Sicherheitsmechanismen in serviceorientierten Architekturen
Hauptseminar im SS 2009

XACML

OASIS Profile in Bezug auf XACML

Philip Daubmeier
Technische Universität München

29.05.2009

Zusammenfassung

Das OASIS-Konsortium hat für die eXtensible Access Control Markup Language (kurz: XACML) Standards für Profile festgelegt, um die Zugriffskontrolle um einige Funktionen zu erweitern. Zum Einstieg wird das generelle Konzept von XACML kurz vorgestellt. In der folgenden Arbeit werden dann folgende drei Profile näher erläutert und anhand von Beispielen beschrieben: RBAC profile, multiple ressource profile und das hierarchical ressource profile.

Inhaltsverzeichnis

1 Was ist XACML?

Die eXtensible Access Control Markup Language (kurz: XACML) Version 2.0 wurde 2005 vom OASIS (Organization for the Advancement of Structured Information Standards)-Konsortium als Standard festgelegt. Dieser definiert Richtlinien und Regeln zur Kontrolle über Ressourcen, deren Zugriff überwacht werden soll ([12]).

XACML ist eine auf XML (eXtensible Markup Language) basierende Sprache, die nicht nur ein XML-Schema für `Security Policies` enthält, sondern auch für Anfragen nach einer Zugriffserlaubnis und deren Beantwortung. Möchte jemand beispielsweise auf eine geschützte Ressource zugreifen, so richtet er seine Anfrage an den entsprechenden Service (z.B. SOAP Webservice) ([4], Seite 428). Daraufhin wird das Zugriffskontrollsystem aktiv, überprüft die Anfrage mithilfe von vorher definierten Regeln (`Policy`) und gewährt oder verweigert den Zugriff auf den Service. Möglich ist auch eine Filterung der Anfrage (Preprocessing), oder eine Filterung der vom Service gelieferten Daten (Postprocessing).

XACML zeichnet sich durch eine sehr hohe Flexibilität und Interoperabilität aus [9]. Als sehr universell definierte Sprache setzt es keine verbindliche Implementierung fest. Jedoch wird eine generelle Architektur des Zugriffskontollsystems empfohlen, die im Folgenden erläutert wird.

1.1 Die Architektur des Zugriffskontrollsystems

Das folgende Modell veranschaulicht den im XACML Standard [6] empfohlenen Aufbau des Zugriffskontrollsystems. Tatsächliche Implementierungen (etwa [3]) müssen sich nicht zwangsläufig an dieses Modell halten, es bietet jedoch eine gute Vorstellung, wie das System realisiert werden kann.

1.1.1 Zugriffskontrolle

Abbildung 1: Ein Benutzer will auf einen Service zugreifen der durch ein Zugriffskontrollsystem überwacht wird.

Das Zugriffskontrollsystem setzt genau in der Mitte zwischen den Benutzern und dem von diesen angesprochenen Service an (siehe Abbildung 1). Es ist notwendig, dass der Benutzer seine Identität vor dem Zugriff bei einem Authentifizierungsdienst verifiziert. Das Zugriffskontrollsystem kann damit die Identität sicherstellen. Diese Authentifizierung ist nicht Teil des Zugriffskontrollsystems und deshalb auch nicht Teil von XACML.

1.1.2 PEP und PDP

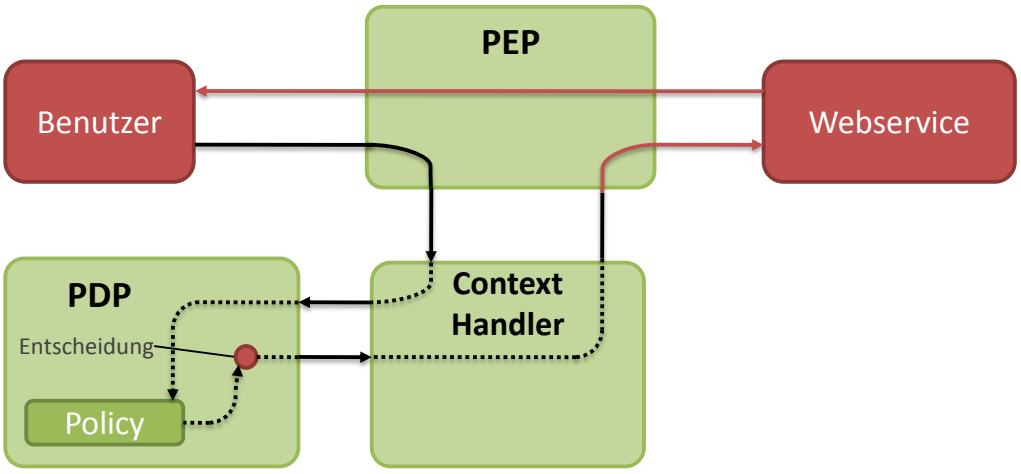

Abbildung 2: Das Zugriffskontrollsystem besteht unter anderem aus dem PEP und PDP.

Das Zugriffskontrollsystem ist nun seinerseits aus mehreren Bestandteilen aufgebaut, wie in Abbildung 2 dargestellt:

- Der *Policy Enforcement Point* (kurz PEP) sorgt für die Durchsetzung der Kontrolle für jede Zugriffsanfrage. Er fängt Anfragen an den Service zuerst ab und lässt sie erst passieren wenn sichergestellt ist, dass sie regelkonform sind. Es ist auch möglich, dass der PEP - je nach Implementierung - die Anfragen oder sogar die Antworten des Services zuerst filtert bevor sie zum Benutzer zurückgeschickt werden (Pre- oder Postprocessing). PEP

- Der *Policy Decision Point* (kurz PDP) errechnet für jede Anfrage mithilfe der darauf anwendbaren Regeln eine Entscheidung (zulassen oder verweigern). Seine Antwort wird an den PEP weitergegeben, der diese Entscheidung dann durchsetzt. ([1], Seite 61-62). PDP

- Der *Context Handler* arbeitet eng mit dem PDP zusammen. In den vielen Fällen werden beide Module auf demselben Rechner ausgeführt. Er bereitet `Decision Requests` mit Kontext Informationen auf oder formt mehrdeutige Daten in eindeutige Versionen um. Er hilft, dass dem PDP Anfragen zugestellt werden, die er auch konsistent beantworten kann. Er ist ein Vermittler zwischen mehreren Modulen, dem PEP, PDP, dem später erwähnten PIP und in manchen Fällen sogar dem Webservice selbst.

1.1.3 PIP

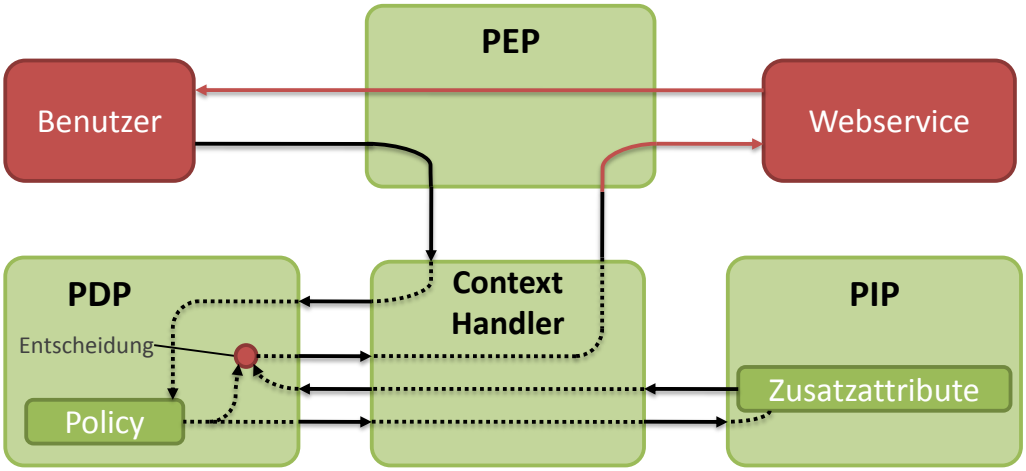

Abbildung 3: PIP

Der *Policy Information Point* (kurz PIP) ergänzt die Zugriffskontrolle um Zusatzattribute, die bei der Zugriffsentscheidung helfen. Er verwaltet diese und stellt diese bereit. Mit diesem letzten Modul ist die Kette der Aktionen vollständig, die für eine Entscheidung durchlaufen werden müssen (siehe Datenfluss in Abbildung 3): PIP

1. Der PEP fängt die Anfrage des Benutzers ab und verpackt sie in eine decision request.
2. Über den Context Handler gelangt die decision request an den PDP. Werden keine Zusatzinformationen mehr benötigt, geht es direkt weiter zu Schritt 4.
3. Zusatzattribute werden über den Context Handler vom PIP abgefragt.
4. Die Entscheidung wird anhand des Regelwerks (policy) getroffen und an den PEP zurückgegeben.
5. Der PEP leitet nun je nach Entscheidung die Anfrage an den Service, und dessen Antwort an den Benutzer zurück, oder gibt dem Benutzer eine Meldung über den nicht gewährten Zugriffsversuch aus.

1.1.4 Pre- und Postprocessing

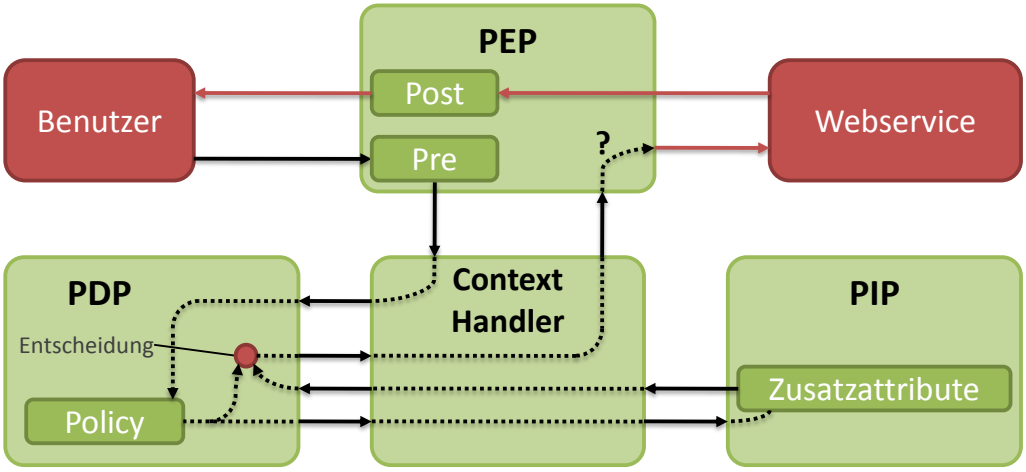

Abbildung 4: PEP mit sowohl Pre- als auch Postprocessing

Im PEP lassen sich Zugriffskontrollentscheidungen je nach Implementierung entweder vor, nach, oder in der Kombination von beidem durchsetzen. Dies nennt man auch Pre- und Postprocessing. In Abbildung 4 ist ein PEP mit beiden Modulen dargestellt.

Beim Preprocessing werden Zugriffsentscheidungen getroffen bevor die Anfrage an den Webservice geschickt wird. Bei einer negativen Entscheidung kann die Anfrage des Benutzers komplett abgelehnt werden. Wahlweise können vom PEP aber auch feingranularere decision requests an den PDP gestellt werden und die Webservice Anfrage daraufhin so modifiziert werden, dass der Webservice nur noch erlaubte Daten zurückliefert.

Bei Postprocessing wurde die Anfrage bereits vom Webservice verarbeitet. Die Antwort des Services wird nun wiederum vom PEP abgefangen und in eine decision request verpackt. Der PDP überprüft nun anhand seiner Regeln ob der Benutzer wirklich alle diese vom Service ausgelieferten Daten sehen darf. Bei einer negativen Entscheidung können die Daten noch nachträglich vom PEP gefiltert werden. Dann werden die Daten schließlich an den Benutzer ausgeliefert.

1.1.5 PAP

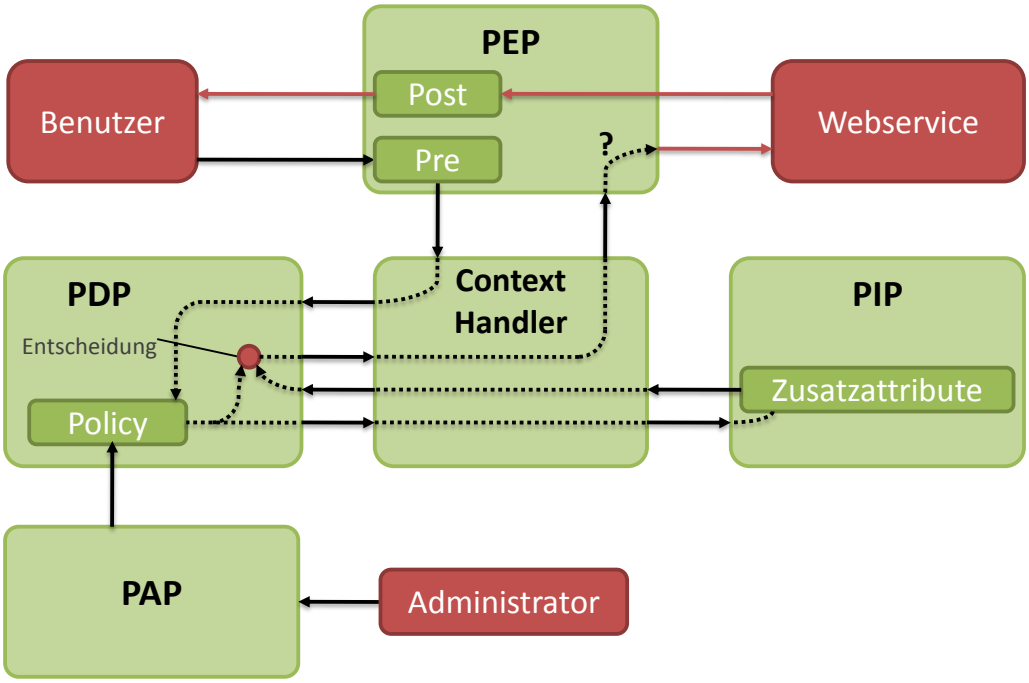

Abbildung 5: PAP

Administratoren des Systems melden sich etwa durch ein Single Sign-On Verfahren am Policy Administration Point (kurz PAP, Abb. 5) an. Dort haben sie die Möglichkeit Regeln zu erstellen oder zu verwalten. Eine weitere wichtige Aufgabe des PAP ist auch die Überwachung des Regelwerks, indem er zum Beispiel auf Konflikte und Inkonsistenzen zwischen Regeln hinweist. PAP
Es können zusätzlich Programme und Editoren entworfen werden, die auf dem PAP aufsetzen und den Administratoren eine Oberfläche bieten, auf der sie die Policies auf eine übersichtliche und komfortable Art und Weise warten können.

1.2 Das Sprachmodell

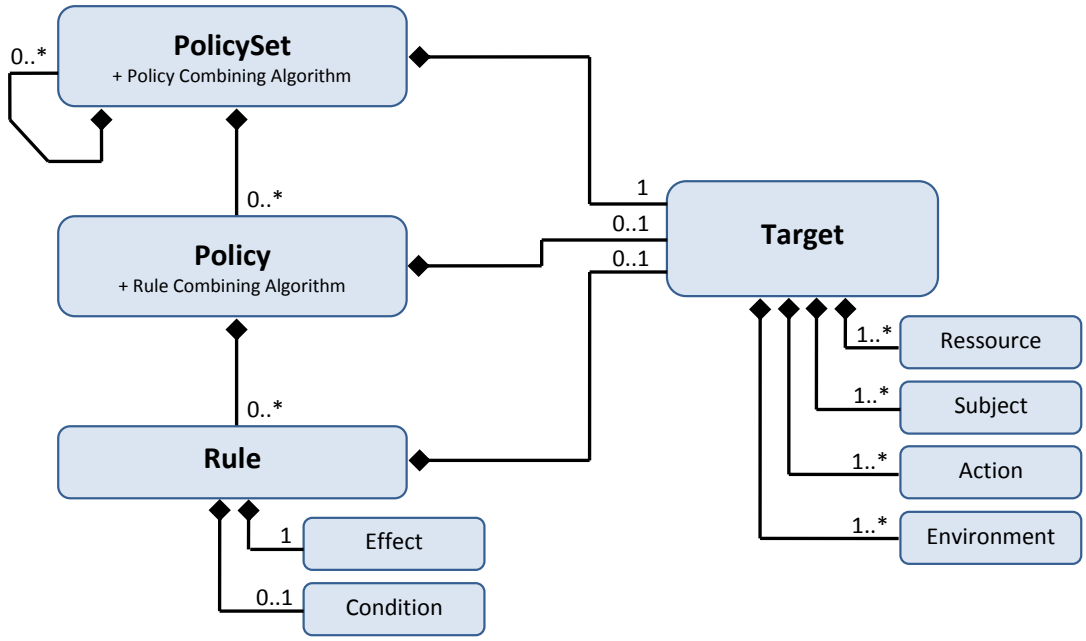

Abbildung 6: XACML Policy Sprachmodell (nach [6], Seite 19)

1.2.1 Die Grundstruktur einer Policy

Eine `Rule` stellt in XACML eine einzelne Regel dar. Eine oder mehrere dieser Regeln werden zu einer `Policy` zusammengefasst. Eine oder mehrere `Policies` werden wiederum in einem `PolicySet` vereint (siehe Abbildung 6). Somit entsteht ein kaskadierender Aufbau der einzelnen Regeln. Ein `Combining Algorithm` gibt dabei jeweils an wie die einzelnen untergeordneten Regeln oder Policies nach der Auswertung miteinander verrechnet werden sollen. In den meisten Fällen wird der Algorithmus `Deny-Overrides` zum Einsatz kommen. Dieser gibt an, dass bei dem Auftreten nur eines `Deny` in der untergeordneten Rule oder Policy, auch das übergeordnete Element `Deny` zurückgibt. Weitere `Combining Algorithms` sind in [6] ausführlich beschrieben.

1.2.2 Aufbau einer Regel

Die Komponenten einer Regel sind das `Target` (dt. Ziel), die `Condition` (dt. Bedingung) und der `Effect` (dt. Wirkung).

- Das `Target` besteht aus den Teilen `Resource`, `Subject`, `Action` und `Environment`, und beschreibt damit auf welche Ressource mit welchem Benutzer und mit welcher Aktion die Regel zutrifft. Ein Beispiel wäre das Tripel („Krankenakte“, „Arzt“, „lesen“). Das Zusatzelement `Environment` kann über Umgebungsinformationen verfügen die zur Entscheidung der Regel wichtig sind, etwa die Uhrzeit oder Ähnliches.
- Die `Condition` beschreibt mit einem booleschen Ausdruck die Bedingung zu der die Regel greift.
- Der `Effect` beschreibt den Rückgabewert einer Regel nach deren positiven Auswertung. Wenn die Regel anwendbar war, also die `Condition` Wahr ist, ist der Rückgabewert der im `Effect` angegebene. Dies kann entweder `Permit` (dt. zulassen) oder `Deny` (dt. verweigern) sein. Ließ sich die Regel nicht anwenden wird sie entsprechend zu `Not Applicable` ausgewertet.

Beispiele für solche Regeln und Policies werden im Folgenden im Zusammenhang mit den einzelnen Profilen gegeben.

2 XACML RBAC profile

RBAC

Role-based access control (kurz: RBAC) definiert eine Semantik mit der jedem Benutzer eine oder mehrere Rollen zugewiesen werden können. Der Vorteil hiervon liegt auf der Hand: Regeln müssen nicht länger für jeden Benutzer einzeln angelegt werden, sondern nur noch für jede Rolle. Bei einer Zugriffsentscheidung wird dann an Hand der Regel die auf die Rolle des Benutzers trifft entschieden. Außerdem können Rollen hierarchisch aufeinander aufgebaut werden. So kann etwa die Rolle „Professor“ die Regeln der Rolle „Lehrstuhlangehöriger“ erben und um zusätzliche eigene erweitern.

Möglich ist auch die Delegation von Rechten für bestimmte Zeiträume. So kann etwa der Professor seine Rechte für eine festgelegte Zeit einem Mitarbeiter seines Lehrstuhles übertragen, während er im Urlaub ist. Dazu wird eine Regel festgelegt die dem Mitarbeiter die Rolle „Professor“ zuweist, jedoch gekoppelt mit der Bedingung, dass sich das aktuelle Datum innerhalb der entsprechend festgelegten Zeitspanne befindet.

RBAC ist für die Anwendung in XACML in dem Profil [5] festgelegt. Dort ist vorgesehen, dass folgende drei Policys für die Realisierung bereitstehen:

- *Role assignment policy:* Gibt an welche Rollen dem Benutzer zugewiesen sind. Dort können auch Bedingungen angegeben sein, etwa wie im vorherigen Beispiel der Delegation der Rolle des Professors auf seinen Mitarbeiter: eine Zeitangabe.
- *Role policy set:* Legt fest welches Permission Policy Set zu einer Rolle gehört.
- *Permission policy set:* Legt fest welche Berechtigungen die entsprechende Rolle hat.

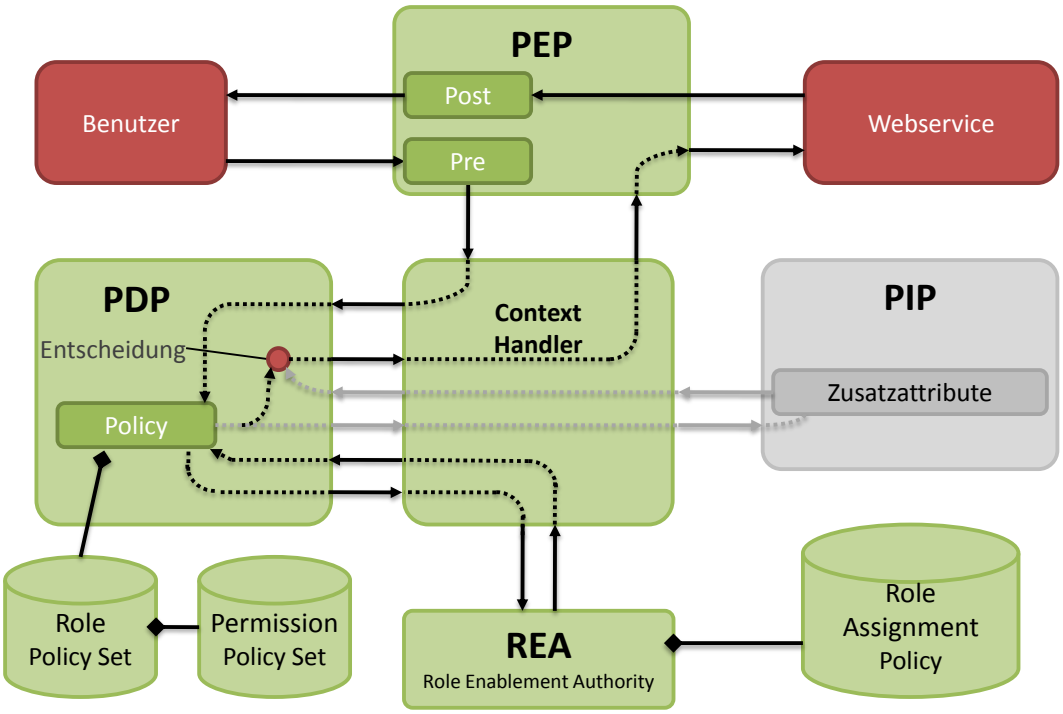

Abbildung 7: RBAC profile in der XACML Architektur

In der Architektur des Zugriffskontrollsystems sieht dies wie in Abbildung 7 dargestellt aus: Die frühere Policy wird in die zwei Bestandteile `Role PolicySet` und `Permission PolicySet` aufgeteilt. Aus diesen beiden `PolicySets` kann der PDP somit die Berechtigungen der entsprechenden Rollen eines Benutzers lesen. Die Role Assignment Policy hingegen wird von einer neuen Einheit verwaltet, der Role Enablement Authority (kurz: REA). Der PDP kann bei der REA anfragen ob ein Benutzer eine bestimmte angegebene Rolle zugewiesen hat, oder sich dort eine Auflistung aller zugewiesenen Rollen zu diesem Benutzer zurückgeben lassen.

REA

Die Gründe für die Auslagerung dieser Rollenzuweisungen in die REA sind:

- Rollen sind üblicherweise eng mit den Benutzern verbunden. Ein Server der eine Benutzerdatenbank verwaltet kann somit sinnvollerweise auch die Rollen und deren Beziehungen zu den Benutzern verwalten. Möglicherweise kann dieser Server auch gleichzeitig als Authentification Authority genutzt werden, um den Benutzer vor der Zugriffskontrolle zu authentisieren. Der PEP kann dann von diesem Server gleichzeitig die bestätigte Identität und die ihm zugewiesenen Rollen beziehen.

- Mit dem REA als logische und möglicherweise physisch unabhängige Einheit kann von mehreren Services und deren entsprechenden Zugriffskontrollsystemen genutzt werden. Dies vermindert den Verwaltungsaufwand drastisch, da Benutzer und Rollen zentral für alle Dienste verwaltet werden können und nicht für jedes Kontrollsystem einzeln. Damit werden auch Inkonsistenzen, die bei der Verwendung mehrerer separater REAs entstehen können, verhindert.

2.1 Role Assignment Policy

Das folgende vereinfachte Beispiel zeigt Anhand des Benutzers „Max“, der die Rolle „Manager“ besitzt, wie eine Richtlinie im Sinne des RBAC Profils aussehen kann. Dabei besitzt die Rolle „Manager“ die Rechte einen Vertrag zu unterschreiben, sowie alle Rechte die der Rolle „Angestellter“ zugewiesen sind. Um dem Benutzer „Max“ nun die Rolle „Manager“ zuzuweisen wird folgende Regel in der `Role Assignment Policy` festgelegt:

```
<Rule RuleId="manager:role:assignment" Effect="Permit">
  <Target>
    <Subjects>
      <Subject>
        <SubjectMatch MatchId="&function;string-equal">
          <AttributeValue>Max</AttributeValue>
          <SubjectAttributeDesignator AttributeId="&subject;subject-id"/>
        </SubjectMatch>
      </Subject>
    </Subjects>
    <Resources>
      <Resource>
        <ResourceMatch MatchId="&function;anyURI-equal">
          <AttributeValue>&roles;manager</AttributeValue>
          <ResourceAttributeDesignator AttributeId="&role;"/>
        </ResourceMatch>
      </Resource>
    </Resources>
    <Actions>
      <Action>
        <ActionMatch MatchId="&function;anyURI-equal">
          <AttributeValue>&actions;enableRole</AttributeValue>
          <ActionAttributeDesignator AttributeId="&action;action-id"/>
```

```
        </ActionMatch>
      </Action>
    </Actions>
  </Target>
</Rule>
```

Damit kann die REA auf die Anfrage „Subject:Max; Action:enableRole; Ressource:role manager“ mit `Permit` antworten, also einer Bestätigung, dass „Max“ die Rolle „Manager“ zugewiesen hat. Die Regel überprüft dazu, ob die mit den `AttributeDesignators` aus der Anfrage gelesenen Werte mit den Werten aus der Regel übereinstimmen. Optional können einer solchen Regel auch Bedingungen in Form eines `Condition` Tags angefügt werden, etwa eine Zeitspanne in der der Benutzer diese Rolle ausüben darf.

2.2 Permission PolicySet

Das `Permission PolicySet`, das im Folgenden vereinfacht dargestellt ist, gibt in diesem Beispiel nun an, dass jeder Rolle das Unterschreiben von Verträgen gestattet ist, sofern dieser Rolle das folgende `PolicySet` zugeordnet ist.

```
<PolicySet xmlns="urn:...:policy:schema:os" PolicySetId="PPS:manager:role"
  PolicyCombiningAlgId="&policy-combine;permit-overrides">
  <Policy PolicyId="Permissions:manager:role"
    RuleCombiningAlgId="&rule-combine;permit-overrides">
    <Rule RuleId="Permission:to:sign:a:purchase:order" Effect="Permit">
      <Target>
        <Resources>
          <Resource>
            <ResourceMatch MatchId="&function;string-equal">
              <AttributeValue>contract</AttributeValue>
              <ResourceAttributeDesignator AttributeId="&resource;resource-id"/>
            </ResourceMatch>
          </Resource>
        </Resources>
        <Actions>
          <Action>
            <ActionMatch MatchId="&function;string-equal">
              <AttributeValue>sign</AttributeValue>
              <ActionAttributeDesignator AttributeId="&action;action-id"/>
            </ActionMatch>
```

```
            </Action>
          </Actions>
        </Target>
      </Rule>
    </Policy>
    <!-- Manager hat auch alle Rechte eines Angestellten -->
    <PolicySetIdReference>PPS:employee:role</PolicySetIdReference>
</PolicySet>
```

Diese Regel bezieht sich, im Gegensatz zur `Role Assignment Policy`, nicht auf eine Anfrage des PDP an die REA, sondern auf eine Anfrage des PEP die der PDP nun validieren muss (siehe Aufteilung der drei Policies in Abbildung 7). Die Regel erlaubt das Unterschreiben von Verträgen, indem sich die Regel zu `Permit` auswertet wenn in der Anfrage die `resource-id` auf „contract“ und die `action-id` auf „sign“ zeigt.
Außerdem referenziert eine `PolicySetIdReference` das hier nicht aufgeführte `PolicySet` „PPS:employee:role“. Dieses ist analog zu diesem `Permission PolicySet` aufgebaut, bis auf den Wert „create“ statt „sign“ als `Action`. Damit erlaubt es Angestellten einen Vertrag zu erstellen, doch nur Managern einen Vertrag zu erstellen und auch zu Unterschreiben.
Das `Permission PolicySet` enthält also die tatsächlich zugriffsrelevanten Ge- und Verbote, genau wie eine Policy bei einem XAMCL basierten System ohne RBAC Profil und kann auch alle denkbaren Bedingungen und Regeln enthalten wie ein solches. Auch weitere Profile wie das hierarchical oder das multiple ressource profile, die auch in dieser Arbeit vorgestellt werden. Diese können hier verwendet werden, solange der PEP diese Profile für seine Anfragen unterstützt.

2.3 Role Policy Set

Das Fehlende Bindeglied zwischen der `Role Assignment Policy` und dem `Permission PolicySet` stellt das `Role PolicySet` dar. Es weist einer Rolle, nachdem diese von der REA bestätigt wurde, alle entsprechenden `Permission PolicySets` zu. Im folgenden Beispiel wird dazu mit einer `PolicySetIdReference` auf das entsprechende `Permission PolicySet` „PPS:manager:role“ verwiesen, wenn die Rolle gleich „Manager“ ist.

```
<PolicySet xmlns="urn:...:policy:schema:os" PolicySetId="RPS:manager:role"
  PolicyCombiningAlgId="&policy-combine;permit-overrides">
  <Target>
    <Subjects>
      <Subject>
        <SubjectMatch MatchId="&function;anyURI-equal">
          <AttributeValue>&roles;manager</AttributeValue>
          <SubjectAttributeDesignator AttributeId="&role;"/>
        </SubjectMatch>
      </Subject>
    </Subjects>
  </Target>
  <!-- Weise der Rolle "Manager" die Rechte aus dem Manager PPS zu  -->
  <PolicySetIdReference>PPS:manager:role</PolicySetIdReference>
</PolicySet>
```

3 XACML hierarchical resource profile

3.1 Funktion

Das hierarchical resource profile [7] beschreibt eine Möglichkeit einfache Regeln für Ressourcen mit hierarchischer Abhängigkeit zu schreiben. Viele Webservices, die mit einer Zugriffskontrolle beschränkt werden, geben Daten zurück, bei denen erst im Zusammenhang mit den hierarchischen Nachfolgern oder Vorgängern eines Knotens entschieden werden kann, ob der Zugriff erlaubt oder verboten werden soll.

Ein Beispiel hierfür ist ein Webservice, der Krankenakten eines Krankenhauses ausliefert. In diesem Szenario könnte ein Patient etwa einen speziellen Eintrag in seiner Krankenakte anfragen. Das Zugriffskontrollsystem muss dann eine Möglichkeit besitzen, festzustellen ob dieser Eintrag zu der Krankenakte des Patienten selbst gehört oder nicht. Daher braucht es Informationen über den Vaterknoten dieses Eintrags, wenn dort der Name des Patienten eingetragen ist.

Das hierarchical resource profile ist in seinem Standard [7] für alle Formen hierarchischer Daten ausgelegt, also nicht nur für strenge Baumstrukturen, sondern DAGs (Directed Acyclic Graphs [11]) allgemein. Linux Dateisysteme besitzen in der Regel keine strenge Baumstruktur, da sie Hard- und Softlink enthalten können. Um diese Strukturen mit diesem Profil verwenden zu können, muss der Context Handler Kenntnis von der Hierarchie haben und Pfade mit enthaltenen Links zu eindeutigen Pfaden umwandeln. Regeln in der Policy könnten andernfalls durch geschickte Ausnutzung von Links umgangen werden und somit im schlimmsten Fall das Kontrollsystem aushebeln.

Im Folgenden werden jedoch nur strenge Hierarchien, also Bäume mit genau einer Wurzel und ohne Querverbindungen, behandelt. Jedes valide XML Dokument erfüllt genau diese Bedingung, und Webservice Anfragen und Antworten sind oft in genau dieser Form (etwa SOAP Webservices [2]).

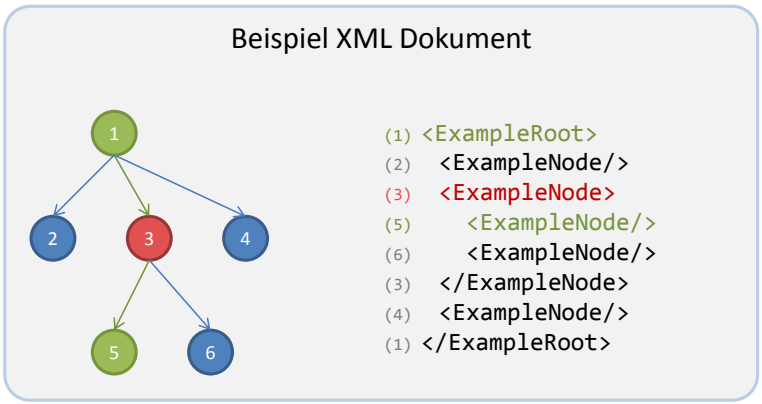

Abbildung 8: Eine beispielhafte Struktur einer Webservice Antwort

Abbildung 8 veranschaulicht einen beispielhaften Aufbau eines solchen XML Dokuments, das etwa die Antwort eines Webservice auf eine Anfrage darstellen könnte. Die Aufgabe des PEP in diesem Profil ist nun zusätzlich zu seinen bisherigen Aufgaben (wie oben in 1.1.2 beschrieben) das Hinzufügen folgender Attribute in die Anfrage an den PDP:

- Ein Attribut `resource-id`, das einen XPath (in 3.2 erläutert) Ausdruck enthält, der genau den Knotens auf die sich die Anfrage bezieht beschreibt. Dies stellt also den Knoten dar, für den entschieden werden soll, ob der Benutzer diesen sehen darf oder nicht.
- Ein Attribut `resource-parent`, mit einem XPath Ausdruck zum Vaterknoten.
- Ein Attribut `resource-ancestor`, das einen XPath Ausdruck enthält der zu dem Nachfolgerknoten in der Hierarchie zeigt.

Diese Attribute können nun in den Regeln der Policy verwendet werden, und machen es damit möglich Forderungen an die Vorgänger und Nachfolger zu stellen, die schnell vom PDP ausgewertet werden können.

3.2 XPath

XPath ist eine Abfragesprache, um einen Tag in einem XML Dokument zu adressieren, ähnlich einem Dateipfad im Dateisystem. XPath Ausdrücke werden im hierarchical resource profile benötigt um einen Basisknoten, sowie dessen Vater- und Kindknoten

im `ResourceContent` zu referenzieren. In den folgenden XACML Beispielen wird lediglich eine XPath Funktion verwendet: Der direkte Pfad zu dem entsprechenden Knoten. Dazu wird der Wurzelknoten und die nacheinander folgenden Knoten der Hierarchie mit Schrägstrichen (/) voneinander getrennt. Die Zahl in den Klammern gibt dabei die Nummer des jeweiligen Knoten an. In einem XML Dokument mit der Wurzel `Root` und deren 2 Kindknoten `Leaf`, wird der zweite Kindknoten folgendermaßen adressiert: `/Root[1]/Leaf[2]`. Alle weiteren Funktionen von XPath wie Prädikate, Vereinigung und mehr sind in [10] beschrieben, und sollen hier nicht näher erläutert werden, da sie in den Beispielen keine Verwendung finden.

3.3 Beispiel

Wie nun eine solche Anfrage des PEP aussehen kann wird anhand des Beispieldokuments aus Abbildung 8 hier verdeutlicht:

```
<Request>
  <Action><!-- Eine Action --></Action>
  <Subject><!-- Ein Subject --></Subject>
  <Resource>
    <ResourceContent>
      <!-- Das XML Dokument aus Abbildung 8 -->
    </ResourceContent>
    <Attribute AttributeId="urn:...:resource-id">
      <AttributeValue>/Request[1]/Resource[1]/ResourceContent[1]/
        ExampleRoot[1]/ExampleNode[2]</AttributeValue>
    </Attribute>
    <Attribute AttributeId="urn:...:resource-parent">
      <AttributeValue>/Request[1]/Resource[1]/ResourceContent[1]/
        ExampleRoot[1]/</AttributeValue>
    </Attribute>
    <Attribute AttributeId="urn:...:resource-ancestor">
      <AttributeValue>/Request[1]/Resource[1]/ResourceContent[1]/
        ExampleRoot[1]/ExampleNode[2]/ExampleNode[1]
      </AttributeValue>
    </Attribute>
  </Resource>
</Request>
```

4 XACML multiple resource profile

Das multiple ressource profile [8] bietet einen Weg, eine Anfrage für mehrere Ressource Elemente gleichzeitig zu formulieren. Besonders bei Postprocessing ist dieses Profil äußerst nützlich. Wenn in einem solchen Fall mehrere Datensätze vom Webservice zurückgeliefert werden, und einzeln überprüft werden müssen, ist es mit diesem Profil nicht notwendig für jeden Datensatz eine eigene Anfrage an den PDP zu schicken, sondern lediglich eine.
Diese globale Anfrage wird als global access control decision request bezeichnet. Der Context Handler, der üblicherweise eng mit dem PDP verzahnt ist und möglicherweise auf dem selben physikalischen Rechner arbeitet, zerlegt dann diese einzige globale Anfrage in viele kleine. Diese kleinen Anfragen repräsentieren dann jeweils eine Anfrage für den entsprechenden Datensatz und werden an den PDP geschickt, der diese dann einzeln auswertet.
Mit diesem Mechanismus reduziert sich der Transfer zwischen PEP und Context Handler enorm. Wenn diese beiden Einheiten auf verschiedenen Servern arbeiten, geht damit eine deutliche Performance Verbesserung einher.
Der Grund für die Notwendigkeit einer Abfrage für jeden einzelnen Datensatz besteht in dem Problem, das einige Regeln nur für einzelne Datensätze ausgedrückt werden können und nicht für ganze Collections. Solche Listen von Daten werden in XACML `Bags` genannt. Auf diesen `Bags` können auch Funktionen ausgeführt werden, etwa: „Sind in diesem Bag nur die Werte X und Y enthalten?“.

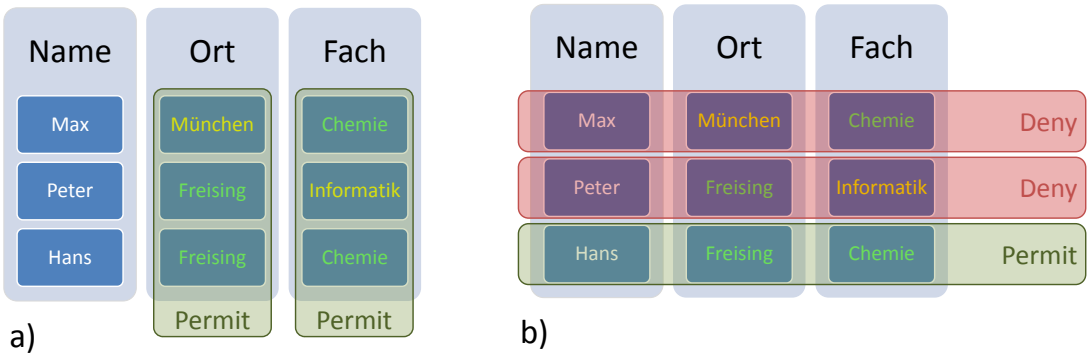

Abbildung 9: Bag-Operation vs. zeilenweise Entscheidung

Nun sei im folgenden Beispiel die Semantik gegeben: „Zugelassen sind nur Informatik Studenten aus München, und Chemie Studenten aus Freising“. In diesem Beispiel existiert ein Webservice, der Studentendaten ausliefert, und gibt in diesem Fall die drei in

Abbildung 9 dargestellten Studenten im XML Format zurück. Würde nun eine Operation auf die beiden `Bags` „Ort“ und „Fach“ ausgeführt und auf gültige Werte geprüft, wäre das Ergebnis `Permit` für alle Datensätze. Der Ort darf nur gleich „München“ oder „Freising“ sein, das Fach nur gleich „Informatik“ oder „Chemie“, und beides ist gegeben (Abb. 9a).
In Wirklichkeit sind aber zwei der drei Datensätze für den Benutzer verboten (Abb. 9b). Dies rührt von der Tatsache her, dass beim erstgenannten Verfahren der Zusammenhang der einzelnen Datensätze verloren geht. Die einzige Lösung ist somit, für alle Datensätze einzeln eine Entscheidung zu erzwingen, und erst im Nachhinein diese Einzelentscheidungen zu einer zu Kombinieren. Möglich ist auch eine Filterung der Daten im Postprocessing Schritt durch den PEP, bevor die Daten an den Benutzer geschickt werden. Hierbei kommt dem PEP zugute, dass für jeden Datensatz eine einzelne Entscheidung (`Permit` oder `Deny`) vorliegt.

4.1 Exemplarischer Webservice

Eine Anfrage an den vorher vorgestellten Webservice für Studentendaten könnte wie folgt aussehen. Der Benutzer würde mithilfe von XML-formatierten Filtern eine bestimmte Menge von Studentendaten vom Service abfragen:

```
<?xml version="1.0"?>
<sample:GetStudents service="UniDB">
  <sample:Query>
    <sample:Filter><!-- studiert Informatik --></sample:Filter>
    <sample:Filter><!-- wohnt in München --></sample:Filter>
  </sample:Query>
</sample:GetStudents>
```

Woraufhin der Webservice mit folgender XML-basierten Liste von Studenten antwortet (Die Liste ist identisch mit der vorher in Abbildung 9 dargestellten):

```
<ws:StudentCollection>
  <ws:Student>
    <ws:Name>Max</ws:Name>
    <ws:Wohnort>München</ws:Wohnort>
    <ws:Studium>Chemie</ws:Studium>
  </ws:Student>
  <ws:Student>
    <ws:Name>Peter</ws:Name>
```

```
    <ws:Wohnort>Freising</ws:Wohnort>
    <ws:Studium>Informatik</ws:Studium>
  </ws:Student>
  <ws:Student>
    <ws:Name>Hans</ws:Name>
    <ws:Wohnort>Freising</ws:Wohnort>
    <ws:Studium>Chemie</ws:Studium>
  </ws:Student>
</ws:StudentCollection>
```

4.2 Global Access Control Decision Request

Der PEP schickt nun vor dem Postprocessing der Studentendaten wie vorher beschrieben folgende `Global Access Control Decision Request`:

```
<Request>
  <Subject>
    <Attribute AttributeId="urn:...:role" DataType="string">
      <AttributeValue>Mitarbeiter</AttributeValue>
    </Attribute>
  </Subject>
  <Resource>
    <ResourceContent><!-- die Webservice Antwort --></ResourceContent>
    <Attribute AttributeId="urn:...:resource-id" DataType="xpath-expression">
      <AttributeValue>/Request[1]/Resource[1]/ResourceContent[1]/
      ws:StudentCollection[1]</AttributeValue>
    </Attribute>
    <Attribute AttributeId="urn:...:scope:xml" DataType="string">
      <AttributeValue>Descendants</AttributeValue>
    </Attribute>
  </Resource>
  <Action>
    <Attribute AttributeId="urn:...:action-id" DataType="string">
      <AttributeValue>GetStudent</AttributeValue>
    </Attribute>
  </Action>
</Request>
```

Die Anfrage enthält:

- Die Rolle des Benutzers: „Mitarbeiter“ (siehe RBAC Profil, 2).
- Die Aktion die der Benutzer durchführen will: „GetStudent“.
- Die komplette Antwort des Webservice (siehe 4.1) innerhalb des `ResourceContent` Tags.
- Ein Attribut das eine Referenz auf den Ausgangsknoten im XPath Format angibt.
- Ein Attribut das den `Scope` enthält, also den Bereich in dem nach Knoten für einzelne Anfragen gesucht werden soll. In diesem Fall `Descendants`, also alle Kindknoten des referenzierten Ausgangsknotens.

4.3 Single Access Control Decision Request

Die `Single Access Control Decision Request` sieht nun fast identisch zur globalen Anfrage aus. Nur diesmal ist kein `Scope` mehr enthalten und die `resource-id` referenziert nun nicht mehr den Ursprungsknoten, sondern einen Kindknoten. Des Weiteren werden zwei Attribute hinzugefügt, die in diesem Beispiel den Wohnort und das Fach des aktuellen Studenten enthalten. Der Context Handler erstellt also für jeden Kindknoten eine solche Anfrage und reicht sie an den PDP weiter:

```
<Request>
  <Subject>
    <Attribute AttributeId="urn:...:role" DataType="string">
      <AttributeValue>Mitarbeiter</AttributeValue>
    </Attribute>
  </Subject>
  <Resource>
    <ResourceContent><!-- die Webservice Antwort --></ResourceContent>
    <Attribute AttributeId="urn:...:resource-id" DataType="xpath-expression">
      <AttributeValue>/Request[1]/Resource[1]/ResourceContent[1]/
      ws:StudentCollection[1]/ws:Student[1]</AttributeValue>
    </Attribute>
    <!-- Keine Angabe über Scope mehr, nur noch auf einzelnen Knoten bezogen -->
    <Attribute AttributeId="urn:unidb-example:wohnort" DataType="string">
      <AttributeValue>München</AttributeValue>
    </Attribute>
    <Attribute AttributeId="urn:unidb-example:studium" DataType="string">
```

```
      <AttributeValue>Informatik</AttributeValue>
    </Attribute>
  </Resource>
  <Action>
    <Attribute AttributeId="urn:...:action-id" DataType="string">
      <AttributeValue>GetStudent</AttributeValue>
    </Attribute>
  </Action>
</Request>
```

4.4 Rule

Die Regel passt nun genau zu jeder einzeln vom Context Handler gestellten Anfrage. Hier werden die Werte der Rolle, der Aktion, der `resource-id`, des Wohnorts sowie des Studienfachs der Anfrage mit den in der Regel enthaltenen Werte abgeglichen. Stimmen alle Werte überein tritt der angegebene Effekt `Permit` ein. Die folgende Regel gibt an, dass ein Mitarbeiter alle Informatik Studenten aus München sehen darf. Die Regel für die Chemie Studenten aus Freising sähe analog aus, mit Ausnahme eben der zwei Werte. Beide Regeln ließen sich dann noch mit einem `CombiningAlgorithm` zu einer Policy kombinieren.

```
<Rule RuleId="SelectStudentsRule" Effect="Permit">
  <Target>
    <Subject>
      <SubjectMatch MatchId="string-equal">
        <AttributeValue DataType="string">Mitarbeiter</AttributeValue>
        <SubjectAttributeDesignator AttributeId="urn:...:role"/>
      </SubjectMatch>
    </Subject>
    <Resource>
      <ResourceMatch MatchId="regexp-string-match">
        <AttributeValue DataType="string">
          /Request\[1\]/Resource\[1\]/ResourceContent\[1\]/
          ws:StudentCollection\[\d+\]/ws:Student\[\d+\]
        </AttributeValue>
        <ResourceAttributeDesignator AttributeId="urn:...:resource-id"/>
      </ResourceMatch>
      <ResourceMatch MatchId="string-equal">
```

```
        <AttributeValue DataType="string">München</AttributeValue>
        <ResourceAttributeDesignator AttributeId="urn:unidb-example:wohnort"/>
      </ResourceMatch>
      <ResourceMatch MatchId="string-equal">
        <AttributeValue DataType="string">Informatik</AttributeValue>
        <ResourceAttributeDesignator AttributeId="urn:unidb-example:studium"/>
      </ResourceMatch>
    </Resource>
    <Action>
      <ActionMatch MatchId="string-equal">
        <AttributeValue DataType="string">GetStudents</AttributeValue>
        <ActionAttributeDesignator AttributeId="urn:...:action-id"/>
      </ActionMatch>
    </Action>
  </Target>
</Rule>
```

Literatur

[1] Jan Herrmann. Entwicklung und implementierung einer raumbezogenen zugriffskontrolle für geo web services, 3 2005.

[2] Curbera Duftler Khalaf Nagy Mukhi Weerawarana. Internet Computing IEEE. Unraveling the web services web: an introduction to soap, wsdl and uddi. 4 2002.

[3] Sun Microsystems. Sun's xacml implementation programmer's guide for version 1.2, 7 2004.

[4] Klaus-Rainer Müller. *Handbuch Unternehmenssicherheit: Umfassendes Sicherheits-, Kontinuitäts- und Risiskomanagement mit System.* Vieweg+Teubner Verlag, 2005.

[5] OASIS. Core and hierarchical role based access control (rbac) profile of xacml v2.0, 2 2005.

[6] OASIS. extensible access control markup language (xacml) version 2.0, 2 2005.

[7] OASIS. Hierarchical resource profile of xacml v2.0, 2 2005.

[8] OASIS. Multiple resource profile of xacml v2.0, 2 2005.

[9] Mark O'Neill. *Web Services Security.* McGraw-Hill Professional, 2003.

[10] W3C. Xml path language (xpath) version 1.0, 11 1999.

[11] Douglas B. West. *Introduction to Graph Theory.*

[12] Wikipedia. extensible access control markup language, 2 2009.