DYNAMIC CONTEXT-AWARE ACCESS CONTROL
Use of Resource Hierarchies to Define Fine-grained, Adaptable Authorization Policies

Annett Laube and Laurent Gomez
SAP Research, SAP Labs France SA, 06250 Mougins, France

Keywords: Access control, Context Awareness, Resource Hierarchy.

Abstract: Complex access control rules often interfere with the business logic within applications. We show a solution based on strict separation of application and security logic that allows dynamic policy enforcement based on context-information as well as the adaptation of granularity outside the applications. The definition of resource hierarchies driven by application needs and related authorization policies make the granularity for the permissions flexible and adaptable without touching the applications themselves. The explicit notation of authorization policies and the enforcement independent from the application offer a new extensibility.

1 INTRODUCTION

Due to emerging Service Oriented Architecture (SOA) data and applications formerly hidden are now being exposed as services to the outside world. This raises unique challenges of securing and governing data exchange. Access control is not anymore based on static role-based access control (RBAC) models. Context-information is used to define more precise and fine-grained authorization policies.

As context-aware access control, we understand the integration of context information into the definition and enforcement of authorization policies. It allows the transition to access control adapting to the current situation. Dynamic authorization enforcement makes authorization decisions based upon runtime parameters rather than simply a role assignment. Following the SOA principle services can be composed to new composite services and applications are constructed out of them. The services are forced to provide a flexible and configurable authorization concept that can be changed according to the application needs. Authorization policies defined on application level are broken down to the service level. We propose the separation of business and authorization logic to fulfill the requirements requested by the loosely coupled SOA services.

The granularity used during the access control should not only be defined by the services. It should be more independent and in relation to the using application. We describe a solution that allows the definition of the authorization policy granularity outside the service itself. We show the use of resource hierarchies as part of the authorization policy.

In section 2, we define context-aware authorization policies with and without resource hierarchies formally. Section 3 contains two scenarios where the policies are applied to real business scenarios. In section 4, we show the architecture used to validate our approach. Section 5 shows related approaches and discusses our solution. At the end a further outlook is given.

2 CONCEPT

In this section, we define our terminology and introduce a definition of context-aware authorization policies. Further, we show the use of resource hierarchies to adapt the granularity of the policies.

2.1 Context-aware Authorizations

Access control is the process of granting permissions in accordance with an **authorization policy**. An authorization policy states “who can do what to what”.

Laube A. and Gomez L. (2007).
DYNAMIC CONTEXT-AWARE ACCESS CONTROL - Use of Resource Hierarchies to Define Fine-grained, Adaptable Authorization Policies.
In *Proceedings of the Second International Conference on Security and Cryptography*, pages 386-393
DOI: 10.5220/0002122903860393

The "who" is a subject, the first "what" is an action, and the other "what" is a resource. In a **context-aware authorization policy** the context is taken into account as additional constraint. The statement can be extended as follows: "who can do what to what under which circumstances". The circumstances correspond to the context of the application.

The **subject** S in an authorization policy is the actor that uses an application or system and requests the access to a resource. Examples are users, as abstraction of real persons or technical processes.

A **resource** R is an object exposed by an application or system. Resources can be data, services or system components.

An **action** A is an operation on a resource. Examples are read, modify or delete. The operations are dependent on the resource to be applied to. The finite **set** of available actions on a resource R is defined as:

$$\underline{A}_R = \{A_{Ri}\}$$

Definition 1 *As **context information** C we understand any property related to an entity. An entity might be the subject or any other participant in the system as well as any object in the real world.*

The context information can be the time or location of the entity, but also more complex characteristics like a role of the subject or the health condition of a patient are possible. Complex context information is the result of aggregation, computing or inference based on atomic context information coming directly from physical or logical sensors.
Every context-aware authorization policy has its own formal **context information set** $\underline{C}$. The ordered set contains all context information types that are used during the policy evaluation:

$$\underline{C} = \{C_i\}$$

In the policy enforcement process a **runtime context information set** $\underline{c}$ is used:

$$\underline{c} = \{c_i\}$$

c_i is the current value of context information type C_i.

The formal context information set $\underline{C}$ contains the type of context information that is later used in the context-aware authorization policy.[1] In the following example, the context information set contains the role of the subject, location, time and the current temperature of the location. The runtime context information set $\underline{c}$ contains the real values that are obtained from the subject and its environment and that are finally used in the access control enforcement process.

$$\underline{C} = \{role, location, time, temperature\}$$
$$\underline{c} = \{\text{`}worker\text{`}, \text{`}room10\text{`}, \text{`}10h00\text{`}, \text{`}10C\text{`}\}$$

Definition 2 *The **context constraint** T is the formal expression of the circumstances in which the permission the subject wants to acquire is granted. The context constraint is expressed as logical disjuncted clauses. A **clause** consists of one or more propositions which are logical conjuncted.*

$$Constraint = \bigcup_i Clause_i$$
$$Clause = \bigcap_j Proposition_j$$

*A **proposition** is a boolean-valued function that can be represented in Backus-Naur-Form:*

$Proposition ::= \langle C \rangle \langle OP \rangle \langle VALUE \rangle$ *where:*
$C \in \underline{C}$;
OP *is a operator in the set* $\{>, \geq, <, \leq, \neq, =\}$, *which can be extended to accommodate user-defined operators as well;*
$VALUE$ *is a specific value of* C.

Definition 3 *A **context-aware authorization policy** is defined as a triple $AP = (S, P, T)$ where:*

- S ... *is the subject in this policy.*
- P ... *is the target permission in this policy $\langle A_R, R \rangle$, where A_R is an action of resource R.*
- T ... *is a context constraint for this policy.*

Definition 4 *We define the **authorization request** as a triple $AR = (S, P, \underline{c})$ where:*

- S ... *is a requestor who triggers the authorization decision.*
- P ... *is the permission the requestor wants to acquire, a pair $\langle A_R, R \rangle$.*
- $\underline{c}$ ... *is the runtime context information set.*

The authorization request AR gets the decision "Permit" only if there exists an applicable policy $AP = (S', P', T)$, such that $S = S'$, $P = P'$, and T evaluates to true under $\underline{c}$ (that means, when all C_i in the constraint T are replaced with the runtime values of $\underline{c}$, then the resulting boolean expression is true).
The **enforcement function** on the policy AP is defined as:

$f_{AP}(AR) = f_{AP}(S, P, \underline{c}) = d$ where:
f_{AP} ... enforcement function,
d ... decision, with $d \in \{$ Permit, Deny, Indeterminate, NotApplicable[2]$\}$

[1] Possible types of context-information, categories and classifications are proposed in (Chen and Kotz, 2000; Mikalsen and Kofod-Petersen, 2004).

[2] We use the decision values defined in XACML(OASIS, 2003) as result value set for the enforcement function.

2.2 Context-aware Resource Filters

Definition 5 *A **resource hierarchy** G_R can be described as directed acyclic graph (DAG) over a finite set of nodes $\underline{R}$, built from a resource and all its direct children and descendants at any depth.*

$$G_R = (\underline{R}, \underline{E})$$

$\underline{R}$ ... *Set of nodes in the hierarchy:*
$\underline{R} = \{r_i\}, R \in \underline{R}$
The set of nodes contains the resource itself.
$\underline{E}$ ... *Set of edges (order pairs of nodes):*
$\underline{E} \subset \underline{R} \times \underline{R}, \underline{E} = \{(r_i, r_j)\}$
R ... *the considered resource R is the root node (source) of G_R:* $(R, r_i) \in \underline{E} \wedge (r_j, R) \notin \underline{E}$

The definition of the node set in the hierarchy and the relations between the nodes are highly dependent of the application and the authorization policy. There are examples later in section 3.

With the definition of resource hierarchies we can introduce a new kind of authorizations policies.

Definition 6 *A **context-aware authorization policy with resource filters** is defined as a set of triples:*

$\underline{AP} = \{AP_i\} = \{(S, P_i, T_i)\}$ *where:*
S ... *is the subject in this policy*
P_i ... *is the target permission, which is defined as pair $\langle A_R, r_i \rangle$, where A_R is an action and r_i is a node of the resource hierarchy: $r_i \in \underline{R}$.*
T_i ... *is a context constraint of resource node r_i.*

Despite the extension of the policy to the resource node level, the authorization request stays unchanged (Definition 4). Only the enforcement function is extended to the nodes of the resource hierarchy.

The **authorization request** $AR = (S, P, \underline{c})$ gets the permission to the node r_i only, if there exists an applicable policy $AP_i = (S', P_i', T_i')$, such that $S = S'$, $P = \langle A_R, R \rangle$, $P_i' = \langle A_R, r_i \rangle$, $r_i \in \underline{R}$, and T_i' evaluates to true under $\underline{c}$.

The new **enforcement function** is now defined as:

$$f'_{AP}(AR) = f'_{AP}(S, P, \underline{c}) = \{(r_i, d)\}$$

f'_{AP} ... enforcement function,
r_i ... node i of the resource hierarchy,
d ... decision, with $d \in \{$ Permit, Deny, Indeterminate, NotApplicable $\}$.

The use of a hierarchy defines a partial order over the set of nodes that applies an additional constraint to the authorization decision. It is not possible to grant the permission to a node while the permission to the father node is denied. This constraint can be described formally as follows:

$(r_i, Permit) \in f'_{AP}(AR)$ while
T_i' evaluates to true under $\underline{c} \wedge$
$(r_i = R$ (r_i is the root node) $\vee$
$(r_i \neq R \wedge (r_j, r_i) \in \underline{E} \wedge (r_j, Permit) \in f'_{AP}(AR)))$

After the formal definition we apply this theory in the following to two application scenarios.

3 SCENARIOS

3.1 Beyond Rbac

Role-based access control (RBAC) (Sandhu et al., 1996) associates permissions with roles to simplify the permission management. Users are members of a role. The role brings together a set of users on one side and a set of permissions on the other side.

We want to show that our approach goes beyond the widespread RBAC model. We define a role as context information related to the subject as proposed in (den Bergh and Coninx, 2005). That implies that the role is part of the runtime context information set.

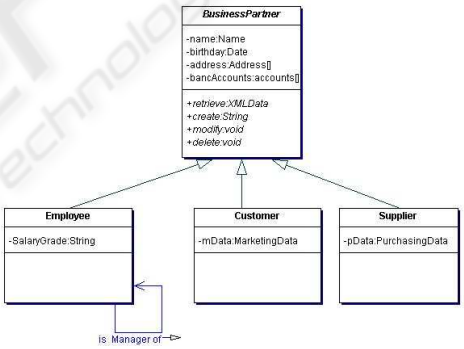

Figure 1: BusinessPartner object hierarchy.

As an example, we imagine a business application that exposes a service to access an object *BusinessPartner*. This object maintains master data of natural persons and companies like customers or suppliers.

Several concrete classes are derived from the abstract class *BusinessPartner*, see Fig. 1. Each derivation adds specific attributes that are only relevant for this type of object. For example, an employee has a salary grade, a private bank account, a private and a business address. A customer has marking data, a shipping address and an invoice address.

The use of the object *BusinessPartner* as interface for the application allows the identical manipulation of the different classes in the hierarchy. The following authorization rules shall be applied to the *retrieve* method of the *Employee* object:

1. *Only a human resource accountant can access all data of an employee.*
2. *An employee has access to the business address of all employees and to his own private address and bank account. He has no access to his salary.*
3. *A manager has access to the data of his/her employees, except for the sensible personal data, like birthday, private address and bank account.*

To formalize the authorization policy for the HR accountant (Rule 1) no resource hierarchy is needed:

$$AP_{Accountant} = (S, \langle read, employee \rangle, (role = \text{`}accountant\text{`}))$$

For the next rules, the definition of a resource hierarchy for the employee object is necessary. The design of a resource hierarchy requires knowledge of the application and understanding of the requirements of the authorization rules. For convenience, the resource hierarchy is graphically presented in Fig. 2. The node set $\underline{R}_{employee}$ is defined by:

$$\underline{R}_{employee} = \{e.employee, e.bus\_address, e.personal\_data, e.salary, e.general, e.privat\_address, e.birthday, e.name, e.privat\_bank, e.manager \dots\}$$

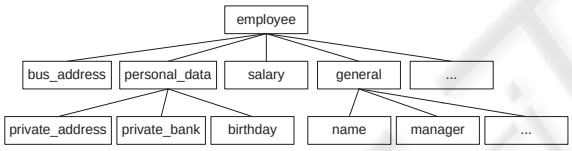

Figure 2: Employee's resource hierarchy.

The authorization rule for an employee (Rule 2) is now a set of authorization policies on the employee's resource hierarchy:

$$\underline{AP}_{Employee} = \{AP_{E1}, AP_{E2}, AP_{E3}, AP_{E4}, AP_{E5}, AP_{E6}, AP_{E7}, AP_{E8}\}$$

$$AP_{E1} = (S, \langle read, e.employee \rangle, (role = \text{`}employee\text{`}))$$
$$AP_{E2} = (S, \langle read, e.bus\_address \rangle, true)$$
$$AP_{E3} = (S, \langle read, e.personal\_data \rangle, (id(e) = id(S)))$$
$$AP_{E4} = (S, \langle read, e.private\_address \rangle, true)$$
$$AP_{E5} = (S, \langle read, e.private\_bank \rangle, true)$$
$$AP_{E6} = (S, \langle read, e.general \rangle, true)$$
$$AP_{E7} = (S, \langle read, e.manager \rangle, true)$$
$$AP_{E8} = (S, \langle read, e.name \rangle, true)$$

Because of the partial order in the hierarchy it is sufficient to apply context constraints only to the highest level in the hierarchy. The role constraint is applied to the root node (e.employee) in the hierarchy. The access to the subnodes like e.bus_address and e.general and their subnodes is only granted when the access to the root node was permitted.

We assume that a manager has two roles: the manager role and the employee role. Only one additional authorization has to be defined for the manager's privilege (Rule 3):

$$AP_{Manager} = (S, \langle read, e.salary \rangle, (role = \text{`}manager\text{`} \wedge manager(e) = S))$$

The authorization rules stated above contain also one implicit obligation. An accountant in his role as employee should not be able to access his own salary grade. Therefore the authorization policy should be enhanced as follows:

$$AP_{Accountant2} = (S, \langle read, employee \rangle, (role = \text{`}accountant\text{`} \wedge e.id \neq S.id))$$

RBAC roles would be powerful enough to differentiate between a HR accountant and a normal employee. However, RBAC gives either permission to a resource or denies it completely (all or nothing paradigm). In order to implement the authorization rules described above, it would be necessary to implement a much more detailed and fine-grained service interface. In this case, there are two possibilities: The first is to implement an interface specialized for each role, which creates a very high dependency to the authorization policy. The second option is to have many smaller services that allow to retrieve parts (sub resources) of the *BusinessPartner*, like the name or bank account. This would create a big impact on performance. Instead of having only one service call to get all data, a couple of calls are necessary.

The requirement, that a manager can access only the data of his/her employees, cannot be enforced with a simple role-based or group-based concept. At least RBAC extensions, based on organizational structure, like OR-BAC(Kalam et al., 2003), are necessary. Our approach considers information about the organization in a company as context-information related to the subject. In addition, it is possible to enforce restrictions related to the separation-of-duty principle, like described in the policy $AP_{Accountant2}$.

3.2 e-Health

The second scenario addresses mobile health care applications. It demonstrates the use of complex context information during the access control to the medical data in emergency situations.

A patient is monitored by several health sensors. The physician needs access to the medical data of the patient. The medical record consists of personal data (name, birthday, address, ...), insurance data, medication history, treatments, sensor data and more.

Contextual Information				Authorization Decision			
Family doctor	Emergency	House call	Proximity	Name, Birthd.	Medication	Treatments	Sensor data
yes	-	-	-	Permit	Permit	Permit	Permit
no	yes	-	far	Permit	Deny	Deny	Deny
no	yes	-	near	Permit	Permit	Deny	Permit
no	no	yes	far	Permit	Deny	Deny	Deny
no	no	yes	near	Permit	Permit	Permit	Permit

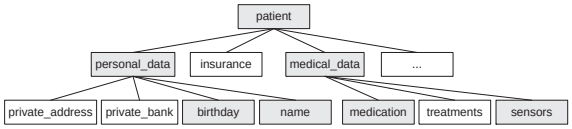

Figure 3: Patient's resource hierarchy.

The object schema in Fig. 1 can easily be extended to cover the requirements of patients and medical data. The resource hierarchy for the patient can be defined similar to the hierarchy for the employee (Fig. 2). The graph is shown in Fig. 3. The node set is described as:

$$\underline{R}_{patient} = \{p.patient, p.personal\_data, p.name, p.private\_address, p.private\_bank, p.birthday, p.insurance, p.medical\_data, p.medication, p.treatments, p.sensors, \dots\}$$

Except the family doctors, a physician gets access only when several conditions apply, see Table 3.1. The resulting authorization policy that satisfies these constraints can be defined as follows:

$$\underline{AP}_{Physician} = \{AP_{P1}, AP_{P2}, AP_{P3}, AP_{P4}, AP_{P5}, AP_{P6}, AP_{P7}, AP_{P8}\}$$

$$AP_{P1} = (S, \langle read, p.patient\rangle, (role = \text{`physician`}))$$
$$AP_{P2} = (S, \langle read, p.personal\_data\rangle, (familyDoctor(p,S) = true \vee emergency(p) = true \vee houseCall(p,S) = true))$$
$$AP_{P3} = (S, \langle read, p.name\rangle, true)$$
$$AP_{P4} = (S, \langle read, p.birthday\rangle, true)$$
$$AP_{P5} = (S, \langle read, p.medical\_data\rangle, (familyDoctor(p,S) \vee ((emergency(p) = true \vee houseCall(p,S) = true) \wedge proximity(p,S) = \text{`near`})))$$
$$AP_{P6} = (S, \langle read, p.medication\rangle, true)$$
$$AP_{P7} = (S, \langle read, p.treatments\rangle, (houseCall(p,S) = true))$$
$$AP_{P8} = (S, \langle read, p.sensors\rangle, true)$$

The used context information set $\underline{C}$ can be noted as:

$$\underline{C} = \{role, familyDoctor(p,S), emergency(p), housecall(p,S), proximity(p,S)\}$$

The context information is often noted as functional expression, which is evaluated during runtime. If a property is stated without reference, it belongs automatically to the subject, for example 'role'.

Let us make an example to demonstrate the evaluation of context constraints during runtime. The physician Dr. Wells wants to read the medical record of patient Bob. The physician role is assigned to the doctor. He is a not member of the group of family doctors assigned to Bob. Bob's health status is critical. It is an emergency situation. Dr. Wells got no house call assignment for Bob, but he is close to Bob.

The authorization request can then be stated as:

$$AR_1 = (\text{`Dr.Wells`}, \langle \text{`read`}, \text{`PatientBob`}\rangle, \{\text{`physician`}, false, true, false, \text{`near`}\})$$

The enforcement function based on the authorization policy $\underline{AP}_{Physician}$ will return the following result:

$$f'_{APPhysician}(AR_1) = \{(p.patient, Permit), (p.personal\_data, Permit), (p.private\_address, Deny), (p.private\_bank, Deny), (p.name, Permit), (p.birthday, Permit), (p.insurance, Deny), (p.medical\_data, Permit), (p.medication, Permit), (p.treatments, Deny), (p.sensors, Permit), \dots\}$$

The result is graphically shown in Fig. 3: All gray shaded nodes of the hierarchy are permitted.

This scenario shows the use of context information together with the use of resource hierarchies. The returned medical information is always adapted to the need of the situation. The health application is completely independent of the access control and focuses on the functional aspect of retrieving the patients' medical record. The definition of authorization policies and the definition of resource hierarchies are technically separated from the health application, but still logically dependent on it.

This separation eases the adding of more players in the health care chain, like specialists, pharmacists or health insurance companies. The medical application can be used to retrieve the relevant data for each player dependent on the situation they are involved in.

4 ARCHITECTURE

Our approach has been validated in a prototype based on the context-aware security framework (see Fig. 4) developed in the MOSQUITO project (MOSQUITO,

2006). The use of resource filters in the authorization policies and during policy enforcement is an extension.

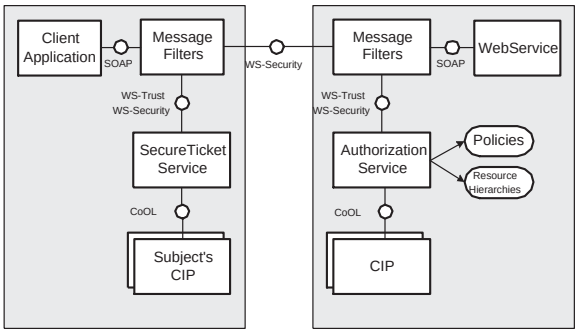

Figure 4: General architecture.

All SOAP messages between the application (web service client) and the web service itself have to pass intermediaries, to enforce the configured security policies on message (SOAP) level (see Fig. 5). Intermediaries are a pipeline of message filters (proxy) which support WS-Security(OASIS, 2006).

The client side intermediary adds encryption, integrity checks and credentials. The server-side intermediary decrypts, verifies the integrity and verifies the authorization in credentials. It offers the same interface as the web service plugged behind it and is therefore invisible from outside. The Security Token Service (STS) (OASIS, 2005) on the client device generates signed context information retrieved from the client's Context Information Providers (CIP) and adds as well the role information for the subject.

The retrieved context information is converted into a XACML request via XSLT (W3C, 1999) transformation, which is sent to a Policy Decision Point (PDP) that enforces the access control policy. If the XACML response is "Permit" the original request from the web service client is passed to the web service and processed there. Otherwise, an exception is sent to the client application.

The response of the web service is as well passed through the security proxy with its message pipeline. First, the response filter is applied to make the web service response compliant to the access control policy, using the XACML response from the Contextual Authorization Service. Afterwards additional filters take care of encryption/decryption and integrity.

The standard for access control policies is XACML(OASIS, 2003). XACML supports RBAC policies as well as context-aware access control. The policy enforcement relies on verifying attribute values distributed in four categories, related to the subject, the resource, the action and the environment. To

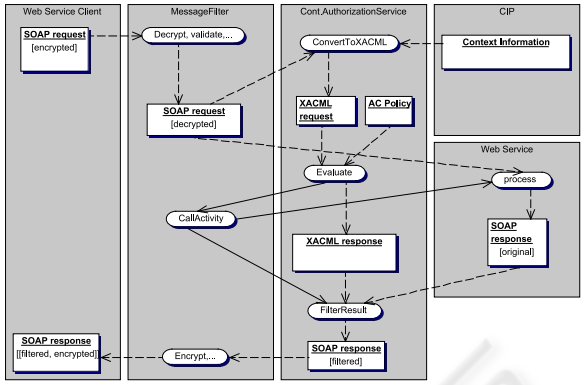

Figure 5: Activity diagram of web service call processing.

support evaluation of context information, the existing implementation can be extended by defining new primitive attributes types that offer a higher level of abstraction for data representation. As an example the authorization policy for emergency is shown in Fig. 6.

The goal of the authorization service is to enforce context-aware access control to web services(Lachmund et al., 2006).

The server-side message filter calls the method *authorizationRequest* of the Contextual Authorization Service providing the following three parameters: **ActionTo** contains in the action to be performed and the resource description. Parameter **Message** contains the SOAP message request. **Credentials** contains all credentials attached to the requestor.

The *ActionTo* parameter is used to retrieve the associated *XACMLRequestGenerator* and the security policy to be applied. The context information needed to enforce the policy is evaluated out of the security policy and requested from the available CIPs.

The *XACMLRequestGenerator* converts the incoming message and the credentials into a XACML request using XSLT transformations. It adds the data retrieved from the CIPs. The generated request is evaluated by the PDP. Before the evaluation starts the set of nodes of the requested resources is determined. The evaluation can be based on the resource as well as on context information or the message content. The resulting response contains a separate result for each node in the hierarchy (see Fig. 7).

When the access to a resource is permitted the original web service request is sent to the web service and processed there. The web service response has again to pass the message pipeline. It evaluates all not permitted subresources and applies a XML transformation to modify the SOAP message accordingly.

The result of the enforcement function $f'_{AP}(AR)$ is used to make the data returned by the web service

```
<Policy PolicyId="EmergencyPolicy">
 <Target>...</Target>
 <Rule RuleId="MixedLocalisationRule"
   Effect="Permit">
  <Target>
   <Resources>...</Resources>
   <Actions>...</Actions> </Target>
  <Condition FunctionId="function:string-equal">
   <Apply FunctionId="function:string-one-and-only">
    <SubjectAttributeDesignator DataType=string
      AttributeId="role"/></Apply>
   <AttributeValue DataType="string">physician
   </AttributeValue>
  </Condition>
  <Condition FunctionId="function:and">
   <Apply FunctionId="coolFunction#CloseTo">
    <Apply FunctionId="coolFunction#findLocation">
     <SubjectAttributeDesignator
       DataType=cool#GPSLocation
       AttributeId="SubjectLocation"/>
    </Apply>
    <Apply FunctionId="coolFunction#findLocation">
     <SubjectAttributeDesignator
       DataType="cool#GPSLocation"
       AttributeId="ObjectLocation"/>
    </Apply>
    <AttributeValue DataType="integer">50
    </AttributeValue>
   </Apply>
   <Apply FunctionId="coolFunction#IsEmergency">
    <Apply FunctionId="coolFunction#findEmergency">
     <SubjectAttributeDesignator
       DataType="cool#Emergency"
       AttributeId="ObjectEmergency"/></Apply>
   </Apply>
  </Condition></Rule>
</Policy>
```

Figure 6: Authorization Policy example.

call compliant to the policy. The message response is passed through a filter, that either removes the data, which is not in the set of permitted nodes.

$$f_{Filter}(R) = \{r_i\} \text{ with } (r_i, Permit) \in f'_{AP}(AR)$$

Two possible modifications can be applied: If the access to an optional node of the response is denied the part can be removed from the structure. In the case of a mandatory element the information must be blanked out (e.g. replaced with "xxx…" for strings). The result of the filtering is a modified response which is compliant to the access control policy.

5 RELATED WORK

RBAC based models are widespread. Many extensions like GRBAC, OR-BAC and DRBAC were introduced to make the original approach more flexible and to overcome static permissions. GRBAC(Moyer et al., 2000) is an extension that uniformly applies the concept of roles not only to subjects, but also to objects and system states. OR-BAC(Kalam et al., 2003) offers the use of contextual rules related to permissions, prohibitions, obligations and recommendations often valid in specific organizational structures. The DRBAC model proposed in (Zhang and Parashar, 2004) uses context information during the enforcement process. A change in the user's context implies a change of his access privileges.

```
<Response>
 <Result ResourceID=
   "http://localhost/MedicalData/personal">
   <Decision>Permit</Decision>
   <Status><StatusCode Value="ok"/></Status>
 </Result>
 <Result ResourceID=
  "http://localhost/MedicalData">
  <Decision>Permit</Decision>
  <Status><StatusCode Value="ok"/></Status>
 </Result>
 <Result ResourceID=
  "http://localhost/MedicalData/insurance">
  <Decision>Deny</Decision>
   <Status><StatusCode Value="ok"/></Status>
 </Result> ...
</Response>
```

Figure 7: XACML response for a resource hierarchy.

A combination of these models would be sufficient to meet all requirements of our 2 scenarios. However, none of the approaches is able to deal with the special requirements of the SOA architecture without influencing the development paradigm and performance. Like already concluded in section 3.1 the use of these approaches is not possible in a system built out of loosely coupled application services with very general interfaces without contradicting the principles behind.

Our approach is based on a strict separation of application and security that is imposed by our architecture. There are other frameworks like the Open Group's Authorization API, the Policy Machine from NIST (Galiasso et al., 2000) and the Object Security Attributes middleware (Beznosov, 2002), which allow to externalize the authorization policies and to enforce them in the framework or middleware. Not all of them allow the use of dynamic context-information during the policy enforcement process. Our approach goes further: It allows the separation of security-related application knowledge in a way that the application focuses only on functional aspects. Authorization policies can be designed and modified independly from the application and can include all context-information relevant and available to the system.

In addition, the use of resource filters allows a

modification of the service response and request in order to modify it according the authorization policy. The service itself can be designed and implemented in a very generic way.

The use of resource hierarchies is quite common. Normally similar single resources are grouped, like printers (Ilechko and Kagan, 2006). The hierarchies are used to assign identical authorization policies, often ACLs, to all members. Our approach is the first using the resource hierarchy in the opposite way and putting single resources in a hierarchy that can be addressed explicitly in the authorization policy.

6 CONCLUSION

We present a dynamic context-aware access control that can be used in a SOA based architecture. We propose a separation of business and security logic on the service level that allows the definition of adaptable and easily extensible authorization policies outside the service. The complexity when mixing access control into the business logic can be avoided. The externalized authorization policies are more explicit.

The authorization policies are dynamically enforced using runtime context information. Resource hierarchies make it possible to define the authorization policy granularity outside the service. The definition of the resource hierarchy itself is based on service, on the needs of the client application and, if needed, on additional context information. The hierarchy helps to keep the relationships between the sub resources, which would not be the case when the service itself would split its functionality to a similar level.

The use of context information questions the way in which the information is acquired. Context-aware services and applications have to deal with issues of trust and dependability. In this paper, we did not address the related problems but we will focus them in the future.

Performance in SOA based application is an issue investigated by industry and the research community. The proposed architecture consisting of messages filters and the Contextual Authorization Services adds an additional bottleneck. Retrieving and processing of context information is also time consuming. Due to the application of the resource filters after the business logic it is also possible that the SOA service processes and retrieves more data than later passed to the client applications. There is a need for adaptation of our architecture in the area of high performance application.

REFERENCES

Beznosov, K. (2002). Object security attributes: Enabling application-specific access control in middleware. In *4th International Symposium on Distributed Objects and Applications (DOA)*, pages 693–710.

Chen, G. and Kotz, D. (2000). A Survey of Context-Aware Mobile Computing Research. Technical Report TR2000-381, Dartmouth College, Computer Science, Hanover, NH.

den Bergh, J. V. and Coninx, K. (2005). Towards integrated design of context-sensitive interactive systems. In *PERCOMW '05: Proceedings of the Third IEEE International Conference on Pervasive Computing and Communications Workshops*.

Galiasso, P., Bremer, O., Hale, J., Shenoi, S., and al. (2000). Policy mediation for multi-enterprise environments. In *ACSAC '00: Proceedings of the 16th Annual Computer Security Applications Conference*.

Ilechko, P. and Kagan, M. (2006). Authorization concepts and solutions for j2ee applications.

Kalam, A. A. E., Benferhat, S., Miège, A., Baida, R. E., Cuppens, F., Saurel, C., Balbiani, P., Deswarte, Y., and Trouessin, G. (2003). Organization based access control. In *POLICY '03: Proceedings of the 4th IEEE International Workshop on Policies for Distributed Systems and Networks*, page 120, Washington, DC, USA.

Lachmund, S., Walter, T., Bussard, L., Gomez, L., and Olk, E. (2006). Context-aware access control. In *IWUAC'06: Proceedings of the third Annual International Conference on Mobile and Ubiquitous Systems*.

Mikalsen, M. and Kofod-Petersen, A. (2004). Representing and Reasoning about Context in a Mobile Environment. In Schulz, S. and Roth-Berghofer, T., editors, *Modeling and Retrieval of Context 2004 (MRC)*, volume 114, pages 25–35.

MOSQUITO (2006). IST 004636 MOSQUITO Project.

Moyer, M., Covington, M., and Ahamad, M. (2000). Generalized role-based access control for securing future applications. In *23rd National Infromation Systems Security Conference (NISSC 2000)*.

OASIS (2003). eXtensible Access Control Markup Language (XACML) 1.1.

OASIS (2005). Web Service Trust Language (WS-Trust) 1.3.

OASIS (2006). Web Service Security: SOAP Message Security 1.1 (WS-Security 2004).

Sandhu, R. S., Coyne, E. J., Feinstein, H. L., and Youman, C. E. (1996). Role-based access control models. *IEEE Computer*, 29(2):38–47.

W3C (1999). W3C XSL transformations (XSLT) 1.0.

Zhang, G. and Parashar, M. (2004). Context-aware dynamic access control for pervasive computing.