Analysis of the added value of the Access Control Metamodel for Web Service-Oriented Architecture by Presenting it in Business Rules
Analysis of the added value of the Access Control Metamodel for Web Service-Oriented Architecture by Presenting it in Business Rules
Abstract

In this thesis, the added value of the “Access Control Metamodel for Web Service-Oriented Architecture” by Emig et al. is analyzed. This model is a welcome addition to the existing authorization models for use in a Web Service-Oriented Architecture, since it handles the authorization decision for orchestrational service compositions. A security flaw was found that undermines the purpose of the Access Control Metamodel for Web Service-Oriented Architecture.

Four claims made by the authors about the Metamodel have been analyzed:

1. Explicit Composition
   The model is capable to handle explicit service composition.

2. Support Parameters
   The model supports including parameters in the authorization decision process.

3. Limit Service Calls
   The model avoids unnecessary service calls.

4. Scaling
   The Access Control Metamodel for Web Service-Oriented Architecture supports the scaling of an increasing amount of both objects and subjects.

These claims are validated by presenting the metamodel in business rules using Ampersand.

It is shown that claim 1, Explicit Composition and claim 2, Support Parameters are in conflict with each other.

Claim 3, Limit Service Calls proved to be valid.

It is shown that to validate claim 4, Scaling, a different research method has to be used.

The conclusion is that possible security vulnerability and the lack of support for choreographical service compositions make the model not mature enough for productional use.
## Contents

<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>List of Figures</td>
<td>8</td>
</tr>
<tr>
<td>Preface</td>
<td>10</td>
</tr>
<tr>
<td><strong>1 Research framework</strong></td>
<td>12</td>
</tr>
<tr>
<td>1.1 Research questions</td>
<td>12</td>
</tr>
<tr>
<td>1.2 Research approach</td>
<td>12</td>
</tr>
<tr>
<td>1.3 Ampersand method</td>
<td>13</td>
</tr>
<tr>
<td><strong>2 Introduction of the Metamodel</strong></td>
<td>16</td>
</tr>
<tr>
<td>2.1 The Access Control Metamodel for Web Service-Oriented Architecture</td>
<td>16</td>
</tr>
<tr>
<td>2.2 Claims made by the creators</td>
<td>17</td>
</tr>
<tr>
<td><strong>3 Background of the Metamodel</strong></td>
<td>18</td>
</tr>
<tr>
<td>3.1 Service-Oriented Architecture</td>
<td>18</td>
</tr>
<tr>
<td>3.1.1 Services</td>
<td>18</td>
</tr>
<tr>
<td>3.1.2 Composition of Services</td>
<td>20</td>
</tr>
<tr>
<td>3.1.3 Security in a Service-Oriented Architecture</td>
<td>21</td>
</tr>
<tr>
<td>3.2 Authorization Models</td>
<td>22</td>
</tr>
<tr>
<td><strong>4 Validation of the Metamodel</strong></td>
<td>26</td>
</tr>
<tr>
<td>4.1 Translation of the Metamodel to Ampersand</td>
<td>26</td>
</tr>
<tr>
<td>4.1.1 Translation of the UML model to Ampersand</td>
<td>26</td>
</tr>
<tr>
<td>4.1.2 The translation of the written business rules</td>
<td>27</td>
</tr>
<tr>
<td>4.2 Validation of the claims</td>
<td>30</td>
</tr>
<tr>
<td>4.2.1 Claim Explicit Composition</td>
<td>30</td>
</tr>
<tr>
<td>4.2.2 Claim Support Parameters</td>
<td>32</td>
</tr>
<tr>
<td>4.2.3 Claim Limit Service Calls</td>
<td>39</td>
</tr>
<tr>
<td>4.2.4 Claim Scaling</td>
<td>39</td>
</tr>
<tr>
<td><strong>5 Other drawbacks of the Metamodel</strong></td>
<td>40</td>
</tr>
<tr>
<td>5.1 Separation of concerns</td>
<td>40</td>
</tr>
<tr>
<td>5.2 Choreographical web service composition</td>
<td>40</td>
</tr>
<tr>
<td><strong>6 Conclusion</strong></td>
<td>42</td>
</tr>
<tr>
<td>References</td>
<td>44</td>
</tr>
<tr>
<td><strong>A ADL Claim 1, Explicit Composition</strong></td>
<td>48</td>
</tr>
<tr>
<td>A.1 Validation of claim 1, Explicit Composition, using ADL</td>
<td>48</td>
</tr>
<tr>
<td>A.2 Source</td>
<td>49</td>
</tr>
</tbody>
</table>
B ADL Claim 2

B.1 Validation of claim 2, Support Parameters, using ADL .......................... 54
B.2 Sources ................................................................................................. 55
  B.2.1 Parameters in a single web service .................................................. 55
  B.2.2 Parameter secured web service in an orchestration ......................... 60
  B.2.3 Parameter secured web service with less restrictive permission .......... 64
List of Figures

<table>
<thead>
<tr>
<th>Figure</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Research Framework</td>
<td>13</td>
</tr>
<tr>
<td>2</td>
<td>The Access Control Metamodel for Web Service-Oriented Architecture as designed by Emig et al. [8]</td>
<td>16</td>
</tr>
<tr>
<td>3</td>
<td>Bind-Invoke-Execute</td>
<td>20</td>
</tr>
<tr>
<td>4</td>
<td>Security Interceptors</td>
<td>22</td>
</tr>
<tr>
<td>5</td>
<td>No violations in success scenario</td>
<td>48</td>
</tr>
<tr>
<td>6</td>
<td>Violation of rule 4.24</td>
<td>49</td>
</tr>
<tr>
<td>7</td>
<td>Violation of rule 4.24 in an orchestration</td>
<td>54</td>
</tr>
<tr>
<td>8</td>
<td>No violations when invoking ws_savecustomerdetails without parameters</td>
<td>55</td>
</tr>
</tbody>
</table>
Preface

In this thesis I present the research I did as a final part of my study Business Process Management & ICT at the Open University of the Netherlands. The past decade I have seen the emergence of Service-Oriented Architecture. Since I’m working as an IT Architect, I realise that using Service-Oriented Architecture in combination with the increased regulations on compliancy (SOX, Basel II), places new demands on the security of services. I am very interested in identity and access management, so I decided to focus my graduate research on this area.

Reading about security in conjunction with Service-Oriented Architecture, I came across the article “Access Control Metamodel for Web Service-Oriented Architecture” of Christian Emig, Frank Brandt and Sebastian Abeck, researchers of the Research Group “Cooperation & Management”, Institute of Telematics at the Universität Karlsruhe [8]. In their article they introduce an authorization metamodel specially equipped for a (Web) Service-Oriented Architecture. What makes their model special, is the ability to determine authorizations based on both user information as well as on the content off the message. I wondered whether their model was feasible in practice, so I made validation of the Metamodel the subject of my research.

Prof. Dr. Ir. S.M.M. Joosten introduced me to the Ampersand method, which he developed in 2008 [14]. This method is capable to check the functional specifications of information systems for errors. This capability made the method very suitable to validate whether the Metamodel does what Emig et al. claim it should do.

To be able to perform this research, I had to learn more about relation algebra. It was interesting to discover that the Ampersand method is very suitable to adopt the requirements of a system, so you can validate a system before it is build. In my work I already used the method. My research of the Metamodel inspired me to develop an alternative authorization model. This model is now being used by my colleagues and myself on a daily basis. Over all, my study Business Process Management & ICT enabled me to develop a scientific view on IT issues. It deepened my understanding of possibilities in my work and has broadened my interest.

I would like to thank Prof. Dr. Ir. S.M.M. Joosten for his guidance and lectures ‘Ampersand’, often in the evening hours. Dr. E.E. Roubtsova took the time to be a critical reader and made suggestions of great value to improve my thesis. Mr. H.J.M. Joosten and Drs. G. Michels have been as kind as to validate my ADL models. Finally thanks to my wife Daphna, for her support during the years it took to finish my master study.

Tilburg, February 2010
Raymond Roelands
1 Research framework

This section presents the subject of the research, the research questions and the research approach.

1.1 Research questions

The purpose of this research is to validate the research, done by Christian Emig, Frank Brandt and Sebastian Abeck of the Research Group Cooperation & Management of the Universität Karlsruhe, of an access control metamodel for web service-oriented architecture [8].

The research question is:
What is the added value of the Access Control Metamodel for Web Service-Oriented Architecture?

This leads to the following subquestions:

1. Is the claim for handling the authorization in explicit service composition scenarios valid?

2. Is the claim that the Access Control Metamodel for Web Service-Oriented Architecture is able to include input parameters in the authorization decision process valid?

3. Is the claim that Access Control Metamodel for Web Service-Oriented Architecture prevents unnecessary service invocations if a subject is not authorized for all services used in an orchestration valid?

4. Is the claim that the Access Control Metamodel for Web Service-Oriented Architecture respects an increasing amount of both objects and subjects valid?

1.2 Research approach

To answer the research questions the following actions will be executed. This approach is schematically shown in figure 1.
First the Access Control Metamodel for Web Service-Oriented Architecture will be examined and the claims made in the article will be extracted. This will done in chapter 2.

Next the background of the Access Control Metamodel for Web Service-Oriented Architecture will be examined. This will be done in chapter 3.

Next the model will be validated using the Ampersand method (see 1.3) and compared against the literature of Service-Oriented Architecture. This will be done in chapter 4.

Finally, I present my conclusions in chapter 6.

1.3 Ampersand method

Ampersand [14] is a method to derive functional specifications from formally defined business requirements. This approach uses relation algebra as a requirements language to specify the functional requirements, whereby each functional requirement is translated to a rule. The relation algebra rules can be described in ADL. Using an ADL tool, the formal specifications can
be tested for rule violations. This way, the functional requirements can be tested before an IT system is built.
2 Introduction of the Metamodel

This chapter introduces the Access Control Metamodel for Web Service-Oriented Architecture and describes the claims about the model made by its creators.

2.1 The Access Control Metamodel for Web Service-Oriented Architecture

The Access Control Metamodel for Web Service-Oriented Architecture was created in 2007 by Christian Emig, Frank Brandt and Sebastian Abeck of the Research Group Cooperation & Management of the Universität Karlsruhe. This model, shown in figure 2, is an approach how to handle authorization verification in a Web Service-Oriented Architecture.

Figure 2: The Access Control Metamodel for Web Service-Oriented Architecture as designed by Emig et al.[8]

Eric Yuan and Jin Tong found that the current security models in place are too simple, static and coarse-grained to provide the rich semantics for web service level access control policies. Many systems simply authorize access based on roles and only a few take operational contexts into consideration in their access control policies such as the transaction at hand, current thread level or the residing "community of interest" [22].
Based on this research Emig et al. wanted to design an authorization model that could cope with the needs of the Web Service-Oriented Architecture [8].

In chapter 3 the use of authorization models in service-Oriented Architecture will be explained.

The model (figure 2) describes the authorization of people or systems (in the model described as subjects) for making use of web services (in the model shown as objects).

Each web service operation is linked one on one to a policy. If a subject wants to invoke a web service, the subject must be authorized for the policy linked to the web service. A subject is authorized for the policy if the subject has at least one permission linked to the policy.

To get a permission, the subject has to have the possession of all subject attributes required by the permission. A subject attribute is an attribute containing information about the subject. An example of a subject attribute is a user-name, a role name or a security token.

A permission can also require that a parameter used to invoke the web service or an environmental variable has a certain value. For example, this can be used to make sure a subject is allowed to see his own account details but not the account details of someone else. The environmental variable can be used to authorize a subject, based on date, time or any other environment type (e.g. test or production).

If an invocation of a web service leads to invocation of other web services, the subject must be authorized for all policies involved.

2.2 Claims made by the creators

The authors of the Access Control Metamodel for Web Service-Oriented Architecture made the following claims about the model [8]:

1. Explicit Composition
   The model is capable to handle explicit service composition.

2. Support Parameters
   The model supports including parameters in the authorization decision process.

3. Limit Service Calls
   The model avoids unnecessary service calls.

4. Scaling
   The Access Control Metamodel for Web Service-Oriented Architecture supports the scaling of an increasing amount of both objects and subjects.
3 Background of the Metamodel

The Access Control Metamodel for Web Service-Oriented Architecture is an authorization model created for use in a service-Oriented Architecture. This chapter explains what service-Oriented Architecture is and what authorization models are.

3.1 Service-Oriented Architecture

A Service-Oriented Architecture establishes an architectural model that aims to enhance the efficiency, agility and productivity of an enterprise by positioning services as the primary means through which solution logic is represented in support of the realization of strategic goals associated with service oriented computing [10]. The emergence of web services developments and standards in support of automated business integration has driven major technical advances in the integration software space, most notably, the service-Oriented Architecture. The purpose of this architecture is to address the requirements of loosely coupled, standard-based and protocol independent distributed computing, mapping enterprise information systems isomorphically to the overall business process flow [20]. Zimmerman [23] defines three levels of abstractions within a service-Oriented Architecture:

- **Business processes**: actions of activities to perform specific business goals by invoking multiple services
- **Services**: logical groupings of operations.
- **Operations**: units of functions within specific interfaces operating on received data and returning structured responses.

Zimmermann explicitly distinguishes between operations and services. Commonly the terms services and operations are used interchangeably (e.g. Papazolou et al. [20]). I share this view and will in future refer to services.

3.1.1 Services

To be independent of any technology, the number of implementation restrictions are limited at the level of the service interface. To be part of a service-Oriented Architecture, the interfaces of a service must be described in a description language and those interfaces should perform useful business processes. The fundamental intent of a service in a service-Oriented Architecture is to represent a reusable unit of business-complete work.
It should comply to the following requirements [20]:

- The service is self-contained and autonomous:
  The service maintains its own state and should function as a black box to external components. Service opaqueness guarantees that external components neither know nor care how services perform their function, they merely anticipate to return the expected result. The implementation and execution space of the application providing the desired functionality, is encapsulated behind the services’ interface [5].

- The service is platform independent:
  A client from any communication device using any computational platform, operating system and programming language should be able to use the service.

- The service should be able to be located, invoked and combined dynamically.

Erl [10] describes services as

physically independent software programs with distinct design characteristics that support the attainment of the strategic goals associated with service oriented computing. Each service is assigned its own distinct functional context and is comprised of a set of capabilities related to this context. Those capabilities suitable for invocation by external consumers programs are commonly expressed via a suitable contract (much like a traditional API).

All technologies that directly implement service interfaces with WSDL [7], and communicate with SOAP messages [4] using a certain transport mechanism (e.g. CORBA, IBM WebSphere MQ or HTTP), can be involved in a Service-Oriented Architecture [20]. But the web service technology (i.e. SOAP over HTTP) is the most preferred implementation to accomplish an interoperable service-Oriented Architecture [15] [20].

The Access Control Metamodel for Web Service-Oriented Architecture is specifically designed for use in a service-Oriented Architecture based on web services. Now, what is the position of services within an service-Oriented Architecture? A service-Oriented Architecture uses a Publish-Find-Bind-Execute paradigm as shown in Figure 3.
Service providers build services and offer them through internet or intranet. They register services with service brokers and publish them in distributed registries. Each service has an interface, known as contract, and functionality, which is kept separate from the implementation. The service consumers search for services (based on certain criteria) and, when found, a dynamic binding is performed. In this case, the service provides the consumer with the contract details and an endpoint address. The consumer then invokes the service [17].

A service-Oriented Architecture provides a flexible architecture that unifies business processes by modularizing large applications into services. This makes it possible for application developers, using any operating system or programming language, to create new business processes using new or existing services and in doing so, offering new business functionality to the enterprise.

3.1.2 Composition of Services

If the existing functionality offered by applications in an enterprise are wrapped by a service, the enterprise can easily create new business processes by combining services. This combining of services is called web service composition. Hafner and Brue [13] distinguish two methods of web service composition.

The first method is an orchestration of web services, where the orchestration describes how web services interact with each other at the message level, including the business logic and the execution order of the interactions from the viewpoint of the partner controlling the workflow execution. Web service orchestrations are described in a language. Examples of such languages are the Business Process Execution Language for Web Services WS-BPEL [2], BPML, WSCI and ebXML. WS-BPEL describes the orchestration of web services in an XML-based language.

The second method is called choreography, in which the sequence of web service invocations is described in a business protocol and there is no central control of workflow execution. The Access Control Metamodel for Web
Service-Oriented Architecture only supports orchestrational compositions.

3.1.3 Security in a Service-Oriented Architecture

Service-Oriented Architecture offers a lot of benefits when it comes to loose coupling and integration but also presents us with some challenges, for instance security \[17\]. Eric Pulier and Hugh Taylor wrote in their book *Understanding Enterprise SOA* \[21\]:

> The SOA’s inherent security problem stem from the ways in which the SOA replaces traditional security parameters with new, open standards. The security problem is twofold in that not only are the new standards completely open. No one owns them but they were also developed without security in mind.

But what is security? Adams describes in his article *Best Practices for Web services, Part 11: Web services security* \[1\] the security triad Authentication, Data Integrity and Data Confidentiality.

**Authentication** means verifying the identity of the communicating principals to one another \[18\]. In web service communication this means that both the service and the client have to prove their identity.

**Data Integrity** addresses the unauthorized or accidental modification of data. This includes data insertion, deletion, and modification. To ensure data integrity, a system must be able to detect unauthorized data modification. The goal is for the receiver of the data to verify that the data has not been altered \[12\].

**Data Confidentiality** means restricting access to sensitive data to only those individuals who are authorized to view the data. Confidentiality measures prevent the unauthorized disclosure of information to unauthorized individuals or processes. \[12\].

To accomplish secure web service communication the WS-Security specification has been set up. It provides encryption, authentication and integrity support \[19\].

Authentication is supported by the specification of security tokens. A requestor is identified by the web service based on the security token that is part of the message. There are three kinds of security tokens:

**Username Tokens** : A username token can be used to identify the user to the web service using a username and password combination.

**Binary Token** : A binary token contains a X.509, Kerberos or other non-XML token identifying the subject.
XML Token: An XML token contains XML identifying the subject. This can be, for example, an SAML Assertion that contains the claims of the subject.

Tokens are supplied in a SOAP header. This way the security details are not mixed with the content of the message. The creation and validation of the SOAP header can be handled by a security interceptor. This makes the security policy transparent to both the client and web service application [3]. This is visualized in figure 4.

![Figure 4: Security Interceptors](image)

When the client application sends a request to the web service using a web service client, the web service client contains the logic of communicating with the web service. The security interceptor at the client side catches the outbound message and adds authentication details in the form of a security token to the message. At the server side, the security interceptor validates the user token and validates the credentials against a user store, e.g. a database or a LDAP store (Lightweight Directory Access Protocol).

Data integrity and confidentiality are supported by WS-Security by offering encryption and the possibility to add digital signatures. Unlike security on transport level, WS-Security offers a full end-to-end security because the authentication, encryption and digital signatures are on message level [19]. This means that instead of encrypting the transport mechanism, the message itself is encrypted. This ensures that the message can not be read unnoticed or tampered with before the message reaches the intended receiver of the message.

3.2 Authorization Models

Access management consists of two parts, authentication and authorization. The authentication handles the identification of the subject (‘who are you’). The authorization handles what the subject is allowed to do. To adminis-
trate the authorizations of subjects in organizations, different authorization 
models are created, each with its own advantages and disadvantages.

In 1969 Lampson [16] described the basics of access control and introduced 
the concepts of objects and subjects. Objects are the items that have to be protected. Subjects are the identities that want to have access to the objects. In the approach of Lampson, the subjects are directly linked to the objects using an access matrix. Because the subjects (identities) in this approach are directly linked to the object, this model is called Identity Based Access Control (IBAC) [8].

In the 1970s the authorization models evolved to Lattice Based Access Control (LBAC). LBAC brought an authorization model based on subjects and objects in combination with two kinds of security policies. One policy described the kind of access to the object (read or write), the second policy was a security qualification of the object (unclassified, confidential, secret, top secret). These two policies defined the security level of the object. The access control required a security level of the subject, dominating the security level of the object. The LBAC mechanisms are not flexible and do not scale very well. They are only useful in military or similar scenarios [22].

The most commonly used access control paradigm at this moment is the Role Based Access Control (RBAC), introduced in 1992 [11]. In RBAC a subject can be linked to one or more business roles. The authorization to access an object is granted to a role. This authorization model makes it easy to scale and reduces the administration needed to grant and revoke permissions of subject to objects.

In 2005, Eric Yuan and Jin Tong introduced the Attribute Based Access Control (ABAC) as a proposition for securing web service invocations. ABAC is based on attributes. An attribute is a security relevant characteristic. In ABAC there are three important attribute types:

**Subject attributes** are attributes containing specific characteristics of the subject, e.g. the user-id, rolenames, organization, unit, email address or age.

**Resource attributes** are attributes of the objects the subject acts upon. Examples of resource attributes are web services, documents and file system properties. These resource attributes contain characteristics or properties that can be used to make authorization decisions.

**Environment attributes** describe the environmental, operational or technical context in which the access to the object occurs. Examples are time, the current version of the operating system or whether a virus scanner is installed.

In an ABAC attribute authorities are responsible for the creation and management of the attributes. A policy enforcement point is responsible for the
request of authorization decisions. *Policy decision points* are responsible for the evaluation of the applicable attributes and decide whether the subject is authorized [22].

The Access Control Metamodel for Web Service-Oriented Architecture is based on a combination of the ABAC and the RBAC models [8].
4 Validation of the Metamodel

In this chapter the Metamodel will be translated to Ampersand. Using the thus obtained functional specifications, the claims made by the authors of the Access Control Metamodel for Web Service-Oriented Architecture will be validated.

4.1 Translation of the Metamodel to Ampersand

The Access Control Metamodel for Web Service-Oriented Architecture (figure 2) is designed using the Unified Modelling Language (UML). To validate the model using Ampersand the first step is to translate this UML model to formal specifications in Ampersand. Next the business rules from the article, see chapter 2, will be translated to Ampersand. Together they form the functional specifications of the Access Control Metamodel for Web Service-Oriented Architecture.

4.1.1 Translation of the UML model to Ampersand

The UML model in figure 2 represents the following specifications:

- A Subject has zero or more Business Roles
  \[ \text{hasBusinessRole} :: \text{Subject} \times \text{BusinessRole} \]  
  \( (4.1) \)

- A Business Role contains zero or more other Business Roles (hierarchy)
  \[ \text{includes} :: \text{BusinessRole} \times \text{BusinessRole} \]  
  \( (4.2) \)

- A Business Role consists of one or more Subject Attributes
  \[ \text{consistsOf} :: \text{BusinessRole} \times \text{SubjectAttribute} \]  
  \[ \vdash \text{consistsOf}; \text{consistsOf} \]  
  \( (4.3) \)

- A Subject has zero or more Subject Attributes
  \[ \text{attributeAssignment} :: \text{Subject} \times \text{SubjectAttribute} \]  
  \( (4.4) \)

- The authorization of an Object is defined using exactly one Policy. Because a Web Service Operation, further referenced in this chapter as Web service, derives from Object and is the only descendant, I will use Web Service instead of Object.
  \[ \text{policyAssignment} :: \text{WebService} \times \text{Policy} \]  
  \[ \vdash \text{policyAssignment}; \text{policyAssignment} \]  
  \[ \vdash \text{policyAssignment} \]  
  \[ \vdash \text{policyAssignment} \]  
  \( (4.5) \)
• A Composition consists of the Policy of the invoked Web Service and dependant Web Services. This can be described as: A Policy is dependant on zero or more Web Services.

$$\text{isDependantOff} :: \text{Policy} \times \text{WebService}$$ (4.6)

• To be authorized for a Policy at least one Permission is needed

$$\text{grantsAuthorizationTo} :: \text{Permission} \times \text{Policy}$$ (4.7)

\[ \sqsubseteq_{\text{Permission}} \vdash \text{grantsAuthorizationTo} ; \text{grantsAuthorizationTo}^\sim \]
\[ \sqsubseteq_{\text{Policy}} \vdash \text{grantsAuthorizationTo} ; \text{grantsAuthorizationTo}^\sim \]

• A Permission has zero or more Subject Attributes

$$\text{requires} :: \text{Permission} \times \text{SubjectAttribute}$$ (4.8)

• A Permission can have zero or more required Parameters

$$\text{requiresParameterOfType} :: \text{Permission} \times \text{SubjectAttributeType}$$ (4.9)

• A Permission can require zero or more Environment State Attributes

$$\text{requiresEnvironment} :: \text{Permission} \times \text{Variable}$$ (4.10)

4.1.2 The translation of the written business rules

The following specifications can be extracted from the accompanying text of the Access Control Metamodel for Web Service-Oriented Architecture in the article [8].

• The lines:

Subjects are characterized by a defined amount of subject attributes. From the business perspective, subjects act in the context of a business role. In our model, the concept of business role relates to a defined amount of (finer-grained) subject attributes. Role hierarchies can be defined as well, all together finally mapping to a set of subject attributes [8, p. 4].

Leads to:

$$\text{hasBusinessRole}; \text{includes} \vdash \text{hasBusinessRole}$$ (4.11)

$$\text{hasSubjectAttributes} :: \text{Subject} \times \text{SubjectAttribute}$$

\[ \sqsubseteq_{\text{Subject}} \vdash \text{hasSubjectAttributes}; \text{hasSubjectAttributes}^\sim \]

$$\text{hasSubjectAttributes} = \text{attributeAssignment} \cup $$ (4.13)

$$\text{hasBusinessRole}; \text{consistsOff}$$
• For the authorization to be validated, a subject first needs to invoke a web service:

\[
\text{makesRequest} :: \text{Subject} \times \text{Request} \quad (4.14)
\]

\[
\text{makesRequest};\text{makesRequest}^{-} \vdash \mathbb{I}_\text{Subject}
\]

\[
\text{makesRequest};\text{makesRequest}^{-} \vdash \mathbb{I}_\text{Request} \vdash \text{makesRequest};\text{makesRequest}^{-}
\]

\[
\text{invokes} :: \text{Request} \times \text{Webservice} \quad (4.15)
\]

\[
\text{invokes};\text{invokes}^{-} \vdash \mathbb{I}_\text{Webservice}
\]

\[
\text{invokes};\text{invokes}^{-} \vdash \mathbb{I}_\text{Request} \vdash \text{invokes};\text{invokes}^{-}
\]

• The lines:

We call the composition of all Permissions for one specific Object (which is in our case a Web Service Operation) Policy; so the authorization of each Object is defined using exactly one Policy. One goal of Service-Oriented Architecture is the reuse of existing services in different contexts. This is why we use the concept of Permission; each Permission covers one service usage context. The Policy is the composition of all Permissions of an Object using boolean ”OR” concatenation \([8, \text{ p. 5}].\)

Can be converted to:

\[
\text{hasPermission} :: \text{Subject} \times \text{Permission} \quad (4.16)
\]

\[
\text{isAuthorizedForPolicy} :: \text{Subject} \times \text{Policy} \quad (4.17)
\]

The attribute constraint verifies the subject has only the permissions the subject has all required attributes for.

\[
\text{hasPermission}^{-} = (\neg \text{requires} \upharpoonright \text{hasSubjectAttributes}) \quad (4.18)
\]

If the subject is authorized for the permission based on the subject attributes, the input parameters linked to the permission also have to correspond the the parameters attached to the request.

\[
\text{hasParameter} :: \text{Request} \times \text{SubjectAttribute} \quad (4.19)
\]

\[
\text{hasValidParameter} :: \text{Request} \times \text{SubjectAttribute} \quad (4.20)
\]

\[
\text{hasValidParameter} \vdash \text{hasParameter} \quad (4.21)
\]
And to compare a parameter against an attribute, an attribute has to be of a certain type:

\[
\text{isTypeOfS} :: \text{SubjectAttribute} \times \text{SubjectAttributeType}
\]

\[
isTypeOfS; \text{isTypeOfS} \vdash \text{I}_{\text{SubjectAttributeType}}
\]

\[
\text{I}_{\text{SubjectAttribute}} \vdash \text{isTypeOfS}; \text{isTypeOfS}
\]  

(4.22)

- If a subject invokes a web service using a parameter, the subject must have a subject attribute containing the same value

\[
\text{makesRequest}; \text{hasValidParameter} \vdash \text{hasSubjectAttributes}
\]  

(4.23)

\[
\text{hasPermission}^- = (\neg \text{requires} \uparrow \text{hasSubjectAttributes}^-)
\]

\[
\cap
\]

\[
(\neg \text{requiresParamaterOfType} \uparrow
\]

\[
\text{makesRequest}; \text{hasValidParameter};
\]

\[
isTypeOfS^-)
\]

\[
\cap
\]

\[
((\text{makesRequest}; \text{invokes}; \text{policyAssignment};
\]

\[
\text{grantsAuthorizationTo}^-)\)
\]

\[
\cup
\]

\[
(\text{makesRequest}; \text{invokes};
\]

\[
\text{policyAssignment}; \text{isDependantOff};
\]

\[
\text{policyAssignment}; \text{grantsAuthorizationTo}^-)\)
\]

\[
\cap
\]

\[
\text{hasPermission}^-  
\]

(4.24)

This implies that a request can have a parameter:

\[
\text{hasParameter} :: \text{Request} \times \text{SubjectAttribute}
\]  

(4.25)

And to compare a parameter against an attribute, an attribute has to be of a certain type:

\[
isTypeOfS :: \text{SubjectAttribute} \times \text{SubjectAttributeType}
\]

\[
isTypeOfS; \text{isTypeOfS} \vdash \text{I}_{\text{SubjectAttributeType}}
\]

\[
\text{I}_{\text{SubjectAttribute}} \vdash \text{isTypeOfS}; \text{isTypeOfS}
\]  

(4.26)

- If a subject invokes a web service using a parameter, the subject must have a subject attribute containing the same value

\[
\text{makesRequest}; \text{hasParameter} \vdash \text{hasSubjectAttributes}
\]  

(4.27)
A subject is authorized for a policy if it has at least one permission linked to the policy. If the policy is dependant on other web services the subject also has to be authorized for all policies in the composition:

\[
isAuthorizedForPolicy = hasPermission; grantsAuthorizationTo
\]

\[
isAuthorizedForPolicy; isDependantOff; policyAssignment \\vdash isAuthorizedForPolicy
\] (4.29)

And to identify whether a subject is authorized to make an invocation of the web service the subject must be authorized for the linked policy:

\[
(makesRequest; invokes) \sim = (\neg policyAssignment \\vdash isAuthorizedForPolicy) \cap (makesRequest; invokes) \sim
\] (4.30)

4.2 Validation of the claims

After translating the Access Control Metamodel for Web Service-Oriented Architecture and its business rules in Ampersand, these functional specifications are used to validate the claims made by the authors of the Access Control Metamodel for Web Service-Oriented Architecture.

4.2.1 Claim Explicit Composition

To validate whether the model is capable to handle explicit service composition a test case is created. This test case contains an orchestration of three web services, each with their own policy:

\[
policyAssignment = \{\langle ws\_findcustomerdetails, pol\_findcustomerdetails \rangle, \\
\langle ws\_savecustomerdetails, pol\_savecustomerdetails \rangle, \\
\langle ws\_createaccount, pol\_createaccount \rangle \}
\] (4.31)

Each web service has one single permission:

\[
grantsAuthorizationTo = \{\langle per\_findcustomerdetails, pol\_findcustomerdetails \rangle, \\
\langle per\_savecustomerdetails, pol\_savecustomerdetails \rangle, \\
\langle per\_createaccount, pol\_createaccount \rangle \}
\] (4.32)
All three web services require the role rl_employee:

\[
requires = \{\langle \text{per\_findcustomerdetails, rl\_employee}\rangle, \\
\langle \text{per\_savecustomerdetails, rl\_employee}\rangle, \\
\langle \text{per\_createaccount, rl\_employee}\rangle\}\] (4.33)

The web service ws\_createaccount depends on ws\_findcustomerdetails and ws\_savecustomerdetails:

\[
isDependantOff = \{\langle \text{pol\_createaccount, ws\_findcustomerdetails}\rangle, \\
\langle \text{pol\_createaccount, ws\_savecustomerdetails}\rangle\}\] (4.34)

The web service create account is invoked by the subject John. John is member of the business role Employee:

\[
hasBusinessRole = \{\langle \text{John, Employee}\rangle\}\] (4.35)

\[
consistsOff = \{\langle \text{Employee, rl\_employee}\rangle\}\] (4.36)

\[
hasSubjectAttributes = \{\langle \text{John, rl\_employee}\rangle\}\] (4.37)

\[
makesRequest = \{\langle \text{John, request1}\rangle\}\] (4.38)

\[
invokes = \{\langle \text{request1, ws\_createaccount}\rangle\}\] (4.39)

To invoke all services in the orchestration, subject John needs all three permissions of definition 4.33:

\[
hasPermission = \{\langle \text{John, per\_createaccount}\rangle, \\
\langle \text{John, per\_findcustomerdetails}\rangle, \\
\langle \text{John, per\_savecustomerdetails}\rangle\}\] (4.40)

\[
(4.41)
\]

And the subject needs all required subject attributes for all permissions

\[
hasRequiredAttributesFor = \{\langle \text{John, per\_createaccount}\rangle, \\
\langle \text{John, per\_findcustomerdetails}\rangle, \\
\langle \text{John, per\_savecustomerdetails}\rangle\}\] (4.42)

Because the subject John has a permission for every policy involved in the request, he is authorized for all policies:

\[
isAuthorizedForPolicy = \{\langle \text{John, pol\_createaccount}\rangle, \\
\langle \text{John, pol\_findcustomerdetails}\rangle, \\
\langle \text{John, pol\_savecustomerdetails}\rangle\}\] (4.43)
All relations now contain the information to fulfill a request and all rules (4.11, 4.13, 4.27, 4.30, 4.28 and 4.24) are valid.

This is the validation in the success scenario. Now a failing scenario will be tested. In which case the subject is authorized for all services in the composition except for \textit{ws\_savecustomerdetails}. To do so, an extra subject attribute requirement is added to the permission linked to the permission \textit{per\_savecustomerdetails}. This changes the relation \textit{requires} in definition (4.33) to:

\[
\text{requires} = \{\langle \text{per\_findcustomerdetails, rl\_employee} \rangle, \\
\langle \text{per\_savecustomerdetails, rl\_employee} \rangle, \\
\langle \text{per\_savecustomerdetails, rl\_manager} \rangle, \\
\langle \text{per\_createaccount, rl\_employee} \rangle\} 
\]

(4.44)

Now the permission \textit{per\_savecustomerdetails} requires that the subject also has to have the business role \textit{rl\_manager}. This ensures that:

\[
(\neg\text{requires} \uparrow \text{hasSubjectAttributes}^{\sim}) = \{\langle \text{per\_createaccount, John} \rangle, \\
\langle \text{per\_findcustomerdetails, John} \rangle\} 
\]

(4.45)

The intersection in (4.24) with \((\neg\text{requires} \uparrow \text{hasSubjectAttributes}^{\sim})\) restricts the permissions of the subject. Now the subject only has the permissions it has the required subject attributes for. Relation (4.45) shows that the elements in \((\neg\text{requires} \uparrow \text{hasSubjectAttributes}^{\sim})\) are not equal to the elements in relation (4.40), therefore rule (4.24) is falsified.

Based on the Ampersand method, I conclude claim Explicit Composition is successfully validated.

### 4.2.2 Claim Support Parameters

An important feature of the Access Control Metamodel for Web Service-Oriented Architecture is the ability to involve the input parameters of the web service request in the authorization decision process. To validate this claim two scenarios are elaborated.

In scenario 1 the Access Control Metamodel for Web Service-Oriented Architecture is tested to see if it handles involving input parameters in a single web service operation. The scenario contains a single web service \textit{SaveCustomerDetails} that will be available to both employees and customers. The employees can save customer details for all customers but the customer himself is restricted to only making changes to his own details.

The web service and its policy are defined as:

\[
policyAssignment = \{\langle \text{ws\_savecustomerdetails, pol\_savecustomerdetails} \rangle\} 
\]

(4.46)
The policy contains two permissions. One for the customers and one for the employees.

\[
\text{grantsAuthorizationTo} = \{ \langle \text{per\_savecustomerdetailsEmpl}, \text{pol\_savecustomerdetails} \rangle, \\
\langle \text{per\_savecustomerdetailsCust}, \text{pol\_savecustomerdetails} \rangle \} 
\]  

(4.47)

The permission \text{per\_savecustomerdetailsCust} is the permission used by the customers and requires the business role \text{rl\_customer}. The permission \text{per\_savecustomerdetailsEmpl} is used by the employees and requires the business role \text{rl\_employee}:

\[
\text{requires} = \{ \langle \text{per\_savecustomerdetailsEmpl}, \text{rl\_employee} \rangle, \\
\langle \text{per\_savecustomerdetailsCust}, \text{rl\_customer} \rangle \} 
\]  

(4.48)

The customer is only authorized to save his own details. To enforce this, a restriction is added that the user id of the customer must match the user id provided in the request message.

\[
\text{requiresParameterOfType} = \{ \langle \text{per\_savecustomerdetailsCust}, \text{uid} \rangle \} 
\]  

(4.49)

The web service is invoked by customer \text{John}:

\[
\text{makesRequest} = \{ \langle \text{John}, \text{request1} \rangle \} 
\]

\[
\text{invokes} = \{ \langle \text{request1}, \text{ws\_savecustomerdetails} \rangle \} 
\]  

(4.50)\hspace{1cm}(4.51)

The subject has a user id assigned as a subject attribute. This identifies the subject:

\[
\text{attributeAssignment} = \{ \langle \text{subject}, \text{uid\_john} \rangle \} 
\]  

(4.52)

The subject is provided the business role customer:

\[
\text{hasBusinessRole} = \{ \langle \text{John}, \text{Customer} \rangle \} 
\]  

(4.53)

This gives the subject an extra subject attribute:

\[
\text{consistsoff} = \{ \langle \text{Customer}, \text{rl\_customer} \rangle \} 
\]  

(4.54)

The subject attributes of the subject are collected in \text{hasSubjectAttributes}:

\[
\text{hasSubjectAttributes} = \{ \langle \text{John}, \text{rl\_customer} \rangle, \\
\langle \text{John}, \text{uid\_john} \rangle \} 
\]  

(4.55)
If subject *John* invokes the web service *ws_savecustomerdetails*, the subject must be authorized for the linked policy:

\[
\text{isAuthorizedForPolicy} = \{(\text{John}, \text{per\_savecustomerdetailsCust})\}\]  
(4.56)

To get a permission, the subject must have all subject attributes linked to the permission in its possession. If the permission requires a parameter, the subject must also provide a parameter matching a subject attribute’s value in its possession:

\[
\text{hasPermission} = \{(\text{John}, \text{per\_savecustomerdetailsCust})\}\]  
(4.57)

So far, the service call by the subject is defined without any parameter. Since permission *per\_savecustomerdetailsCust* requires a parameter (defined in 4.49), all rules but rule 4.24 are true. Because rule 4.24 is false, the subject is not authorized to invoke the web service.

To be authorized to invoke the web service, the subject needs to provide a valid parameter of type *uid* matching a subject attribute of this same type in his possession:

\[
\text{hasParameter} = \{(\text{request1}, \text{uid\_john})\}\]  
(4.58)

\[
\text{hasValidParameter} = \{(\text{request1}, \text{uid\_john})\}\]  
(4.59)

\[
\text{hasValidParameter} = \{(\text{request1}, \text{uid\_john})\}\]  
(4.60)

The adding of the parameter to the request causes the state of 4.24 to flip to the boolean value *true*. This indicates this request, made by subject *John*, is authorized. Instead of providing no parameter or a valid parameter, the subject can also provide a parameter that does not match with any subject attribute:

\[
\text{hasParameter} = \{(\text{request1}, \text{uid\_richard})\}\]  
(4.61)

\[
\text{hasValidParameter} = \{(\text{request1}, \text{uid\_richard})\}\]  
(4.62)

This will cause rule 4.27 to be invalid, since the composition of *makesRequest* and *hasParameter* is equal to *(John, uid\_richard)* and this is not included in the relation *hasSubjectAttributes* which contains *(John, uid\_john)*.

The scenario above shows us that the rules are true if the subject provides a valid parameter to the request and the rules are flagged false in case an invalid or no parameter is provided. This leads to the conclusion that the Access Control Metamodel for Web Service-Oriented Architecture succeeds in authorizing requests based on subject attributes and parameters in the scenario of a single web service invocation.

34
The same web service is used in scenario 2. But unlike in scenario 1, the web service is now used as a dependant web service in an orchestration and is invoked by the web service CreateAccount. Let us assume the web service CreateAccount does not require the input parameter, because the input parameter used for the invocation of SaveCustomerDetails is retrieved from the invocation of the web service FindCustomerDetails.

First we define the web services in the orchestration including their policies:

\[
policyAssignment = \{(ws\_findcustomerdetails, pol\_findcustomerdetails), (ws\_savecustomerdetails, pol\_savecustomerdetails), (ws\_createaccount, pol\_createaccount)\} \tag{4.63}
\]

The web service CreateAccount is dependant on the web services FindCustomerDetails and SaveCustomerDetails:

\[
isDependantOff = \{(pol\_createaccount, ws\_findcustomerdetails), (pol\_createaccount, ws\_savecustomerdetails)\} \tag{4.64}
\]

The subject invoking the web service CreateAccount should be authorized for all policies in the orchestration:

\[
isAuthorizedForPolicy = \{(John, pol\_findcustomerdetails), (John, pol\_savecustomerdetails), (John, pol\_createaccount)\} \tag{4.65}
\]

Six permissions are defined, three permissions used by employees and three permissions used by the customers:

\[
grantsAuthorizationTo = \{(per\_findcustomerdetailsCustomer, pol\_findcustomerdetails), (per\_savecustomerdetailsCustomer, pol\_savecustomerdetails), (per\_createaccountCustomer, pol\_createaccount), (per\_findcustomerdetailsEmployee, pol\_findcustomerdetails), (per\_savecustomerdetailsEmployee, pol\_savecustomerdetails), (per\_createaccountEmployee, pol\_createaccount)\} \tag{4.66}
\]
The permission intended for authorizing employees requires the subject attribute of the business role Employee. The permissions intended for authorizing the customers require the subject attribute of the business role Customer:

\[
\text{requires} = \{ \langle \text{per\_findcustomerdetailsCustomer}, \text{rl\_customer} \rangle, \\
\langle \text{per\_savecustomerdetailsCustomer}, \text{rl\_customer} \rangle, \\
\langle \text{per\_createaccountCustomer}, \text{rl\_customer} \rangle, \\
\langle \text{per\_findcustomerdetailsEmployee}, \text{rl\_employee} \rangle, \\
\langle \text{per\_savecustomerdetailsEmployee}, \text{rl\_employee} \rangle, \\
\langle \text{per\_createaccountEmployee}, \text{rl\_employee} \rangle \}\quad (4.67)
\]

The web service CreateAccount is invoked by the subject John. The subject John is identified by the authentication system as a subject with the business role Customer and the user id provided is \text{uid\_john}.

\[
\text{makesRequest} = \{ \langle \text{John}, \text{request}2 \rangle \}\quad (4.68)
\]
\[
\text{invokes} = \{ \langle \text{request}2, \text{ws\_createaccount} \rangle \}\quad (4.69)
\]
\[
\text{hasBusinessRole} = \{ \langle \text{John}, \text{Customer} \rangle \}\quad (4.70)
\]
\[
\text{consistsOf} = \{ \langle \text{Customer, rl\_customer} \rangle \}\quad (4.71)
\]
\[
\text{attributeAssignment} = \{ \langle \text{John, uid\_john} \rangle \}\quad (4.72)
\]
\[
\text{hasSubjectAttributes} = \{ \langle \text{John, rl\_customer} \rangle, \\
\langle \text{John, uid\_john} \rangle \}\quad (4.73)
\]

Each subject attribute is of a certain type:

\[
\text{isTypeOfS} = \{ \langle \text{uid\_john, uid} \rangle, \\
\langle \text{rl\_customer, role} \rangle, \\
\langle \text{rl\_employee, role} \rangle \}\quad (4.74)
\]

To be authorized for a policy, the subject must have at least one permission linked to the policy. Based on the subject attributes, the subject John has got the following permissions:

\[
\text{hasPermission} = \{ \langle \text{John, per\_createaccountcustomer} \rangle, \\
\langle \text{John, per\_findcustomerdetailescustomer} \rangle, \\
\langle \text{John, per\_savecustomerdetailscustomer} \rangle \}\quad (4.75)
\]

Like in the first scenario, the permission \text{per\_savecustomerdetailscustomer} requires an input parameter of type \text{uid}. This parameter is used to ensure a customer is only allowed to save his own details:

\[
\text{requiresParameterOfType} = \{ \langle \text{per\_savecustomerdetailsCustomer, uid} \rangle \}\quad (4.76)
\]
Only in this scenario there is no parameter supplied in the message sent to the web service `CreateAccount`. This has as a result that the subject does not have the permission `per_savecustomerdetailscustomer`, which leads to a falsification of rule \( 4.24 \). To ensure rule \( 4.24 \) is true, the pair \((John, per\_savecustomerdetailscustomer)\) has to be removed from \( 4.79 \). This leads to the falsification of rule \( 4.28 \). In rule \( 4.28 \), the relation is defined between the subject and the policies the subject is authorized for. If the pair \((John, pol\_savecustomerdetails)\) is removed from this relation, rule \( 4.29 \) raises an error indicating the subject is not authorized for the dependent web services of `pol\_createaccount`. Removing this pair will eventually falsificate rule \( 4.30 \) that states that the subject is not authorized to invoke the web service `CreateAccount`.

The only way to grant the subject access to the web service `CreateAccount`, is to extend the policy `pol\_savecustomerdetails` with an extra permission that does not require a parameter. The relation `requires` defined in \( 4.67 \) will then be:

\[
\text{requires} = \{(\text{per\_findcustomerdetailsCustomer}, \text{rl\_customer}), \\\n(\text{per\_savecustomerdetailsCustomer}, \text{rl\_customer}), \\\n(\text{per\_savecustomerdetailsCustomer2}, \text{rl\_customer}), \\\n(\text{per\_createaccountCustomer}, \text{rl\_customer}), \\\n(\text{per\_findcustomerdetailsEmployee}, \text{rl\_employee}), \\\n(\text{per\_savecustomerdetailsEmployee}, \text{rl\_employee}), \\\n(\text{per\_createaccountEmployee}, \text{rl\_employee})\} \quad (4.77)
\]

The new permission has to be linked to the policy `pol\_savecustomerdetails`. 

---

37
The relation `grantsAuthorizationTo` defined in 4.66 will have to change to:

\[
grantsAuthorizationTo = \{(\text{per}\_\text{findcustomerdetails}\text{Customer}, \text{pol}\_\text{findcustomerdetails}) , \\
(\text{per}\_\text{savecustomerdetails}\text{Customer}, \text{pol}\_\text{savecustomerdetails}) , \\
(\text{per}\_\text{savecustomerdetails}\text{Customer2}, \text{pol}\_\text{savecustomerdetails}) , \\
(\text{per}\_\text{createaccount}\text{Customer}, \text{pol}\_\text{createaccount}) , \\
(\text{per}\_\text{findcustomerdetails}\text{Employee}, \text{pol}\_\text{findcustomerdetails}) , \\
(\text{per}\_\text{savecustomerdetails}\text{Employee}, \text{pol}\_\text{savecustomerdetails}) , \\
(\text{per}\_\text{createaccount}\text{Employee}, \text{pol}\_\text{createaccount})\} \tag{4.78}
\]

Now, the subject has a permission authorizing him for every policy involved in the request and thus will be authorized for the web service CreateAccount.

\[
\text{hasPermission} = \{(\text{John, per}\_\text{createaccountcustomer}) , \\
(\text{John, per}\_\text{findcustomerdetailscustomer}) , \\
(\text{John, per}\_\text{savecustomerdetailscustomer2})\} \tag{4.79}
\]

However, this leads to a security risk. As quoted from the article “An Access Control Metamodell for Web Service-Oriented Architecture” [8]:

The **Policy** is the composition of all **Permissions** of an **Object** using Boolean “OR” concatenation.

This means the least restrictive permission linked to the permission is used to authorize the subject for a policy. With the adding of the permission `per\_\text{savecustomerdetails}\text{Customer2}` the subject is also authorized in scenario 1 with a parameter of another subject:

\[
\text{hasParameter} = \{(\text{request1, uid}\_\text{john})\} \tag{4.80}
\]

\[
\text{hasValidParameter} = \{} \tag{4.81}
\]

The request does have a parameter, although this one is not in the relation `hasValidParameter`. Still, all rules are valid because of the extra permission added.
The conclusion is that the Metamodel handles authorization based on input parameters well, but only on single web services. The last scenario shows us that making the authorization decision based on input parameters may introduce a security vulnerability by making the permissions less restrictive when these services are used in an orchestration of web services. This claim proves to be not valid.

4.2.3 Claim Limit Service Calls

One goal the authors tried to achieve with the Access Control Metamodel for Web Service-Oriented Architecture, is avoiding unnecessary service invocations. In section 4.2.1 it is already verified that the Access Control Metamodel for Web Service-Oriented Architecture succeeds in validating the authorizations of all services in a web service orchestration. This leads to a certain performance gain. No unnecessary service calls utilize network bandwidth and server resources. The main benefit they achieved is that there is no need for compensating service calls when transactions have to be aborted because of an authorization failure. Therefore I conclude claim Limit Service Calls is valid.

4.2.4 Claim Scaling

The goal of this claim, according to the authors, is to offer a better scaling solution than the general existing authorization models currently in use. The Access Control Metamodel for Web Service-Oriented Architecture is based on the attribute-based access control (ABAC) model introduced by Eric Yuan and Jin Tong [22]. In ABAC, permissions are based merely on security based characteristics of objects and subjects. This prevents the unnecessary addition of subjects and roles [8, 22].

Since business rules cannot validate scalability, the validity of the claim Scaling cannot be determined using Ampersand. So, I found my research method is not sufficient to examine the validation of this claim. Therefore I cannot rule on whether or not this claim is valid.
5 Other drawbacks of the Metamodel

In examining the Metamodel, I encountered some drawbacks. These are described in the following sections.

5.1 Separation of concerns

In the software industry the *Separation of concerns* theory is based on the notion that it is beneficial to break down large problems into a series of individual concerns. This allows the logic required to solve the problem to be decomposed into a collection of smaller, related pieces. Each piece of logic addresses a specific problem. Service orientation can be viewed as a distinct manner in which to realize a separation of concerns. The principles of service orientation provide the means of supporting this theory, while achieving a foundation paradigm upon which many contemporary Service-Oriented Architecture characteristics can be built. Thomas Erl has identified and defined a set of principles for service-Oriented Architecture. One of these principles is *Services are autonomous*. Erl describes this principle as:

> The logic governed by a service resides within an explicit boundary. The service has control within this boundary, and is not dependent on other services for it to execute its governance.

In some cases services contain business logic to validate whether the subject is authorized to execute an action based on input parameters. For example, a bank customer is authorized to view bank account information, with the restriction he can only get information about his own account. This kind of logic is a fundamental part of the business logic of the service. During development this kind of logic can be tested using unit tests.

In the approach of Emig et al. this business logic is outsourced to an external system. This leaves no control to the service itself. Or even worse, the control is administrated in two systems. These systems are the policy decision service and the web service itself. This violates the view that a service is self-contained and autonomous.

5.2 Choreographical web service composition

Another drawback of the Access Control Metamodel for Web Service-Oriented Architecture is that only explicit service composition is handled, i.e. only web services part of an orchestrated composition are verified. This model does not handle web service compositions based on a choreography. This does not mean a choreography scenario can not be handled by this model, but it is difficult to administer it in a consistent manner.
6 Conclusion

In my research I analyzed the added value of the Access Control Metamodel for Web Service-Oriented Architecture by presenting it in business rules. To do this I extracted four claims made by the authors of “Access Control Metamodel for Web Service-Oriented Architecture” \cite{Emig2013} and demonstrated their validity using the Ampersand method.

First I researched claim 1, Explicit Composition. My research showed that the rules were violated when the subject was not in the possession of all required attributes needed for the web services that are part of the orchestration. When the subject did have all required subject attributes none off the rules were violated. This implicates the model only allows the subject to invoke the orchestration when the subject is authorized to do so. This concludes claim 1, Explicit Composition is valid.

Next, I validated claim 2, Support Parameters. Based on my translation to Ampersand, I found that the Access Control Metamodel for Web Service-Oriented Architecture handles authorization based on input parameters well, but only on single web services. However, introducing input parameters in the authorization decision process, causes a security vulnerability by making the permissions less restrictive when these services are used in an orchestration of web services. This claim proves to be not valid.

The research shows that claim 1, Explicit Composition and claim 2, Support Parameters are conflicting claims. It is demonstrated that the involvement of parameters in a authorization decision is only possible if no orchestrational composition is supported. It seems like both claims could be valid if tested in separate test scenarios. Combined in one test scenario however, they can not both be valid.

Thirdly, I validated claim 3, Limit Service Calls. Since the model can determine the authorization decision for all services used in an orchestration, no other services are unnecessarily invoked. This concludes claim 3, Limit Service Calls is valid.

Finally, I tried to validate claim 4, Scaling. Since the Ampersand method is not suitable to validate scalability, I was not able to rule on whether or not this claim is valid.

Note that my conclusions about the claims being valid or not, are based on my interpretations of the article by Emig et al.. Since the Access Control Metamodel for Web Service-Oriented Architecture is not described in formal functional specifications, it will be possible that other researchers find different business rules and might draw other conclusions.
Based on my research, I find the Access Control Metamodel for Web Service-Oriented Architecture a welcome addition to the existing authorization models. The model fully supports the feature of the Service-Oriented Architecture to (re)use various services to facilitate new business processes. The ability to determine the authorization decision before the execution of the first service in an orchestration, prevents unnecessary service calls and roll-backs of actions due to access violations.

However, in the present form of the Access Control Metamodel for Web Service-Oriented Architecture, I do not recommend the use of input parameters to determine the authorization decision for two reasons. The most important reason is it can lead to serious security vulnerabilities. The second reason is that it can lead to the spread of business logic over the web service itself and the authorization mechanism. The security flaw I demonstrated in chapter 4.2.2 undermines the purpose of the Access Control Metamodel for Web Service-Oriented Architecture.

As for his comment on my findings, Christian Emig remarks:

The process of enhancing the “policy” with further “permissions” especially with less restrictive ones, needs to be carefully revised. As you showed in your example, this can be a door-opener for unauthorized usage. As this process is manually done, there should be (automated) help for the developer to be warned in such cases. There definitely is need to concentrate on policy conflicts.

During my research I found a few drawbacks of the Access Control Metamodel for Web Service-Oriented Architecture. One them being the lack of supporting choreographical web service compositions. An authorization model for Web Service-Oriented Architecture that does support both kind of compositions, would be an improvement of the Access Control Metamodel for Web Service-Oriented Architecture.

Starting my research, I wondered whether the Access Control Metamodel for Web Service-Oriented Architecture was feasible in practice. The validation of access to all services in an orchestration is a problem I have encountered in my work as an IT Architect. So I do understand the problem the authors of the Metamodel tried to solve. In my opinion they partially succeeded. Despite the benefits, using the Metamodel provides us with a few security problems. I recommend further research to be done on authorization models for Web Service-Oriented Architecture. An access management system that would support both orchestrellational and choreographical compositions would, in my opinion, meet the intended flexibility of a Web Service-Oriented Architecture.
References

Web services description language (wsdl) 1.1.
IBM, march 2004.
NOTE.

Business process execution language for web services (version 1.1.

Web services, authentication and identity management.

Simple object access protocol (soap) 1.1.
W3C, 05 2000.

Migrating to a service-oriented architecture.

A simple and practical approach to unit testing: The jml and junit way,

Web services description language (wsdl) 1.1.
W3C, 03 2001.
NOTE.

An access control metamodel for web service-oriented architecture.
icsea, 0:57, 2007.

Core principles for service-oriented architectures.
http://www.looselycoupled.com/opinion/2005/erl-core-infr0815.html,
08 2005.

Role-based access controls.  


Realizing model driven security for inter-organizational workflows with ws-cdl and uml 2.0.  

Deriving functional specifications from business requirements with ampersand.  
2008.

Fulfilling the web services promise.  

Dynamic protection structures.  

Service oriented architecture: a new paradigm for enterprise application integration.  

Using encryption for authentication in large networks of computers.  

[19] OASIS.  

Service oriented architectures: Approaches, technologies and research issues.  

*Understanding Enterprise SOA*.  
[22] Eric Yuan and Jin Tong.
Attributed based access control (abac) for web services.
{
icws

Elements of service-oriented analysis and design, June 2004.
A  ADL Claim 1, Explicit Composition

A.1  Validation of claim 1, Explicit Composition, using ADL

The ADL source in appendix A.2 is used to test the validation of claim 1, Explicit Composition, in the ADL tool Atlas. To steps below can be used to reproduce the validation of this claim

• Load the ADL source in ATLAS.
  This is the source using the success scenario in chapter 4.2.1

• Compile the source

• The source is now succesfully loaded and no violations are shown (see figure 5).

![Figure 5: No violations in success scenario](image)

• Now remove the comment in the population
  ```
  --; ("per_savecustomerdetails", "rl_manager")
  ```
  This will add the subjectattribute rl_manager the the permission per_savecustomerdetails.

• Reload the source and compile the source again

• Now Atlas will show a violation of rule 4.24 (see figure 6.)
A.2 Source

The source below contains the ADL of the business rules in chapter 4.1 and the population of the relations in chapter 4.2.1.

-- Description : Access Metamodel for Webservice Oriented Architecture verification.

CONTEXT WSOAClaim1

PATTERN GrantAuthorization

CONCEPT BusinessRole "A business role"
CONCEPT Permission "A permission grants access to a policy"
CONCEPT Request "An invocation of the (primary) webservice by the subject"
CONCEPT Subject "A person or system who wants to have access to an Object"
CONCEPT SubjectAttribute "An attribute containing information about a subject relevant for authorization"
CONCEPT SubjectAttributeType "Describes the type of the subject attribute type needed for validation"
CONCEPT Variable "A parameter send with a request to the webservice"
CONCEPT WebService "The object that has to be protected"

isTypeOfS :: SubjectAttribute * SubjectAttributeType [UNI, TOT]
PRAGMA "" " is of type "

= ["uid_john", "uid"]
; ["rl_customer", "role"]
; ["rl_manager", "role"]
]

attributeAssignment :: Subject * SubjectAttribute
hasBusinessRole :: Subject * BusinessRole
PRAGMA "" " has business role "
= [ ('John','Customer') ].

includes :: BusinessRole * BusinessRole
PRAGMA "" " includes business role"
= [].

hasBusinessRole;includes |- hasBusinessRole
EXPLANATION "A subjects inherits all businessroles the businessrole has"

consistsOff :: BusinessRole * SubjectAttribute [TOT]
PRAGMA "" " gives subjectattributes "
= [ ('Customer','rl_customer')
; ('Manager','rl_manager') ].

hasSubjectAttributes :: Subject * SubjectAttribute [TOT]
PRAGMA "" " has subject attribute "
= [ ('John','rl_customer')
; ('John','uid_john') ].

hasSubjectAttributes = attributeAssignment \ hasBusinessRole;consistsOff
EXPLANATION "A hasSubjectAttributes is the collection of individual attributes
and attributes assigned by businessrole"

ENDPATTERN

PATTERN DefineWebServices

policyAssignment :: Webservice * Policy [UNI,TOT,INJ,SUR]
PRAGMA "" " requires policy "
= [ 

50
isDependantOff :: Policy * Webservice
PRAGMA "" " is dependant of webservice "
= [
("pol_createaccount","ws_findcustomerdetails")
; ("pol_createaccount","ws_savecustomerdetails")
].

isAuthorizedForPolicy :: Subject * Policy
PRAGMA "" " is authorized for policy "
= [
("John","pol_createaccount")
; ("John","pol_findcustomerdetails")
; ("John","pol_savecustomerdetails")
].

grantsAuthorizationTo :: Permission * Policy [TOT,SUR]
PRAGMA "" " grants authorization to "
= [
("per_findcustomerdetails","pol_findcustomerdetails")
; ("per_savecustomerdetails","pol_savecustomerdetails")
; ("per_createaccount","pol_createaccount")
].

requires :: Permission * SubjectAttribute
PRAGMA "" " requires subjectattribute "
= [
("per_findcustomerdetails", "rl_customer")
; ("per_savecustomerdetails", "rl_customer")
--; ("per_savecustomerdetails", "rl_manager")
; ("per_createaccount", "rl_customer")
].

requiresEnvironment :: Permission * Variable
PRAGMA "" " requires the system variable ".

hasPermission :: Subject * Permission
PRAGMA "" " has all required subjectattributes and extended parameters for "
= [
("John","per_createaccount")
; ("John","per_findcustomerdetails")
; ("John","per_savecustomerdetails")
].

hasEnvironment :: Permission * Variable.

ENDPATTERN

PATTERN MakeRequest

makesRequest :: Subject * Request[INJ,SUR]
PRAGMA "" " makes the request "
= [ ("John","request1") ].

invokes :: Request * Webservice [UNI,TOT]
PRAGMA "" " invokes webservice "
= [ ("request1","ws_createaccount") ].

ENDPATTERN

PATTERN Parameter

requiresParamaterOfType :: Permission * SubjectAttributeType
PRAGMA "" " requires a parameter in the request of type "
= [ ].

hasParameter :: Request * SubjectAttribute
PRAGMA "" " has a parameter with the value of "
= [ ].

hasPermission~ = hasPermission~ /
((makesRequest;invokes;policyAssignment;grantsAuthorizationTo~)~)
EXPLANATION "To have a permission that requires parameters of a certain type"
makesRequest;hasParameter |- hasSubjectAttributes
EXPLANATION "If a parameter is used for authorization, the subject must have the corresponding subject attribute"

ENDPATTERN

PATTERN AttributeValidation

isAuthorizedForPolicy;isDependantOff;pollyAssignment |- isAuthorizedForPolicy
EXPLANATION "A subject is authorized for a policy if the subject is also authorized for all dependant webservice"

(makesRequest;invokes)~ = -((policyAssignment) ; (-isAuthorizedForPolicy~)) /
\ (makesRequest;invokes)~
EXPLANATION "Access to a webservice is granted if the subject has the policy coupled to the Webservice"

isAuthorizedForPolicy = hasPermission;grantsAuthorizationTo
EXPLANATION "A subject is authorized for a policy if it has at least one permission linked to the policy"

hasPermission~ = (-( requires ; -(hasSubjectAttributes~) ) ) /
\ (-( requiresParamaterOfType ;
   -( (makesRequest;hasParameter;isTypeOfS~))) ) /
\ ((makesRequest;invokes;pollyAssignment;
   grantsAuthorizationTo~)) /
\ (makesRequest;invokes;pollyAssignment;
   isDependantOff;pollyAssignment;
   grantsAuthorizationTo~) ~) /
\ hasPermission~
EXPLANATION "The request contains all required parameters for the permissions the subject is authorized for by attributes"

ENDPATTERN
ENDCONTEXT
B ADL Claim 2

B.1 Validation of claim 2, Support Parameters, using ADL

The source of [B.2.1] contains a scenario with the web service \textit{ws\_savecustomerdetails} directly invoked by the subject. The policy of the web service requires a permission with a subject attribute and parameter that must be equal to a subject attribute in the possession of the subject. This will restrict the subject to only save his own details.

The source of [B.2.2] contains the same web service but know the subject is invoking the web service indirectly via another web service \textit{ws\_createaccount}. The invocation of the first web service does not contain the required parameter. This will create a violation because the permission \textit{per\_savecustomerdetailsCustomer} requires a parameter. This can be tested using ADL, see figure 7.

![Figure 7: Violation of rule 4.24 in an orchestration.](image)

To grant the subject access to the orchestration another permission, \textit{per\_savecustomerdetailsCustomer2}, is added. This permission only requires a subject attribute and no parameter. This allows the subject to invoke the web service \textit{ws\_savecustomerdetails} indirectly.

To test this in ADL:

- Uncomment the lines with \textit{per\_savecustomerdetailsCustomer2}
- Remove the line ("John","per\_savecustomerdetailsCustomer") -- REMOVE THIS LINE
- Recompile the source
- Check ADL does not show any violations.

The source of [B.2.3] contains the newly created permission, allowing a successful orchestration. However, in this case the web service \textit{ws\_savecustomerdetails} is invoked directly again. Only this time without the parameter in the request used for validation during the access control decision. If this script is compiled in ADL, no violations are shown.
This shows the possibility, when a web service is secured using permission restricted with parameters and reused in an orchestration, the access to the web services is less restrictive than initially intended.

B.2 Sources

B.2.1 Parameters in a single web service

-- Description : Access Metamodel for Webservice Oriented Architecture verification.

CONTEXT WSOACLAIM2

PATTERN GrantAuthorization

isTypeOfS :: SubjectAttribute * SubjectAttributeType [UNI, TOT]
PRAGMA "" " is of type "
= [  
  ("uid_john", "uid")  
  ; ("rl_customer", "role")  
  ; ("rl_employee", "role")  
].

attributeAssignment :: Subject * SubjectAttribute
PRAGMA "" " has subject attribute"
= [  
  ("John", "uid_john")  
].

Figure 8: No violations when invoking ws_savecustomerdetails without parameters.
hasBusinessRole :: Subject * BusinessRole
PRAGMA "" " has business role "
= [ ('John',"Customer") ].

includes :: BusinessRole * BusinessRole
PRAGMA "" " includes business role"
= [].

hasBusinessRole;includes |- hasBusinessRole
EXPLANATION "A subject inherits all businessroles the businessrole has"

consistsOff :: BusinessRole * SubjectAttribute [TOT]
PRAGMA "" " gives subjectattributes "
= [ ('Customer','rl_customer') ].

hasSubjectAttributes :: Subject * SubjectAttribute[TOT]
PRAGMA "" " has subject attribute "
= [ ('John','rl_customer') ; ('John','uid_john') ].

hasSubjectAttributes = attributeAssignment \/ hasBusinessRole;consistsOff
EXPLANATION "A hasSubjectAttributes is the collection of individual
attributes and attributes assigned by businessrole"
ENDPATTERN

PATTERN DefineWebServices

policyAssignment :: Webservice * Policy [UNI,TOT,INJ,SUR]
PRAGMA "" " requires policy "
= [ ('ws_findcustomerdetails','pol_findcustomerdetails')
; ('ws_savecustomerdetails','pol_savecustomerdetails')
; ('ws_createaccount','pol_createaccount') ].
isDependantOff :: Policy * Webservice
PRAGMA "" " is dependant of webservice "
= [
  ("pol_createaccount","ws_findcustomerdetails")
; ("pol_createaccount","ws_savecustomerdetails")
].

isAuthorizedForPolicy :: Subject * Policy
PRAGMA "" " is authorized for policy "
= [
  ("John","pol_savecustomerdetails")
].

grantsAuthorizationTo :: Permission * Policy [TOT,SUR]
PRAGMA "" " grants authorization to "
= [
  ("per_findcustomerdetailsCustomer","pol_findcustomerdetails")
; ("per_savecustomerdetailsCustomer","pol_savecustomerdetails")
; ("per_savecustomerdetails","pol_savecustomerdetails")
; ("per_createaccount","pol_createaccount")
].

requires :: Permission * SubjectAttribute
PRAGMA "" " requires subjectattribute "
= [
  ("per_findcustomerdetailsCustomer","rl_customer")
; ("per_savecustomerdetailsCustomer","rl_customer")
; ("per_savecustomerdetails","rl_employee")
; ("per_createaccount","rl_customer")
].

requiresEnvironment :: Permission * Variable
PRAGMA "" " requires the system variable ".

hasPermission :: Subject * Permission
PRAGMA "" " has all required subjectattributes and extended parameters for "
= [
  ("John","per_savecustomerdetailsCustomer")
].

hasEnvironment :: Permission * Variable.
ENDPATTERN

PATTERN MakeRequest

makesRequest :: Subject * Request[INJ,SUR]
PRAGMA "" " makes the request "
= [  
  ("John","request1")
].

-- rel:invokes
invokes :: Request * Webservice [UNI,TOT]
PRAGMA "" " invokes webservice "
= [  
  ("request1" ,"ws_savecustomerdetails")
].

ENDPATTERN

PATTERN Parameter

requiresParameterOfType :: Permission * SubjectAttributeType
PRAGMA "" " requires a parameter in the request of type "
= [  
  ("per_savecustomerdetailsCustomer", "uid")
].

hasParameter :: Request * SubjectAttribute
PRAGMA "" " has a parameter used for authorization with the value of "
= [  
  ("request1" , "uid_john")
].

hasPermission~ = hasPermission~ \/
((makesRequest;invokes;policyAssignment; grantsAuthorizationTo~)~ \/

58
(makesRequest;invokes;policyAssignment;isDependantOff;policyAssignment;grantsAuthorizationTo~)~)
EXPLANATION "To have a permission that requires parameters of a certain type"

makesRequest;hasParameter |- hasSubjectAttributes
EXPLANATION "If a parameter is used for authorization, the subject must have the corresponding subject attribute"

ENDPATTERN

PATTERN AttributeValidation

isAuthorizedForPolicy;isDependantOff;policyAssignment |- isAuthorizedForPolicy
EXPLANATION "A subject is authorized for a policy if the subject is also authorized for all dependant webservice"

(makesRequest;invokes)~ = -((policyAssignment) ; (-isAuthorizedForPolicy~)) /
(makesRequest;invokes)~
EXPLANATION "Access to a webservice is granted if the subject has the policy coupled to the Webservice"

isAuthorizedForPolicy = hasPermission;grantsAuthorizationTo
EXPLANATION "A subject is authorized for a policy if it has at least one permission linked to the policy"

hasPermission~ = (-( requires ; -(hasSubjectAttributes~))) /
(-( requiresParameterOfType ; 
-(makesRequest;hasParameter;isTypeOfS~))) /
((makesRequest;invokes;policyAssignment;grantsAuthorizationTo~)~) /
(makesRequest;invokes;policyAssignment;isDependantOff;policyAssignment;grantsAuthorizationTo~)~) /
hasPermission~
EXPLANATION "The request contains all required parameters for the permissions the subject is authorized for by attributes"

ENDPATTERN
ENDCONTEXT
B.2.2 Parameter secured web service in an orchestration

-- Description : Access Metamodel for Webservice Oriented
-- Architecture verification.

CONTEXT WSOACLAIM2

PATTERN GrantAuthorization

isTypeOfS :: SubjectAttribute * SubjectAttributeType [UNI, TOT]
PRAGMA "" " is of type "
= [
("uid_john", "uid")
; ("rl_customer", "role")
; ("rl_employee", "role")
].

attributeAssignment :: Subject * SubjectAttribute
PRAGMA "" " has subject attribute"
= [
("John","uid_john")
].

hasBusinessRole :: Subject * BusinessRole
PRAGMA "" " has business role "
= [
("John","Customer")
].

includes :: BusinessRole * BusinessRole
PRAGMA "" " includes business role"
= [].

hasBusinessRole;includes |- hasBusinessRole
EXPLANATION "A subjects inherits all businessroles the businessrole has"

consistsOf :: BusinessRole * SubjectAttribute [TOT]
PRAGMA "" " gives subjectattributes "
= [
("Customer","rl_customer")
]
hasSubjectAttributes :: Subject * SubjectAttribute[TOT]
PRAGMA "" " has subject attribute "
= [ 
("John","rl_customer")
; ("John","uid_john")
].

hasSubjectAttributes = attributeAssignment \ hasBusinessRole;consistsOf
EXPLANATION "A hasSubjectAttributes is the collection of individual attributes and attributes assigned by businessrole"

ENDPATTERN

PATTERN DefineWebServices

policyAssignment :: Webservice * Policy [UNI,TOT,INJ,SUR]
PRAGMA "" " requires policy "
= [ 
("ws_findcustomerdetails","pol_findcustomerdetails")
; ("ws_savecustomerdetails","pol_savecustomerdetails")
; ("ws_createaccount","pol_createaccount")
].

isDependantOff :: Policy * Webservice
PRAGMA "" " is dependant of webservice "
=[
("pol_createaccount","ws_findcustomerdetails")
; ("pol_createaccount","ws_savecustomerdetails")
].

isAuthorizedForPolicy :: Subject * Policy
PRAGMA "" " is authorized for policy "
= [ 
("John","pol_savecustomerdetails")
; ("John","pol_findcustomerdetails")
; ("John","pol_createaccount")
].

grantsAuthorizationTo :: Permission * Policy [TOT,SUR]
PRAGMA "" " grants authorization to "
= [ 
]
requires :: Permission * SubjectAttribute
PRAGMA "" " requires subjectattribute "
= [ 
("per_findcustomerdetailsCustomer","rl_customer")
--; ("per_savecustomerdetailsCustomer2","rl_customer")
; ("per_savecustomerdetailsCustomer","rl_customer")
; ("per_savecustomerdetails","rl_employee")
; ("per_createaccount","rl_customer")
].

requiresEnvironment :: Permission * Variable
PRAGMA "" " requires the system variable ".

hasPermission :: Subject * Permission
PRAGMA "" " has all required subjectattributes and extended parameters for "
= [ 
--("John","per_savecustomerdetailsCustomer2")
("John","per_savecustomerdetailsCustomer") -- REMOVE THIS LINE
; ("John","per_findcustomerdetailsCustomer")
; ("John","per_createaccount")
].

hasEnvironment :: Permission * Variable.

ENDPATTERN

PATTERN MakeRequest

makesRequest :: Subject * Request[INJ,SUR]
PRAGMA "" " makes the request "
= [ 
("John","request2")
].

-- rel:invokes
invokes :: Request * Webservice [UNI,TOT]
PRAGMA "" " invokes webservice "

62
ENDPATTERN

PATTERN Parameter

requiresParamaterOfType :: Permission * SubjectAttributeType
PRAGMA "" " requires a parameter in the request of type "
= [ (~"per_savecustomerdetailsCustomer", "uid") ].

hasParameter :: Request * SubjectAttribute
PRAGMA "" " has a parameter with the value of "
= [
  ~("request2", "uid_john") ].

hasPermission~ = hasPermission~ /
  (makesRequest;invokes;policyAssignment;
    grantsAuthorizationTo~)~ /
  (makesRequest;invokes;policyAssignment;isDependantOff;
    policyAssignment;grantsAuthorizationTo~)

EXPLANATION "To have a permission that requires parameters of a certain type"

makesRequest;hasParameter |- hasSubjectAttributes
EXPLANATION "If a parameter is used for authorization, the subject must have
the corresponding subject attribute"

ENDPATTERN

PATTERN AttributeValidation

isAuthorizedForPolicy;isDependantOff;policyAssignment |- isAuthorizedForPolicy
EXPLANATION "A subject is authorized for a policy if the subject is also
authorized for all dependant webservice"

(makesRequest;invokes)~ = -((policyAssignment); (~isAuthorizedForPolicy~))
\ (makesRequest;invokes)~

EXPLANATION "Access to a webservice is granted if the subject has the policy
coupled to the Webservice"
isAuthorizedForPolicy = hasPermission;grantsAuthorizationTo
EXPLANATION "A subject is authorized for a policy if it has at least one permission linked to the policy"  

hasPermission" = (-( requires ; -(hasSubjectAttributes~)) ) /
 (-( requiresParameterOfType ;
   -(requires;isTypeOfS~))) /
 ((makesRequest;invokes;policyAssignment;
  grantsAuthorizationTo~)~ /
   (makesRequest;invokes;policyAssignment;isDependantOff;
    policyAssignment;grantsAuthorizationTo~)~) /
 hasPermission"
EXPLANATION "The request contains all required parameters for the permissions the subject is authorized for by attributes"

ENDPATTERN
ENDCONTEXT

B.2.3 Parameter secured web service with less restrictive permission

-- Description : Access Metamodel for Webservice Oriented Architecture verification.

CONTEXT WSOACLAIM2

PATTERN GrantAuthorization

isTypeOfS :: SubjectAttribute * SubjectAttributeType [UNI, TOT]  
PRAGMA "" " is of type "  
= [  
  ("uid_john", "uid")  
; ("rl_customer", "role")  
; ("rl_employee", "role")  
].

attributeAssignment :: Subject * SubjectAttribute  
PRAGMA "" " has subject attribute"  
= [  
  ("John"."uid_john")  
].
hasBusinessRole :: Subject * BusinessRole
PRAGMA "" " has business role "
= [ ("John","Customer") ].

includes :: BusinessRole * BusinessRole
PRAGMA "" " includes business role"
= [].

hasBusinessRole;includes |- hasBusinessRole
EXPLANATION "A subjects inherits all businessroles the businessrole has"

consistsOff :: BusinessRole * SubjectAttribute [TOT]
PRAGMA "" " gives subjectattributes "
= [ ("Customer","rl_customer") ].

hasSubjectAttributes :: Subject * SubjectAttribute[TOT]
PRAGMA "" " has subject attribute "
= [ ("John","rl_customer") ; ("John","uid_john") ].

hasSubjectAttributes = attributeAssignment \ hasBusinessRole;consistsOff
EXPLANATION "A hasSubjectAttributes is the collection of individual attributes and attributes assigned by businessrole"

ENDPATTER

PATTERN DefineWebServices

policyAssignment :: Webservice * Policy [UNI,TOT,INJ,SUR]
PRAGMA "" " requires policy "
= [ ("ws_findcustomerdetails","pol_findcustomerdetails") ; ("ws_savECustomerdetails","pol_savECustomerdetails") ; ("ws_creatEaccount","pol_creatEaccount") ].

isDependantOff :: Policy * Webservice
PRAGMA "" " is dependant of webservice "
=[
  ("pol_createaccount","ws_findcustomerdetails")
; ("pol_createaccount","ws_savecustomerdetails")
].

isAuthorizedForPolicy :: Subject * Policy
PRAGMA "" " is authorized for policy "
= [
  ("John","pol_savecustomerdetails")
].

grantsAuthorizationTo :: Permission * Policy [TOT,SUR]
PRAGMA "" " grants authorization to "
= [
  ("per_findcustomerdetailsCustomer","pol_findcustomerdetails")
; ("per_savecustomerdetailsCustomer","pol_savecustomerdetails")
; ("per_savecustomerdetailsCustomer","pol_savecustomerdetails")
; ("per_savecustomerdetails","pol_savecustomerdetails")
; ("per_createaccount","pol_createaccount")
].

requires :: Permission * SubjectAttribute
PRAGMA "" " requires subjectattribute "
= [
  ("per_findcustomerdetailsCustomer","rl_customer")
; ("per_savecustomerdetailsCustomer","rl_customer")
; ("per_savecustomerdetailsCustomer","rl_customer")
; ("per_savecustomerdetails","rl_employee")
; ("per_createaccount","rl_customer")
].

requiresEnvironment :: Permission * Variable
PRAGMA "" " requires the system variable ".

hasPermission :: Subject * Permission
PRAGMA "" " has all required subjectattributes and extended parameters for "
= [
  ("John","per_savecustomerdetailsCustomer2")
].
hasEnvironment :: Permission * Variable.

ENDPATTERN

PATTERN MakeRequest

makesRequest :: Subject * Request[INJ,SUR]
PRAGMA "" " makes the request ":
= [
("John","request3")
].

-- rel:invokes
invokes :: Request * Webservice [UNI,TOT]
PRAGMA "" " invokes webservice ":
= [
("request3","ws_savecustomerdetails")
].

ENDPATTERN

PATTERN Parameter

requiresParameterOfType :: Permission * SubjectAttributeType
PRAGMA "" " requires a parameter in the request of type ":
= [
("per_savecustomerdetailsCustomer","uid")
].

hasParameter :: Request * SubjectAttribute
PRAGMA "" " has a parameter with the value of ":
= [
--("request2","uid_john")
].

hasPermission~ = hasPermission~ /
((makesRequest;invokes;policyAssignment;
grantsAuthorizationTo~)~ /
(makesRequest;invokes;policyAssignment;isDependantOff;
policyAssignment;grantsAuthorizationTo~)~
EXPLANATION "To have a permission that requires parameters of a certain type"
makesRequest;hasParameter |- hasSubjectAttributes
EXPLANATION "If a parameter is used for authorization, the subject must have
the corresponding subject attribute"
ENDPATTERN

UTILITY AttributeValidation
PATTERN AttributeValidation

isAuthorizedForPolicy;isDependantOff;policyAssignment |- isAuthorizedForPolicy
EXPLANATION "A subject is authorized for a policy if the subject is also
authorized for all dependant webservices"

(makesRequest;invokes)~ = -((policyAssignment); (-isAuthorizedForPolicy~))
\/(makesRequest;invokes)~
EXPLANATION "Access to a webservice is granted if the subject has the
policy coupled to the Webservice"

isAuthorizedForPolicy = hasPermission;grantsAuthorizationTo
EXPLANATION "A subject is authorized for a policy if it has at least one
permission linked to the policy"

hasPermission~ = (-( requires ; -(hasSubjectAttributes~)) ) \/
( -( requiresParamaterOfType ;
   -((makesRequest;hasParameter;isTypeOfS~)))) /\ ((makesRequest;invokes;policyAssignment;
   grantsAuthorizationTo~)~ \/
   (makesRequest;invokes;policyAssignment;isDependantOff;
   policyAssignment;grantsAuthorizationTo~)~) ~
\ hasPermission~
EXPLANATION "The request contains all required parameters for the permissions
the subject is authorized for by attributes"

ENDPATTERN
ENDCONTEXT