:resource:type:url:param:bbox"
      DataType="http://www.w3.org/2001/XMLSchema#string">
      <AttributeValue>-130,24,-66,50</AttributeValue>
    </Attribute>
    <Attribute AttributeId="org:geoserver:resource:type:url:param:Format"
      DataType="http://www.w3.org/2001/XMLSchema#string">
      <AttributeValue>image/png</AttributeValue>
    </Attribute>
    <Attribute AttributeId="org:geoserver:resource:type:url:param:height"
      DataType="http://www.w3.org/2001/XMLSchema#string">
      <AttributeValue>250</AttributeValue>
    </Attribute>
    <Attribute AttributeId="org:geoserver:resource:type:url:param:width"
      DataType="http://www.w3.org/2001/XMLSchema#string">
      <AttributeValue>550</AttributeValue>
    </Attribute>
    <Attribute AttributeId="org:geoserver:resource:type:url"
      DataType="http://www.w3.org/2001/XMLSchema#string">
      <AttributeValue>/wms</AttributeValue>
    </Attribute>
  </Resource>
  <Action>
    <Attribute AttributeId="urn:oasis:names:tc:xacml:1.0:action:action-id"
      DataType="http://www.w3.org/2001/XMLSchema#string">
      <AttributeValue>GET</AttributeValue>
    </Attribute>
  </Action>
  <Environment>
    <Attribute AttributeId="org:geoserver:environment:IPAddress"
      DataType="urn:oasis:names:tc:xacml:2.0:data-type:ipAddress">
     <AttributeValue>127.0.0.1</AttributeValue>
    </Attribute>
    <Attribute AttributeId="org:geoserver:environment:DNSName"
      DataType="urn:oasis:names:tc:xacml:2.0:data-type:dnsName">
      <AttributeValue>localhost</AttributeValue>
    </Attribute>
  </Environment>
</Request>
```

Es wurde also der URL selbst und jeder URL Parameter als Resource Attribut kodiert, die HTTP Methode (GET) als Action Attribut und die IP Adresse bzw. der Hostname des Clients als Environment Attribute. Die Rolle ist ROLE_AUTHENTICATED

Es muss ja nicht jedes Attribut von den Policies berücksichtigt werden, besser man schickt mehr Information im XACML Request mit als zu wenig.

In den Policies selbst können nun beispielhaft folgende Fragen beantwortet werden.

a) Darf ein authentifizierter Benutzer einen WMS GetMap Request auf den Layer „topp:states“ ausführen ?

b) Kommt der Request von einer nicht autorisierten IP Adresse ?

c) Ist das zu erstellende Bild zu groß ? (*width* und *height*)

Im Sinne von GeoXACML ist es natürlich besser den *bbox* Parameter statt

```
<Attribute AttributeId="org:geoserver:resource:type:url:param:bbox"
  DataType="http://www.w3.org/2001/XMLSchema#string">
  <AttributeValue>-130,24,-66,50</AttributeValue>
</Attribute>
```

als GML Attribute zu kodieren

```
<AttributeValue AttributeId="org:geoserver:resource:type:url:param:bbox"
 DataType="urn:ogc:def:dataType:geoxacml:1.0:geometry">
<gml:Polygon gid="Europe"
 xmlns:gml="http://www.opengis.net/gml"
      <gml:exterior>
       <gml:LinearRing>
        <gml:posList srsDimension="2">
          -130 24 -66 24 -66 50 -130 50 -130 24
        </gml:posList>
       </gml:LinearRing>
     </gml:exterior>
   </gml:Polygon>
</AttributeValue>
```

Die Kodierung als Geometrie Attribut erlaubt die Anwendung der GeoXACML Funktionen !

Der Nachteil dieses Ansatzes ist, dass man

1) Im voraus die Parameter kennen muss, welche in eine Geometrieattribut konvertiert werden müssen.
2) Das Prinzip nur für HTTP Requests funktioniert.

Für den Einsatz dieser Methode in einem J2EE Container ist noch zu beachten, dass man die vorgegebene J2EE URL Zugriffskontrolle nicht verwenden sollte, andernfalls würde man ein und dieselbe Ressource (URLs) auf zwei verschiedene Arten schützen.

13 . Abgrenzung zu GeoDRM

DRM Systeme benutzen für die Zugriffskontrolle kryptografische Verfahren. Digitaler Inhalt wird durch Verschlüsselung an eine Lizenz (Benutzungsbestimmungen) gebunden. Der physikalische „Besitz“ von Daten ermöglicht nicht gleichzeitig seine Benutzung.

Bei GeoXACML erfolgt die Zugriffskontrolle einmalig beim Anfordern von Daten. Wird Zugriff gewährt, so erhält der Benutzer die Daten in Form eines Bildes oder Vektordaten. Die Daten selbst sind dann nicht mehr geschützt.

DRM bietet also eine persistente (ständige) Zugriffskontrolle. Besitz der Daten alleine macht sie nicht nutzbar. Natürlich benötigen DRM Systeme Verfahren zur Authentifizierung, Autorisierung muss je nach Lizenz des authentifizierten Benutzers(kann auch ein Gerät sein) durchgeführt werden.

GeoDRM ist zum Zeitpunkt dieser Arbeit noch kein offizieller OGC Standard. Das Referenzmodell findet man unter [11]

GeoDRM basiert, wie der Name schon zum Ausdruck bringt, auf klassischem DRM. Hinzugefügt wurden Anforderungen betreffend geographischer Daten.

1) Lizenzen und Sublizenzen.
   Lizenzvereinbarungen werden für B2B (Business to Business) und B2C (Business to Consumer) Beziehungen benötigt. Hinzugefügt wurde die Möglichkeit, das auch ein Lizenznehmer wiederum die Möglichkeit hat, gewisse Lizenzen weiter zu geben.
2) Zugriffskontrolle für dynamisch generierte Daten. Während klassisches DRM statische Daten schützt (Musik, Video,...), bietet GeoDRM auch Zugriffskontrolle für OGC Web Services. Bezeichnet wird dies im Referenz Modell als „Parameter Range Conditions“. Im Prinzip entspricht dies einer Zugriffskontrolle auf URL Basis speziell für OGC Web Services.
3) DRM hat eine formale „Rights Language“, eine maschinell zu verarbeitende Sprache um Berechtigungen auszudrücken. Diese muss um geographische Features erweitert werden, analog zu GeoXACML und XACML.

14 . Erkenntnisse aus der Arbeit

XACML selbst ist bereits ein sehr mächtiges Konzept. Das bedeutet natürlich auch, dass die Handhabung nicht trivial ist. Die Zusatzspezifikation GeoXACML an sich ist nicht sehr schwierig zu verstehen, wenn man mit dem GIS Bereich vertraut ist. Es wird lediglich ein Datentyp Geometrie hinzugefügt und eine Menge von bereits bekannten Funktionen.

Nochmals wird hier auf das Kapitel GeoXACML Performance verwiesen, hier besteht wirklich die Gefahr, ein Zugriffskontrollsystem zu implementieren, welches zu rechenintensiv ist.

Ganz besonders nützlich im GIS Umfeld ist die Möglichkeit, mit <Obligation> Elementen zu arbeiten. Dieses Konstrukt in Verbindung mit OGC Filtern ist wirklich ein äußerst mächtige Möglichkeit, komplexe GIS Zugriffskontrolle zu implementieren.

Diese Variante wird auch Einzug in das OS Projekt www.geoserver.org halten. Die komplette Zugriffskontrolle wird auf GeoXACML unter Verwendung der Ergebnisse dieser Arbeit umgestellt. Anfragen an das XACML System werden dann nicht mehr mit JA/Nein beantwortet, sondern mit einem Filter. (JA/Nein ausgedrückt durch spezielle Filter „IncludeAll“ und „ExcludeAll“).

Bezüglich GeoDRM sind meiner Meinung Vorbehalte angebracht. (Das gilt natürlich prinzipiell für alle DRM Systeme). Wie die Erfahrungen zeigen, gibt es hier wirklich Probleme, besonders störend ist die enge Bindung an einen Lieferanten wie es z. Bsp. im Office Bereich schon seit langer Zeit der Fall ist. (Nicht wegen DRM, aber das Prinzip ist dasselbe)

Interessant wird auch noch die Art und Weise, wie man ein solch komplexes Zugriffssystem administrierbar macht, die XACML Policies von Hand zu erstellen ist kein gangbarer Weg. Aber auch das wird sich im Rahmen des OS Projektes noch zeigen.

15. Literaturverzeichnis

1 INSPIRE
http://inspire.jrc.ec.europa.eu/

2 Open Geospatial Consortium
http://www.opengeospatial.org/

3: Organisation for the Advancment of Structured Information Standards
http://www.oasis-open.org/home/index.php

4 Multiple Resource Profile
http://docs.oasis-open.org/xacml/2.0/access_control-xacml-*2.0-mult-profile-spec-os.pdf*

5 XACML Profile for Role Based Access Control
http://docs.oasis-open.org/xacml/cd-xacml-rbac-profile-01.pdf

6 GeoXACML
http://www.opengeospatial.org/standards/geoxacml

7 OpenGIS Simple Feature Access
http://portal.opengeospatial.org/files/?artifact_id=18241

8 GeoXACML und SAML
http://gdi.berlin-brandenburg.de/papers/ws04_04_geoxacml_und_saml.pdf

9 OGC Catalog Services
http://www.opengeospatial.org/standards/cat

10 OGC Filter Encoding
http://www.opengeospatial.org/standards/filter

11 GeoDRM,
http://www.opengeospatial.org/standards/as/geodrmrm