Reliable and Adaptable Security Engineering for Database-Web Services

Martin Wimmer*, Daniela Eberhardt‡, Pia Ehrnlechner‡, and Alfons Kemper*

*Technische Universität München, 85748 Garching b. München, Germany
‡Universität Passau, 94032 Passau, Germany
{wimmerma,kemper}@in.tum.de, {eberhard,ehrnlech}@db.fmi.uni-passau.de

Abstract. The situation in engineering security for Web services that access databases is as follows: On the one hand, specifications like WS-Security are concerned with the security management for Web services, while on the other hand there exist well established mechanisms for access control in the area of commercial database systems. In handling security for services that rely on database systems, two extreme approaches can currently be observed: The more database-centric one, where the access control decisions are left to the DBMS, and the service-centric authorization approach. The service-centric approach requires a Web service to run under control of the database system provider as operations like queries and updates have to be executed with comprehensive privileges. Authorization has to be enforced by the service itself. In case access control policies of a service are defined independently with regard to the database policies, authorization mismatches are likely to be induced.
In our new approach we bridge this gap between DBMS authorization and access control of Web services by supporting reliable and adaptable access control engineering. The policies of the DBMS constitute the basis for the authorization of Web services. These are therefore automatically extracted before they are refined by additional conditions. As a final step, it must be verified that service policies do not grant more permissions than database policies do, thus ensuring reliable service execution.

1 Introduction

The concept of Web services has attracted the interest of the research as well as the commercial sector. Web services (also called *services* or *e-services*) are autonomous software components that are uniquely identified by a URI and can be invoked by use of standard Internet protocols. Security and privacy issues are always in the center of concern of any distributed system. This is even more intensified when service execution and aggregation are done dynamically. Secure message exchange at the transport layer can be assured by the use of protocols like SSL and TSL, while message confidentiality and integrity are addressed by SAML [1] and WS-Security [2]. These specifications focus on the exchange of security credentials for authentication and authorization in the area of e-services. Most Web service development tools like .Net or JWSDP[1] support substantial

[1] Java Web Services Developer Pack

security functionality and allow the realization of sophisticated access control concepts for e-services.

When engineering Web services that depend on databases, which applies to the majority of business applications, the access control mechanisms of the underlying DBMS have to be taken into consideration, too. In this area, common authorization standards are discretionary access control (DAC), mandatory access control (MAC) [3] and role based access control (RBAC) [4, 5]. Two extreme approaches regarding the access control of Web services that interface with databases can currently be observed. On the one hand, there exists a database-centric approach where the complete access monitoring is left to the database system. On the other hand, authorization can be enforced on the side of the service. This service-centric approach requires a service to run under control of the database system provider as operations like queries and updates have to be executed with comprehensive privileges. The common practice lies in-between these two extreme approaches. Consequently, two access control systems have to be taken into consideration, one for the service and one for the database. In case the two are administered independently, severe authorization mismatches are likely to occur.

Our approach addresses this topic and provides solutions for the reliable definition of service policies. Obviously the service policy depends on the permissions granted to the DBMS account that is used by the Web service to establish connections to the database. As depicted in Fig. 1, the authorization settings that are applicable in the database context are extracted automatically. The preprocessed policy is subsequently refined by the service developer, insofar, as further constraints are added. Apart from the modification of database policies, service policies can also be generated semi-automatically based on service specifications. In both cases, in order to guarantee reliable service execution, it must be ensured that the final Web service policy is a valid refinement of the database policy, in terms of being less permissive.

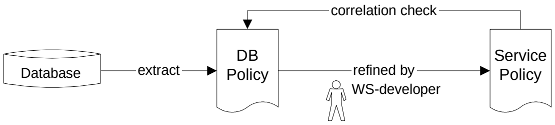


Fig. 1. Workflow of service policy definition.

Today it is common practice that database dependent Web applications have over-privileged or even total access to the DBMS. In case of attackers gaining control over the Web service, e.g., by infiltrating the computer on which the service is running, privacy and security threats arise. Concerning this, we present practical solutions for reducing the security risks for databases.

The remainder of this paper is structured as follows: In Sect. 2 we discuss security requirements of Web services that access databases. The basic techniques of our approach, which we integrated in our prototype Web service platform ServiceGlobe, are presented in Sect. 3. In Sect. 4 we describe how to design access control for Database-Web services and additionally elaborate on the dynamic exchange of policies. Finally, Sect. 5 presents some related work, prior to a conclusion and a brief discussion of future work in Sect. 6.

2 Considering Access Control of Database-Web Services

In this section we describe the shortcomings that arise when e-service policies and database policies are defined in an uncorrelated manner. We illustrate these regarding an example of a hospital administration scenario. As depicted in Fig. 2 we use a simplified model of a hospital data set, assuming that patients and physicians are uniquely identified by their names.

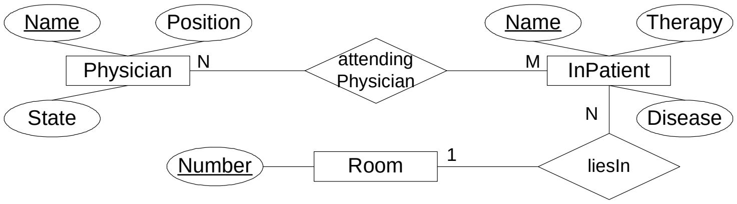
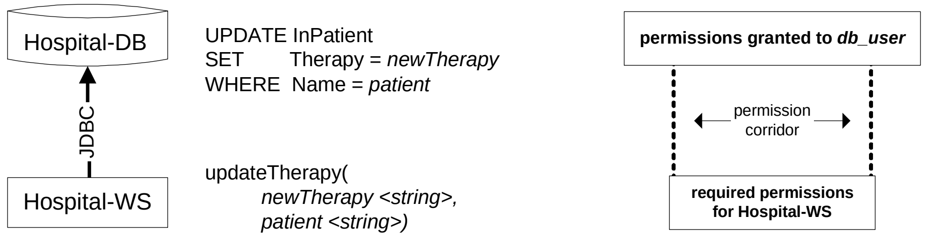


Fig. 2. Entity-Relationship diagram of a simplified hospital application.

As use case we consider a portal for physicians that allows them to modify the therapy of patients. The implementation might be realized as illustrated in Fig. 3. A quite elementary e-service named *Hospital-WS* is created, which, behind the scenes, establishes a JDBC-connection to the database *Hospital-DB*. By invoking *updateTherapy()* the denoted update statement is executed.

The service interfaces with the database under the account of *db_user*. Authorization takes place in two phases: First, the requestor has to be authorized to invoke *updateTherapy()* depending on the service policy. Second, the execution of the aforementioned query under the account of *db_user* has to be allowed. Several shortcomings can be observed if the two authorization policies deviate:

- Obviously, the access control of the service depends on the authorization settings of the DBMS. The service can at most provide functionality that is also supported by the database. If this condition is not satisfied, a successful authorization on part of the service is no guarantee for effective service execution as the DBMS can still deny access on certain resources. Ensuring this relationship thus states a precondition for reliable service execution.


Hospital-DB
JDBC
Hospital-WS
permissions granted to *db_user*
permission corridor
required permissions for Hospital-WS

```
UPDATE InPatient
SET     Therapy = newTherapy
WHERE  Name = patient
```

```
updateTherapy(
        newTherapy <string>,
        patient <string>)
```

Fig. 3. Illustration of the portal realization and the associated access control.

- In order to avoid administration overhead, connections to databases are oftentimes established under over-privileged accounts. As illustrated in Fig. 3 *db_user* is granted more privileges than needed to perform *updateTherapy()* of the *Hospital-WS* service. This constitutes an avoidable safety risk.
  In the case of attackers gaining control over the service by exploiting flaws of the system, they – in the worst case – will also be able to execute any command *db_user* is privileged to execute. As a consequence severe inconsistencies and privacy violations may arise. Our intention is to reduce these risks by providing access to the resource only through a tailored permission corridor as illustrated in the right part of Fig. 3.

Considering the service method *updateTherapy()*, a plain verification whether access to the database will be allowed is not sufficient. Physicians shall only be granted access to the information concerning patients of whom they are the attending physicians. Access to any other medical record has to be denied. This check can be performed in one of the following ways.

User related views: Security on the level of data sets or single columns can be attained by views that are related to database users. Assuming that the column *physicianName* of *attendingPhysician* contains user identifiers of the database system the following view represents a restricted copy of *InPatient*.

```
CREATE VIEW InPatientView AS
SELECT * FROM InPatient p
WHERE EXISTS (SELECT * FROM attendingPhysician
  WHERE  patientName = p.Name AND physicianName = USER)
```

Authorization by query: Obviously, the authorization check can also be included in the original query by joining *InPatient* and *attendingPhysician*.

External policy: New policy languages like XACML allow the expression of fine-grained conditions. So, another solution would be to formulate the constraint as part of a self-contained Web service policy.

User related views constitute a well suited authorization mechanism. One disadvantage is that clients require database accounts and have to supply their login information for the database when calling the Web service. Thus it is no useful concept for hiding the DBMS to service users. Including the authorization into the query statement will counteract the software design principle of separating security and business logic. So, the third approach of refining the access control on part of the Web service by the use of separate policies is the most practical and moreover flexible design guideline.

Apart from this quite simple example, more significant conditions can be expressed. Regarding our scenario it might be necessary to demand that first-year residents should not be allowed to change the therapy of patients, though they might be the corresponding attending physicians. Changing the therapy requires a chief physician to agree to the resident's decisions. Such constraints can not be expressed with common access control techniques of databases, because authorization in this case depends on complex context information and security credentials of several subjects have to be evaluated.

3 Policy Management

We integrated the required authorization functionality in our Web service platform ServiceGlobe. Before we present details of how policies are generated and compared, we give a brief introduction to the eXtensible Access Control Markup Language (XACML) which we use as policy language.

3.1 Short Introduction to XACML

XACML [6] is based on XML and is consequently well applicable as policy language in Web service environments: Existing parser technologies can be utilized and basic language constructs are well known to the community thus increasing acceptability.

XACML provides a request/response mechanism to support communications. A typical scenario for employing XACML is depicted in Fig. 4: A distinguished Web service operates as Policy Enforcement Point, PEP. A PEP has to ensure that requestors are authorized to execute the demanded operation and to access the requested resources. Therefore the incoming SOAP-request has to be transformed into an appropriate XACML-request. This is done by the Context Handler. The XACML-request document is passed to the Policy Decision Point, PDP. The PDP evaluates the request against applicable policies that are supplied by a Policy Administration Point (PAP). The result is sent back to the Web service that either aborts the execution and returns an error message or executes as demanded.

A `Policy` consists of a `Target`, a set of `Rules` and optional `Obligations` that specify the access conditions. `Rules` contain most of the logic of a `Policy`. A single `Rule` is set up by a `Target`, an optional `Condition` and an `Effect` that will be either `Permit` or `Deny`. The way several rules are combined in a `Policy` is specified by its rule combining algorithm that, among others, can be `deny-overrides` or `permit-overrides`. While the first one states that `Deny` is

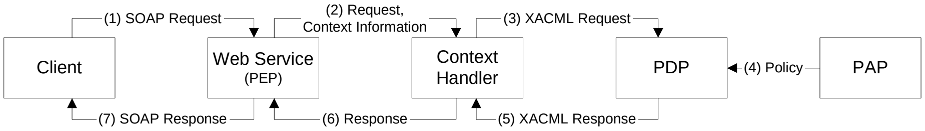


Fig. 4. Control flow of the authorization process

returned in case any rule evaluates to `Deny`, the second will return `Permit` in the analogous situation[2]. In case a processing error occurs `Indeterminate` is returned. A `Policy` evaluates to `NotApplicable` if none of its `Rule`s applies. XACML defines further combining algorithms and allows the definition of new ones.

If the authorization decision depends on further constraints these are coded by the optional `Condition` of a `Rule` that either evaluates to `True`, `False` or `Indeterminate`. `Condition`s can be quite complex, i.e., any arbitrary Boolean function can be expressed. The refinement of a `Rule` takes place by imposing further constraints on its `Target`, thus restricting the `Rule`'s applicability.

The `Target` of a `Policy` or `Rule` is composed of a set of `Subject`s, a set of `Resource`s and a set of `Action` elements. The Target thus constitutes the actor, the module to be accessed and the operation to be performed. In case any of the mentioned elements are not explicitly determined, `AnySubject`, `AnyResource` and `AnyAction` are used.

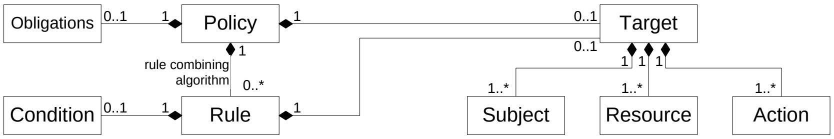


Fig. 5. XACML components (adapted from [6])

In order to use XACML for policy enforcement the SOAP request must be transformed into an XACML `Request`. A `Request` is composed of one or many `Subject` elements that represent the initiator of a `Request`, e.g., identities of humans or other e-services. Required elements of a `Request` are its `Resource` and `Action` that specify the resource for which access is requested and the way access shall be performed. A `Request` is evaluated against applicable `Policy` definitions and `Rule`s, whose `Target` matches with the `Request`. The set of `Subject`s of a `Target` is evaluated disjunctively. In contrast, each `Subject` element contains a

[2] In the following we use `permit-overrides` due to the assembly of database policies that consist of privileges. Consequently, an evaluation to `NotApplicable` states that access is denied.

conjunctive sequence of SubjectMatch elements that define the attributes of the Subject. So, for the match of the subject of a Target with the one of a Request, it is necessary that at least one of the Subject elements applies, while for the match of one Subject element it is necessary that all associated attributes match with the Request. Analogous considerations apply to Resources and Actions. The way Targets are evaluated are significant for Def. 1 in Sect. 3.3.

In the following we equate terms like policy or subject with the corresponding XACML terms, i.e., Policy or Subject.

3.2 Web Service Platform ServiceGlobe

We integrated the functionality for access control management in our prototype ServiceGlobe. ServiceGlobe [7] is a lightweight, distributed and extensible e-service platform. It is completely implemented in Java and based on standards like XML, SOAP, WSDL, and UDDI. ServiceGlobe specific services are mobile code. They can be loaded from code repositories and executed on so-called *service hosts*. These are standard Internet servers additionally running a ServiceGlobe runtime engine.

Two security issues arise in the given situation. On the one hand, security for service hosts has to be ensured: Because possibly unknown Web services are loaded, the execution of erroneous or even malicious code must be monitored. Security for service hosts is based on the Java security architecture, i.e., the Java sand-box mechanism, as described in [8].

On the other hand, communication between services and clients must be secure which includes confidential message exchange as well as authentication and authorization of requestors. The exchange of security credentials is based on WS-Security that relies on XML-Encryption and XML-Signature and specifies how security information is embedded in SOAP-documents. Authentication of service requestors constitutes the pre-requisite for the subsequent access control.

3.3 Authorization Functionality

The access control functionality described in the following paragraphs has been implemented in Java and was integrated into the ServiceGlobe platform. An Oracle 9i installation was used as relational DBMS for examining the correlation of database and e-service authorization concepts.

Generation of policies, requests and user profiles. As we focus on Web services that interact with databases, service operations are related to SQL-commands. Instead of writing policy and request documents from scratch, a preprocessing of SQL statements allows the preparation of these XACML constructs. In particular, the referenced tables or columns form resources while operations like update or select constitute actions. Thus targets of policies and rules as well as requests can be formulated. In general, the generated documents need to be refined by the service developer. This means that the access rules are redesigned more restrictively by additional conditions. For example, the subject identity that specifies to whom the policies apply cannot be automatically extracted from SQL-commands and has to be added by the service developer.

As motivated in Sect. 2, accessing the database via the account of a highly privileged user shall be avoided due to security considerations. Again, the required privileges are extracted from the SQL-statement and scripts for database user profiles are generated. These profiles are designed to establish exactly the authorization corridor needed for the particular service.

Extraction of database policies. The extraction of the access rules of the underlying DBMS depends on the specific database system. Taking an Oracle DBMS as an example, privileges are stored in data dictionary views like *user_tab_privs*. Policies of the following three categories are generated:

role assignment policies: Commonly, databases support role based access control. This type of policy is used to assign roles to users or, in order to support hierarchical RBAC, to senior roles. This refers to [5] as senior roles acquire the permissions of their junior roles. Following the recommendation of [9], policies are generated consisting of rules, whose target is composed as follows: The actual role constitutes the resource, while the subject is either a user or a senior role. The target's action is of an *enable*-state. See [5, 9] for further issues on separation of duty that is beyond the scope of this paper.

permission policies: A *permission policy* defines a privilege that can be granted to users and/or roles. The target of a *permission policy* is not specified in order to enable its assignment to arbitrary roles and users. That means that the elements of the target are `AnySubject`, `AnyResource` and `AnyAction`. Each policy contains one rule that represents the privilege to execute a certain operation on a resource, e.g., update on a table.

base policies: A policy of this category assigns permissions to users or roles. This is done by defining the policy target with the respective user or role identity as subject. The granted privileges are expressed by referencing the appropriate *permission policies*. These policies are included by the use of `PolicyIdReference` elements of the XACML specification.

3.4 Partial order for policies.

The refinement of access rules demands a way to compare policies in order to determine whether one policy is more restrictive than another one. An informal definition of *more restrictive* is that fewer permissions are granted to fewer subjects under harder constraints. Before we define a partial order for policies, the following preceding considerations regarding the comparisons of targets and rules have to be made.

Definition 1. *Partial order for targets*
Let T_1 and T_2 be targets, with $T_1.sub$, $T_2.sub$ denoting the associated sets of

subjects, $T_1.res$, $T_2.res$ the sets of resources and $T_1.act$, $T_2.act$ the sets of actions. A target T_1 is subordinate or equal to a target T_2, $T_1 \sqsubseteq T_2$, if and only if

$$(\forall s_1 \in T_1.sub : \exists s_2 \in T_2.sub : s_1 \sqsupseteq s_2) \wedge \tag{1.1}$$

$$(\forall r_1 \in T_1.res : \exists r_2 \in T_2.res : r_1 \sqsubseteq r_2) \wedge \tag{1.2}$$

$$(\forall a_1 \in T_1.act : \exists a_2 \in T_2.act : a_1 \sqsubseteq a_2) \tag{1.3}$$

The relation $a \sqsubseteq b$ with a and b both being either subjects, resources or actions denotes that *a is subordinate or equal to b regarding the authorization context*. This means that less privileges are associated with a than with b. A Target is said to be subordinate to another target if it applies to less subjects, resources as well as actions. Nevertheless, Def. 1 demands more than a common subset relation: According to (1.2) (respectively (1.3)), for every element r_1 of $T_1.res$ ($a_1 \in T_1.act$) an element r_2 of $T_2.res$ ($a_2 \in T_2.act$) must exist with r_1 being included in r_2 (a_1 being a sub-action of a_2). The opposite applies to subjects: (1.1) states that for a requestor it must be harder to fulfill s_1 than s_2, thus s_1 being of higher order in the sense of authorization than s_2.

	Target T_1		**Target T_2**
subjects	{chief-physician}	$\sqsupseteq$	{physician}
resources	{InPatient.Therapy}	$\sqsubseteq$	{InPatient}
actions	{select}	$\sqsubseteq$	{update, select}

Fig. 6. Example of a target comparison: T_1 is subordinate to T_2.

Let's assume that physicians are granted the permissions to execute update and select on the table *InPatient*. The appropriate target T_2 is depicted in Fig. 6. T_1 constitutes a subordinate version of T_2 because of the following reasons: First of all, T_1 applies to chief physicians. In the meaning of RBAC, chief-physician constitutes a senior role of physician. As senior roles acquire the permissions of their junior roles, the subject of T_2 is subordinate to the subject of T_1. Second, T_1 applies to a restricted resource, as the column *Therapy* is only an extract of the complete table *InPatient*. Third, the set of operations is reduced to select.

A rule R consists of a target ($R.tgt$), an optional condition ($R.cnd$) and its effect ($R.efc$) that can be either `Permit` or `Deny`. Def. 2 defines a partial order for rules.

Definition 2. *Partial order for rules*
A rule R_1 is equal to or more restrictive than a rule R_2, $R_1 \sqsubseteq R_2$, if

$$\begin{aligned}&((R_1.efc = R_2.efc = Permit) \wedge (R_1.tgt \sqsubseteq R_2.tgt) \wedge \\ &(R_1.cnd \sqsubseteq R_2.cnd)) \vee\end{aligned} \tag{2.1}$$

$$\begin{aligned}&((R_1.efc = R_2.efc = Deny) \wedge (R_2.tgt \sqsubseteq R_1.tgt) \wedge \\ &(R_2.cnd \sqsubseteq R_1.cnd))\end{aligned} \tag{2.2}$$

(2.1) denotes the case that both rules express permissions. The target of the more restrictive rule has to be more specific and the condition of the rule must be retained or even strengthened. As a consequence R_1 denotes a restricted privilege compared to R_2. Analogously (2.2) states that R_1 constitutes a more general prohibition and therefore is also more restrictive than R_2. According to this, the relationship of rule sets can be defined as follows:

Definition 3. *Partial order for rule sets*
A set of rules RS_1 is equal to or more restrictive than a set of rules RS_2, $RS_1 \sqsubseteq RS_2$, if and only if

$$(\forall R_1 \in RS_1 : R_1.efc = Permit \Rightarrow \exists R_2 \in RS_2 : R_1 \sqsubseteq R_2) \land \tag{3.1}$$

$$(\forall R_2 \in RS_2 : R_2.efc = Deny \Rightarrow \exists R_1 \in RS_1 : R_1 \sqsubseteq R_2) \tag{3.2}$$

Please note that $\sqsubseteq$ refers to access rights granted. Thus (3.1) states that privileges are restricted while (3.2) constitutes an expansion of prohibitions.

With Def. 4 we provide a partial order for policies. As commercial database systems commonly do not support the declaration of prohibitions, we use the following restrained definition:

Definition 4. *Partial order for (permissive) policies*
Let P_1 and P_2 be policies with the rule combining algorithm of P_1 as well as of P_2 being permit-overrides and the effect of any rule of P_2 being Permit. P_1 is equal to or more restrictive than P_2, in short $P_1 \sqsubseteq P_2$, if the target T_1 of P_1 is equal to or subordinate to the target T_2 of P_2, and its set of rules is equal to or more restrictive than the set of rules of P_2.

Def. 4 does not exclude that P_1 contains prohibitions in contrast to policy P_2. A comparison of the semantics of two policies is undecidable in the general case because of the expressiveness of rule conditions. As the access control rules of database systems are not restricted by additional conditions, policy comparison is eased and computable.

4 Engineering Adaptable Access Control Policies

The reliable definition of service policies can take place in two ways. One approach is to extract and adapt database policies. An alternative is to generate the service policies based on the predefined service functionality.

Adapting database policies. This approach is practical for administering the access control of already implemented Web services. In this case the service functionality as well as the database interaction are already given. As depicted in Fig. 1, the access control rules that apply to the user, whose account is employed to establish the connection to the database, are extracted. These constitute the maximum set of privileges that can be granted to the Web service users. The privileges are stored in *permission policy* sets as presented in Sect. 3. The Web

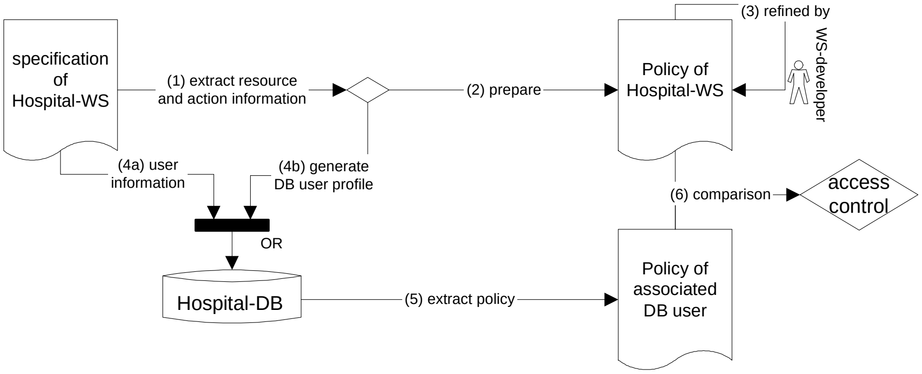


Fig. 7. Policy development process

service developer is able to select the *permission policies* that are applicable for the service operation. The rights can be further adapted and restricted by additional conditions. Finally, by the definition of *base policies*, these possibly modified privileges are assigned to user profiles of the Web service domain. In order to guarantee that the service policies are supported by the database, they are compared to the extracted database policies according to Def. 4.

Defining policies – starting from the service's point of view. Reliable Web service development starts with the specification of the desired functionality. As the service interacts with one or more underlying databases, service operations are related to SQL-statements. Thus, it is recommended to generate the service policy based on the specification (steps (1) and (2) of Fig. 7). Similar to the previous approach, the service developer can restrict the service policy by additional conditions (step (3)): Considering our Web service portal for physicians, the constraint of granting only attending physicians access to their patients' medical records can be formulated.

The extraction of the database policy is illustrated in the lower part of Fig. 7. This policy depends on the database account that is used to run interactions with the DBMS. Step (4a) represents the case that information about an already existing profile is provided by the service specification document, while step (4b) outlines that a new profile is generated that is appropriate for the service functionality. In step (5) the access rights of the respective profile are extracted and converted into XACML policies.

In the general case the Web service provides its own user management that differs from the one of the database. This is reasonable as the DBMS shall not be visible to service users. Consequently, in order to enable comparison of service policies and database policies (step (6)), service users have to be associated with database accounts. This can be achieved by the use of policies that are similar to the *role assignment policies* presented in the previous section.

Multiple policies and exchange of policies. Web service environments are highly dynamic, especially when services constitute mobile code that can be executed on arbitrary hosts as in the case of ServiceGlobe. In order to show how the presented techniques can be utilized for a flexible authorization design, we focus on a new use case for our hospital scenario. We assume that an e-service named *Management-WS* for administrative officers exists, which is used to inquire and change information about physicians as well as the allocation of beds. As illustrated in Fig. 8 the *Employee Policy* regulates the access for hospital employees on the side of the Web service. Connection to the DBMS is established under the account of *db_user1*. Assuming that an emergency situation arises, designated experts need to gather information about human resources and free capacities from institutions like police and fire departments, aid organizations and hospitals, in order to manage the catastrophic event. In the normal case, access for persons of foreign domains will be denied. By temporarily adding an appropriate policy that is tailored to the requirements of *Emergency coordinators*, access can be granted as long as the emergency situation persists and be revoked afterwards. In Fig. 8 an additional switch of the database connection is shown. This approach is advisable when the required access rights for the database vary in both situations.

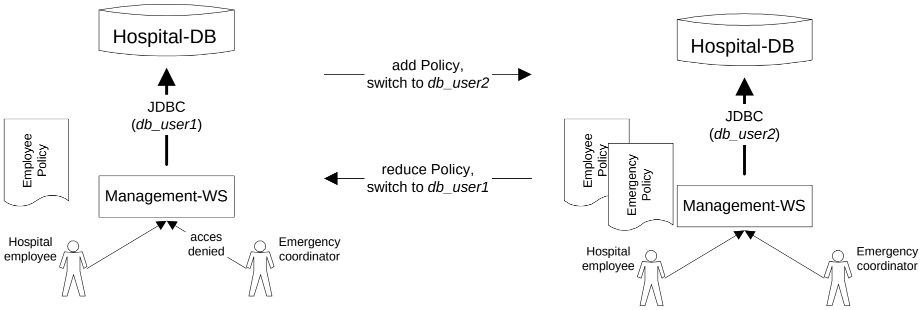


Fig. 8. Dynamic exchange of policies.

Thus, instead of developing partially redundant services for different situations, existing ones can be reused. Obviously this approach is scalable and flexible because the ports of services can be independently adapted to different situations by the dynamic exchange of policies.

5 Related Work

Several research platforms that are concerned with the access control management of distributed systems have been presented, among the first ones *Policy-Maker* and *KeyNote* [10]. Some more recent approaches deal with the security management of Web service applications, e.g., [11] shows how to define XACML policies for service interfaces and [12] presents an RBAC policy framework for Web services. Typically, it is entirely up to the service developer to define the access policies for services. To the best of our knowledge there does not exist any security system that focusses as much on the relationship between service policies and the access control of underlying resources – especially databases – like our approach.

Web service platforms and development toolkits like Bea WebLogic, IBM WebSphere or the freely available JWSDP employ Java 2 security technology, which enables the realization of sophisticated authentication and authorization for most Web applications. Nevertheless business and security logic are oftentimes admixed. In our approach, authorization and service implementation are decoupled, thus easing the maintenance of applications.

Data Grids, as representatives of distributed systems, provide the infrastructure for sharing resources in wide area networks, involving numerous data providers as well as consumers. The Community Authorization Service (CAS) approach [13] enables resource administrators to delegate the authorization management to the community. [14] describes an approach for fine-grained authorization for single resources. As the new grid platform Open Grid Service Architecture (OGSA) introduces Web service techniques into the Grid area, mechanisms for security management for e-services are also employed in the Grid context. In [14] it is mentioned that the use of XACML would be a practical alternative to the previously employed policy language. But resource administrators have to ensure on their own that the deployed access policy conforms with the authorization of the underlying resource. Our approach addresses this issue for databases by verifying that service policies constitute valid refinements of database policies.

6 Conclusion and Future Work

We have described our approach of a more reliable and adaptable engineering of access control for database dependent Web services: Service developers are assisted by the preparation of policy and request documents. Considering this, one approach is to extract the authorization settings of the underlying databases and use them as a basis for a subsequent definition of service policies. As an alternative access control of services can be designed based on service specifications. By verifying that the access control of a service constitutes a valid refinement of the corresponding database policies, service deployment is getting more reliable. In this way, access control is relocated to the Web application and adapted to the requirements of the Web service, which thus operates as a gatekeeper for the database as, by blocking unauthorized access at an early stage, denial of service attacks on the underlying database can be prevented. In contrast to database systems, services can be replicated more easily. Additionally, security for underlying databases is improved by adequate user profiles that are generated automatically with regard to service requirements. Thus security and privacy threats in case of service infiltrations are reduced.

A further significant aspect is the separation of security and business logic. Because of the application logic not being mixed with the security functionality, concise software design is enabled and code adaptations in case of policy changes are avoided. Web services are designed for application to application communication. Thus, the dynamic creation of service coalitions has to be considered. Further research topics include the delegation of rights and roles across administrative domains.

References

1. E. Maler, P. Mishra, and R. Philpott, "Assertions and Protocol for the OASIS Security Assertion Markup Language (SAML) v1.1," specification, OASIS, Sept. 2003.
2. B. Atkinson, G. Della-Libera, S. Hada, M. Hondo, and P. H.-B. et. al., "Web Services Security (WS-Security)," specification, IBM, Microsoft Corp., VeriSign Inc., April 2002.
3. S. Castano, M. G. Fugini, G. Martella, and P. Samarati, *Database Security.* ACM Press, Reading, MA, USA: Addison-Wesley, 1995.
4. R. Sandhu, D. Ferraiolo, and R. Kuhn, "The NIST Model for Role-based Access Control: Towards a unified standard," *proceedings of the 5th ACM Workshop on Role-Based Access Control*, pp. 47–64, July 2000.
5. D. F. Ferraiolo, R. Sandhu, S. Gavrila, D. R. Kuhn, and R. Chandramouli, "Proposed NIST Standard for Role-Rased Access Control," *ACM Trans. Inf. Syst. Secur.*, vol. 4, no. 3, pp. 224–274, 2001.
6. S. Godik and T. Moses, "eXtensible Access Control Markup Language (XACML)," specification, OASIS, February 2003.
7. M. Keidl, S. Seltzsam, K. Stocker, and A. Kemper, "ServiceGlobe: Distributing E-Services across the Internet (Demonstration)," in *Proc. of the Conf. on Very Large Data Bases (VLDB)*, pp. 1047–1050, 2002.
8. S. Seltzsam, S. Börzsönyi, and A. Kemper, "Security for Distributed E-Service Composition," in *Proc. of the 2nd Intl. Workshop on Technologies for E-Services (TES)*, vol. 2193 of *Lecture Notes in Computer Science (LNCS)*, pp. 147–162, 2001.
9. A. Anderson, "XACML RBAC Profile," working draft, OASIS, June 2003.
10. M. Blaze, J. Feigenbaum, and A. D. Keromytis, "The Role of Trust Management in Distributed Systems Security," in *Secure Internet Programming*, pp. 185–210, 1999.
11. T. Moses, "XACML profile for web-services," working draft, OASIS, September 2003.
12. R. Bhatti, J. B. D. Joshi, E. Bertino, and A. Ghafoor, "Access Control in Dynamic XML-based Web-Services with X-RBAC," 2002.
13. V. Welch, F. Siebenlist, I. Foster, J. Bresnahan, K. Czajkowski, J. Gawor, C. Kesselman, S. Meder, L. Pearlman, and S. Tuecke, "Security for Grid Services," tech. rep., Globus Alliance, June 2003.
14. K. Keahey, V. Welch, S. Lang, B. Liu, and S. Meder, "Fine-Grain Authorization Policies in the GRID: Design and Implementation," tech. rep., 1st International Workshop on Middleware for Grid Computing, 2003.