

Technische Universität München  
Fakultät für Informatik  
Lehrstuhl III – Datenbanksysteme



---

## Efficient Access Control for Service-oriented IT Infrastructures

Diplom-Informatiker Univ.  
Martin Rudolf Wimmer

Vollständiger Abdruck der von der Fakultät für Informatik der Technischen Universität München zur Erlangung des akademischen Grades eines

**Doktors der Naturwissenschaften (Dr. rer. nat.)**

genehmigten Dissertation.

Vorsitzender: Univ.-Prof. Dr. Helmut Krçmar

Prüfer der Dissertation:

1. Univ.-Prof. Alfons Kemper, Ph. D.
2. Univ.-Prof. Dr. Joachim Posegga,  
Universität Hamburg

Die Dissertation wurde am 18.12.2006 bei der Technischen Universität München eingereicht und durch die Fakultät für Informatik am 08.05.2007 angenommen.

---



## Acknowledgments

Above all, I would like to thank Prof. Alfons Kemper, Ph.D., my supervisor at the TU München. He supported me with valuable comments and provided encouragement and guidance throughout my work on this thesis. I enjoyed working with his research group and appreciated the very pleasant working atmosphere created by my colleagues Stefan Aulbach, Daniel Gmach, Benjamin Gufler, Stefan Krompaß, Richard Kuntschke, Angelika Reiser, Tobias Scholl, Stefan Scholz, Stefan Seltzsam, and Bernhard Stegmaier. Not to forget the research group of Prof. Dr. Torsten Grust, Jan Rittinger and Jens Teubner, and my former colleagues at the University of Passau, Markus Keidl, Christian Wiesner, and Bernhard Zeller. I also would like to thank Evi Kollmann, the good soul of the group.

I also thank Martina-Cezara Albutiu, Daniela Eberhardt, Pia Ehrnlechner, Armin Fischer, Jakob Gajdzik, and Alexander Schuster, whose theses I supervised. They did an excellent work on implementing prototypes which allowed me to test and improve the access control concepts presented in this thesis. Policy consolidation strategies have been put to the test in the context of a cooperation with SAP Research, for which I want to thank Volkmar Lotz and Maarten Rits of SAP Research, Sophia Antipolis. I also thank Prof. Dr. Joachim Posegga for being the second supervisor of my thesis.

I thank all the people who gave me helpful criticism and advice on my doctoral thesis. In particular, I want to thank Richard Kuntschke for his thorough and fast proofreading, giving me valuable support to improve my phrasing.

Finally, I want to say thanks to my family and especially to my wife Andrea for supporting me in all my endeavors and for reminding me that there are still other things out there apart from computers and work.

München, June 2007

Martin Wimmer



## Abstract

Web services represent the emerging technology for many enterprise application architectures. Due to widely accepted standards for the specification of service interfaces and communication protocols, they constitute the preferred approach for integrating resources and legacy systems, easing the reusability of modules and the reconfiguration of higher-order business processes. Furthermore, inter-organizational value creation chains can be realized by the seamless integration of distributed services. Besides these amenities, the emerging service-oriented computing concepts also introduce new security challenges. In this thesis, we present flexible authorization techniques providing efficient access control for service-oriented IT infrastructures. In particular, the proposed authorization strategies (1) provide efficient access control for intra-organizational service compositions, (2) support the reliable integration of resources like database systems into service-oriented architectures, and (3) enable optimized policy enforcements in dynamic Web service coalitions.

As Web services represent self-contained modules that autonomously enforce security, user requests are iteratively authorized and evaluated by service compositions. Apart from repeated and possibly redundant authorization checks, performance drawbacks can arise due to unnecessary service executions. This happens, for example, if users are allowed to execute certain sub-processes but lack authorizations for later stages of a workflow. Furthermore, considering the execution of Web service transactions, this can demand for rollbacks or costly compensating transactions. These drawbacks can be avoided through the early filtering of ultimately unauthorized requests, thus, providing solutions for issue (1). Our contributions are a formal model and algorithmic solutions for consolidating the access control of composite applications. We demonstrate by means of Web service workflows how access control can be shifted to the layer of the composite application, thus, reducing policy enforcement costs.

Considering issue (2), we propose a security engineering approach for the reliable implementation of database backed Web services. Today, often over-privileged database authorities are used to realize the interaction between services and underlying database systems. In case of security vulnerabilities on the services' side, confidential data is in danger of being disclosed. In this thesis, we describe our approach for the semi-automatic generation of service interfaces that realize the principle of least privilege. Based on the specification of the service-to-database interaction, the access control of Web services is defined and consolidated with the security configuration of the underlying database systems.

In order to provide access on local resources within larger collaboration networks, privileges need to be granted to entities of cooperating domains. We present our authorization infrastructure supporting the delegation of privileges and roles across organizations. Access control in loosely coupled federations is performed through the interplay of local and distributed policy enforcements. By use of adequate caching techniques, our proposed access control strategy is also applicable for large-scale and dynamically growing coalitions, as addressed by issue (3). Hence, the combination of the proposed techniques is an approach to provide efficient and flexible access control for service-oriented IT infrastructures.



---

# Contents

---

<b>1</b>	<b>Introduction and Overview</b>	<b>1</b>
1.1	Classification of Service Compositions . . . . .	3
1.2	Contributions . . . . .	5
1.3	Outline . . . . .	6
<b>2</b>	<b>Access Control Models and Terminology</b>	<b>7</b>
2.1	The Role of Access Control . . . . .	7
2.2	Access Control Models . . . . .	9
2.2.1	Mandatory Access Control . . . . .	9
2.2.2	Discretionary Access Control . . . . .	10
2.2.3	Role Based Access Control . . . . .	11
2.2.4	Administration of Authorization . . . . .	12
2.3	Access Control Requirements of Service-oriented Architectures . . . . .	13
2.4	Design Principles of our Authorization Framework . . . . .	14
<b>3</b>	<b>Optimized Access Control for Composite Applications and Workflows</b>	<b>17</b>
3.1	Motivation . . . . .	18
3.2	Policy Model . . . . .	20
3.2.1	Notation . . . . .	20
3.2.2	Semantics . . . . .	22
3.2.3	Policy Combining Operators . . . . .	23
3.3	Policy Consolidation . . . . .	24
3.3.1	Problem Specification . . . . .	25
3.3.2	Workflow Dependencies . . . . .	25
3.3.3	Analysis of SEQUENCE Patterns . . . . .	26
3.3.4	Analysis of SWITCH Patterns . . . . .	28
3.3.5	Structural Analysis . . . . .	28

3.3.6	Evaluation of the Policy Consolidation Approach . . . . .	30
3.4	Algorithmic Solutions . . . . .	32
3.4.1	Implementing the Conjunction Operator . . . . .	32
3.4.2	Checking Privilege Relaxation . . . . .	33
3.4.3	Implementing the Subtraction Operator . . . . .	37
3.5	Optimizing the Access Control of Intra-organizational Web Service Workflows . . . . .	37
3.5.1	Running Example . . . . .	37
3.5.2	Performing Policy Consolidation . . . . .	41
3.5.3	Implementation . . . . .	44
3.6	Related Work . . . . .	46
3.7	Conclusion . . . . .	49
<b>4</b>	<b>Security Engineering for Database Backed Web Services</b>	<b>51</b>
4.1	Motivation . . . . .	52
4.2	Access Control of Database Systems and Web Services – the two Poles Apart . . . . .	55
4.2.1	Access Control of Database Management Systems . . . . .	55
4.2.2	Access Control Mechanisms for Web Services . . . . .	58
4.2.3	Access Control of Database Web Services . . . . .	60
4.3	Security Engineering for Database Web Services – Bridging the Gap . . . . .	64
4.3.1	Determining the Least Required Privileges . . . . .	64
4.3.2	Automated Policy Generation . . . . .	75
4.3.3	Extraction of Database Policies . . . . .	76
4.3.4	Engineering Adaptable Access Control Policies . . . . .	76
4.4	Implementation . . . . .	79
4.5	Related Work . . . . .	82
4.6	Conclusion . . . . .	84
<b>5</b>	<b>Access Control in Dynamic Service Coalitions</b>	<b>85</b>
5.1	Motivation . . . . .	86
5.2	Extended Policy Model . . . . .	88
5.2.1	Terminology and Notation . . . . .	88
5.2.2	Multistep Delegations . . . . .	91
5.2.3	Policy Representation and Implementation . . . . .	91
5.2.4	Revocation Schemes . . . . .	92
5.3	Policy Evaluation . . . . .	96
5.3.1	Local Policy Evaluation . . . . .	96
5.3.2	Distributed Policy Evaluation . . . . .	97
5.3.3	Example . . . . .	99
5.4	Caching of Authorization Paths . . . . .	101
5.4.1	Caching Strategies . . . . .	103
5.4.2	Experimental Results . . . . .	104
5.5	Application Scenarios . . . . .	107
5.5.1	Support of Loosely and Tightly Coupled Federations . . . . .	107



---

5.5.2	Treating Revocations During (Long-lasting) Transactions . . . . .	108
5.6	Related Work . . . . .	108
5.7	Conclusion . . . . .	111
<b>6</b>	<b>Conclusion</b>	<b>113</b>
<b>A</b>	<b>Graphical Workflow Notation</b>	<b>115</b>
<b>B</b>	<b>Probabilistic Performance Estimation of Policy Comparisons</b>	<b>116</b>
<b>C</b>	<b>Policy Representation</b>	<b>119</b>
C.1	Permission Policies . . . . .	120
C.2	Base Policies . . . . .	121
C.3	Role Assignment Policies . . . . .	122
C.4	Role Delegation and Revocation Policies . . . . .	123
	<b>Bibliography</b>	<b>127</b>

---

## List of Figures

---

1.1	Traditional and upcoming service-oriented middleware infrastructures . . . . .	2
1.2	Classification of composite applications depending on locality and coupling . . .	4
2.1	Policy enforcement strategies . . . . .	8
2.2	Example role hierarchies . . . . .	11
3.1	Multilayered architecture of a hospital's accounting system . . . . .	18
3.2	Employed policy model . . . . .	21
3.3	Policy enforcement strategies . . . . .	23
3.4	Composite patterns . . . . .	27
3.5	Prerequisites for the consolidation of $P_0$ w.r.t. $P_1$ and $P_2$ . . . . .	28
3.6	Tree representation of the composite application $APP_0$ illustrated in Figure 3.4(a)	29
3.7	Example of a loop nested switch . . . . .	30
3.8	Algorithm intersect . . . . .	33
3.9	Matching conjunctive terms . . . . .	34
3.10	Algorithm implies . . . . .	35
3.11	Algorithm subtract . . . . .	35
3.12	Visualization of predicate subtraction . . . . .	36
3.13	Example of an e-health (Web service) workflow . . . . .	38
3.14	BPEL4WS-extract and workflow tree representation of the e-health process . . .	40
3.15	Optimizing the access control through policy enforcements at the workflow layer	43
3.16	Processing steps of the policy consolidation prototype . . . . .	45
3.17	SAP Research's workflow management system . . . . .	46
3.18	Policy consolidation within SAP Research's workflow management tool-suite . .	47
4.1	Reducing security vulnerabilities through access corridors . . . . .	53
4.2	Architecture of a simple database service . . . . .	54
4.3	Access control granularity levels of RDBMS . . . . .	55

---

4.4	Control flow of the policy enforcement process . . . . .	59
4.5	XACML policy model . . . . .	60
4.6	Parse tree . . . . .	73
4.7	Dependency graph . . . . .	74
4.8	Semi-automatic security engineering for database Web services . . . . .	78
4.9	Adaptive policy management . . . . .	79
4.10	Example for integrating database queries into XACML policies . . . . .	81
4.11	Policy evaluation process of the ServiceGlobe security system . . . . .	82
5.1	Intra-organizational and remote service executions . . . . .	87
5.2	Extended policy model supporting role delegations . . . . .	90
5.3	Effects of different revocation schemes . . . . .	94
5.4	Distributed policy evaluation . . . . .	99
5.5	Distributed policy evaluation in a dynamic coalition scenario . . . . .	100
5.6	Experimental settings and results . . . . .	106



## Introduction and Overview

---

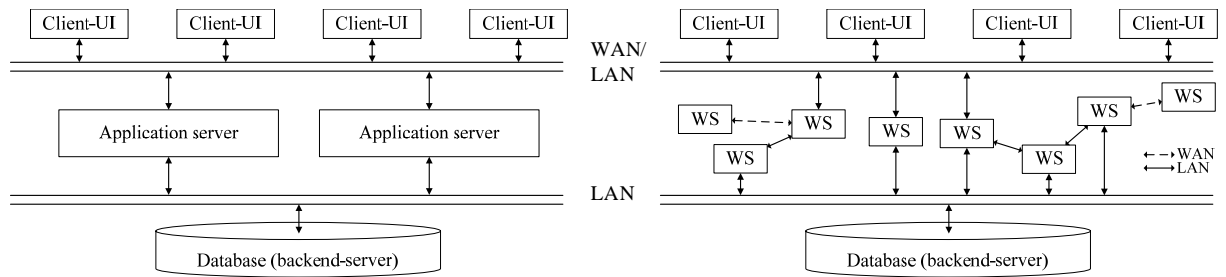
A company's economic success depends to a great extent on its possibilities to quickly react upon market changes by adapting its portfolio and services to the customers' needs. This demand for flexibility is also reflected by the design of current and future information systems. Monolithic middleware systems are increasingly replaced by service-oriented architectures. One of the most prominent representatives confirming this development is SAP's Enterprise Service Architecture (ESA) with its enterprise services repository and the integration platform NetWeaver which are aiming to supersede SAP R/3.<sup>1</sup> In the course of this technological trend, applications are decomposed into their basic functionalities which are provided in form of self-contained software components. These can flexibly be linked to realize higher-order business processes. Thus, service-oriented architectures often better support the reuse of modules and the reconfiguration of business processes than traditional middleware architectures.

Apart from e-commerce, the service-oriented computing paradigm also found its way into e-science and e-health applications. Massively distributed applications can be realized through inter-organizational service orchestrations. For instance, astrophysical observations can be organized via grid technology or diagnostic findings can be exchanged among physicians and medical specialists. The most widely used technology for the realization of service-oriented middlewares are Web services that excel in high interoperability. They allow higher-order business processes to be realized via compositions of specialized services. Thereby, service compositions can include local as well as distributed services of various service providers as indicated in Figure 1.1(b). This is enabled by widely adopted standards for service description and discovery as well as communication protocols.<sup>2</sup> The most important specifications in this regard are the XML-based Web Services Description Language (WSDL), Universal Description, Discovery and Integration (UDDI), and the Simple Object Access Protocol (SOAP).

---

<sup>1</sup>See [Woods (2004)].

<sup>2</sup>See [Weerawarana et al. (2005)].



(a) Traditional three-tier architecture (adopted from [Kemper and Eickler (2006)])

(b) Three-tier architecture with service-oriented middleware

Figure 1.1: Traditional and upcoming service-oriented middleware infrastructures

On the one hand, service-oriented computing enables high interoperability and flexibility. On the other hand, it also brings about new security concerns. In his talk at the ETRICS 2006 conference, Sachar Paulus, chief security officer of SAP, pointed out that the emergence of service-oriented architectures, in particular the decomposition of applications into autonomous services, led to

*“A system security mindset shift:  
From building and managing elephants  
to maintaining and directing ant colonies.”*

As services are accessible via Internet protocols like HTTP, they demand for secure execution, in particular, secure messaging over potentially insecure channels. Thus, similar to classical Web applications, message integrity and confidentiality have to be ensured.<sup>3</sup> By use of existing security standards like the Security Assertion Markup Language (SAML) and the WS-Security framework, security for service-oriented architectures can be implemented reliably.<sup>4</sup> Nevertheless, efficiency remains an open issue. In the above allegory, the ants (i.e., the services) represent autonomous individuals. From the security viewpoint this denotes that services have their own security requirements which they enforce on their own. Thus, access control of business processes that are realized by means of service compositions is conducted iteratively. Many business processes like order processing or financial accounting are rarely modified. Hence, they constitute more or less static workflows which are executed frequently. For such processes, one obvious drawback of the separation of security enforcements is its negative impact on performance. Therefore, new strategies are required to make policy evaluation of service compositions efficient enough for large-scale applications.

As Web services are used to integrate existing applications and legacy systems into service-oriented middlewares,<sup>5</sup> security not only needs to be ensured on the services' layer, but also for the underlying resources. Typically, as highlighted in Figure 1.1(b), many enterprise services

<sup>3</sup>See [Wimmer et al. (2006c)].

<sup>4</sup>See [Cantor et al. (2005)] and [Nadalin et al. (2006)].

<sup>5</sup>See, for example, [Linthicum (2001, 2003)] and [Weilbach and Herger (2005)].

rely on database functionality. While it can be meaningful for traditional application servers to provide full access on underlying database systems, the same does not hold for services that usually only require access to a rather limited extract of the data. In case over-privileged accounts are used to interface with the database systems, malicious users succeeding in exploiting security flaws in service implementations will also obtain comprehensive access permissions on the database systems. In order to avoid such threats, the access control of services needs to be restricted to the least required privileges.

Web services not only provide the technological basis for the mentioned intra-company application integration but, above all, are particularly suitable for realizing inter-organizational value creation chains. Handling requests on distributed resources in a network of cooperating organizations demands for a flexible and capable authorization framework. Many distributed e-science networks are examples of dynamic virtual organizations where partners can join and resign on demand. In order to enter a community, access rights have to be granted to members of the new partner organizations. Depending on the assumed trust relationships, various privilege delegation schemes and designs of collaboration networks are possible. For instance, access control can be realized by a central authority in case of tightly coupled partners that trust each other. Different access control strategies are required for dynamic service coalitions where the organizations have to retain their authorization autonomy. To give an example, access control that is based on the identity of users usually constitutes a limiting factor for dynamically growing collaboration networks. Instead, authorization has to be enforced based on the requesters' characterizing attributes like their job profiles.

## 1.1 Classification of Service Compositions

Web service choreographies and generic composite applications can be classified with regard to the dimensions locality and coupling as illustrated in Figure 1.2. Locality denotes the differentiation between intra- and inter-organizational composite applications, depending on whether only local services or also distributed resources of cooperating service providers are used. Due to sub-activities autonomously enforcing security, additional overhead arises when performing access control of composite applications. Furthermore, the individual security policies might be conflicting, hindering the successful execution of a composite application. Hence, a prerequisite for the reliable implementation of intra-organizational composite applications is the consolidation of access control policies. In order to enable inter-organizational workflows, the authorization infrastructure has to support the integration of distributed authorization schemes. For members of a federation being able to access distributed resources, access rights need to be assigned across administrative boundaries. Thus, by means of privilege delegations, collaboration networks can be established.

Access control of composite applications can be performed either in a centralized or decentralized manner, depending on the type of coupling. A system is called tightly coupled if its underlying resources are permanently integrated. For example, classical enterprise resource planning systems are rather monolithic middleware systems with the underlying database systems usually being tightly integrated. Analogously, database backed services can be regarded

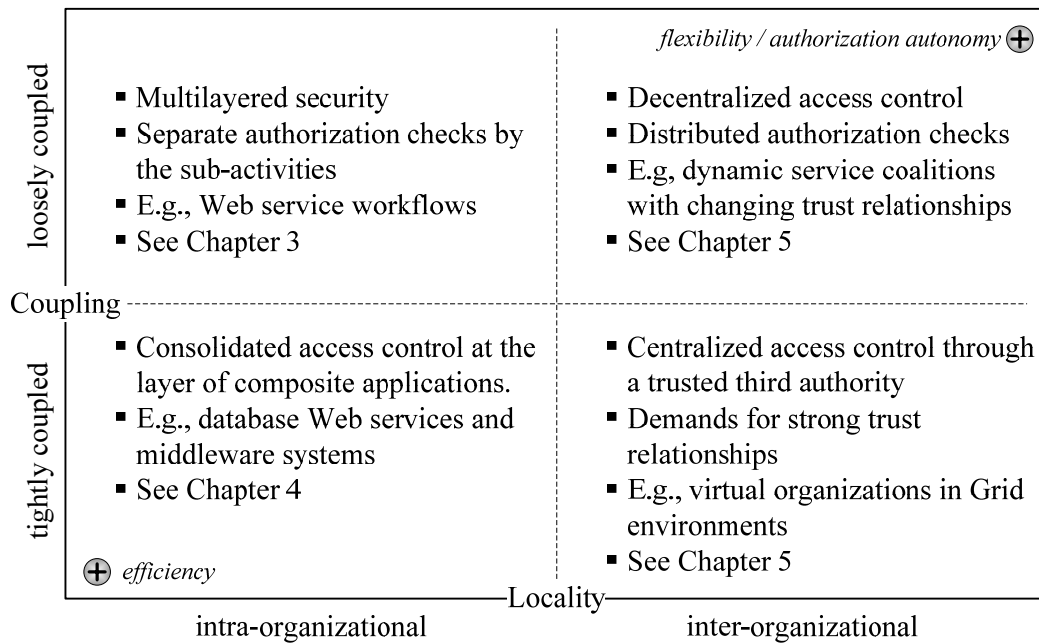


Figure 1.2: Classification of composite applications depending on locality and coupling

as tightly coupled composite applications as well. Loosely coupled systems on the other hand allow dynamic substitutions of services and integrated resources. That is, services can flexibly be integrated and replaced on demand, which is also referred to as dynamic service selection.<sup>6</sup> While in tightly coupled systems access control can often be realized in a centralized manner, resources and services that are loosely interlinked need to retain their authorization autonomy. In loosely coupled distributed systems, these characteristics demand for decentralized, distributed policy evaluation.

The various design principles have a strong impact on the efficiency of access control and on the autonomy of authorization. Referring to Figure 1.2, efficiency decreases from the lower left corner to the upper right corner, while flexibility increases in the same direction. Access control for tightly coupled intra-organizational composite applications can be realized efficiently as authorizations can be inferred by means of a centralized policy evaluation. However, tightly coupled inter-organizational coalitions imply that cooperating partners give up at least part of their authorization autonomy by delegating access control to a trusted authority. In contrast to this, authorization autonomy is preserved within dynamic coalitions that provide high levels of adaptability and interoperability. Therefore, dynamic coalitions are well applicable for cooperations with unstable trust relationships but flexibility has to be traded for increased policy evaluation complexity.

<sup>6</sup>See [Keidl (2004)].



## 1.2 Contributions

In this thesis, we present flexible authorization techniques providing efficient access control for service-oriented IT infrastructures. In particular, we present an optimization approach for reducing policy evaluation costs of intra-organizational composite applications and Web service workflows. Furthermore, we describe best practices for the reliable integration of database functionality into service-oriented middleware systems. Based on these techniques, we present our authorization framework for distributed Web service workflows. Altogether, the described models and enforcement strategies support the reliable integration of data sources into tightly and loosely coupled Web service federations.

We start with analyzing the access control of intra-organizational composite applications whose sub-activities are self-contained software components that autonomously conduct authorization checks. We arrive at the conclusion that this access control separation in general leads to redundant and therefore possibly inefficient policy enforcements. Performance drawbacks due to superfluous executions can be caused by users that are allowed to execute some sub-processes but are not entitled to perform later stages of a service choreography. Moreover, they can even demand for transaction rollbacks or the execution of compensating transactions. Our contributions are a formal model and algorithmic solutions for the consolidation of access control policies. Consolidated policies allow to shift access control to the composite application's layer, thus, making authorization checks by the individual sub-activities unnecessary and enabling the pre-filtering of requests. Hence, the mentioned drawbacks can be avoided and access control for composite applications can be realized more efficiently.

Many Web services rely on the data processing capabilities of database backend servers. In this regard, often over-privileged profiles are used to realize the service-to-database interaction, e.g., to keep administrative overhead low. Therefore, in case security vulnerabilities of service implementations are abused, confidential data are in danger of disclosure. In this thesis, we present a security engineering approach for the reliable implementation of database backed Web services. Our approach is based on the semi-automatic generation of security policies and database authorization profiles that are restricted to the required service functionality, thus, fulfilling the necessary principle of least privilege.

Providing shared access to local resources demands for possibilities to grant privileges to entities of cooperating domains. We present our authorization framework supporting tightly and loosely coupled Web service coalitions. The underlying authorization scheme supports the delegation of access rights to subjects of cooperating organizations, employing a flexible distributed role based access control model. Within loosely coupled coalitions, access control is performed by an interplay of local and distributed policy evaluation steps. The use of secure caching strategies makes this approach also suitable for large-scale and dynamically growing coalitions. Therefore, the proposed security concepts provide efficient and flexible access control for service-oriented IT infrastructures, supporting various designs of composite applications as depicted in Figure 1.2.

## 1.3 Outline

The thesis is organized as follows:

### **Chapter 2 – Access Control Models and Terminology**

gives an overview of existing access control models like mandatory access control, discretionary access control, and role based access control. Based on this outline, we discuss the security requirements of service-oriented architectures and present the basic principles of our own authorization scheme.

### **Chapter 3 – Optimized Access Control for Composite Applications and Workflows**

introduces the syntax and semantics of our policy model. We describe how policies of composite applications can be consolidated with the access control schemes of their underlying autonomous sub-activities. Policy consolidation constitutes the basis for optimizing access control. We demonstrate this by means of Web service workflows.

### **Chapter 4 – Security Engineering for Database Backed Web Services**

presents best practices for the reliable integration of database functionality into service-oriented IT infrastructures. Thereby, our main objective is to reduce security risks for the underlying database systems. We achieve this goal through semi-automatically consolidating the access control of database systems with the security configuration of the service frontends.

### **Chapter 5 – Access Control in Dynamic Service Coalitions**

presents our authorization infrastructure for distributed Web service applications. The underlying access control model supports cross-domain assignments and role delegations and is an extension of the authorization scheme presented in Chapter 3. The proposed framework allows to realize tightly as well as loosely coupled federations. Policy evaluation for loosely coupled coalitions is optimized by means of secure caching techniques.

### **Chapter 6 – Conclusion**

summarizes the thesis and shows how the combination of the proposed techniques provides efficient access control for service-oriented IT infrastructures.

---

# Access Control Models and Terminology

---

Access control is the process of intercepting user requests for evaluating whether the requested activities can be granted according to the system's security policy. In this chapter, we elaborate on basic access control models and terminology. After presenting the role of access control and its relationships to further security services in Section 2.1, we give an overview over widely used access control models in Section 2.2. The access control requirements of service-oriented architectures are discussed in Section 2.3. Subsequently, in Section 2.4, we motivate the basic design decisions of our authorization framework which we present in detail in the remainder of this thesis.

## 2.1 The Role of Access Control

The objective of access control is to prevent illegitimate access to protection objects, whereby access for authorized requesters must be ensured at the same time, of course. For that purpose, requests are intercepted and evaluated against applicable policies. A policy consists of individual authorization rules according to which access control is performed. Such access rules declare relationships between subjects and objects. They are categorized into privileges and denials, depending on whether they grant or prohibit actions on protection objects. In this context, subjects are active entities like human users. Examples for protection objects (which represent passive entities) are files managed by a file system or tables, views, or stored procedures of a database system. Applications can represent subjects as well as objects. They act as objects in case their execution is regulated by access rules. Applications represent active entities in case they access protection objects during their execution, thereby potentially acting on behalf of other subjects, e.g., human users.

Figure 2.1(a) schematically illustrates the access control process. Access control is subdivided into authentication and authorization. The function of authentication is to verify that the identity of a subject indeed coincides with the identity it claims to have. In the following, we

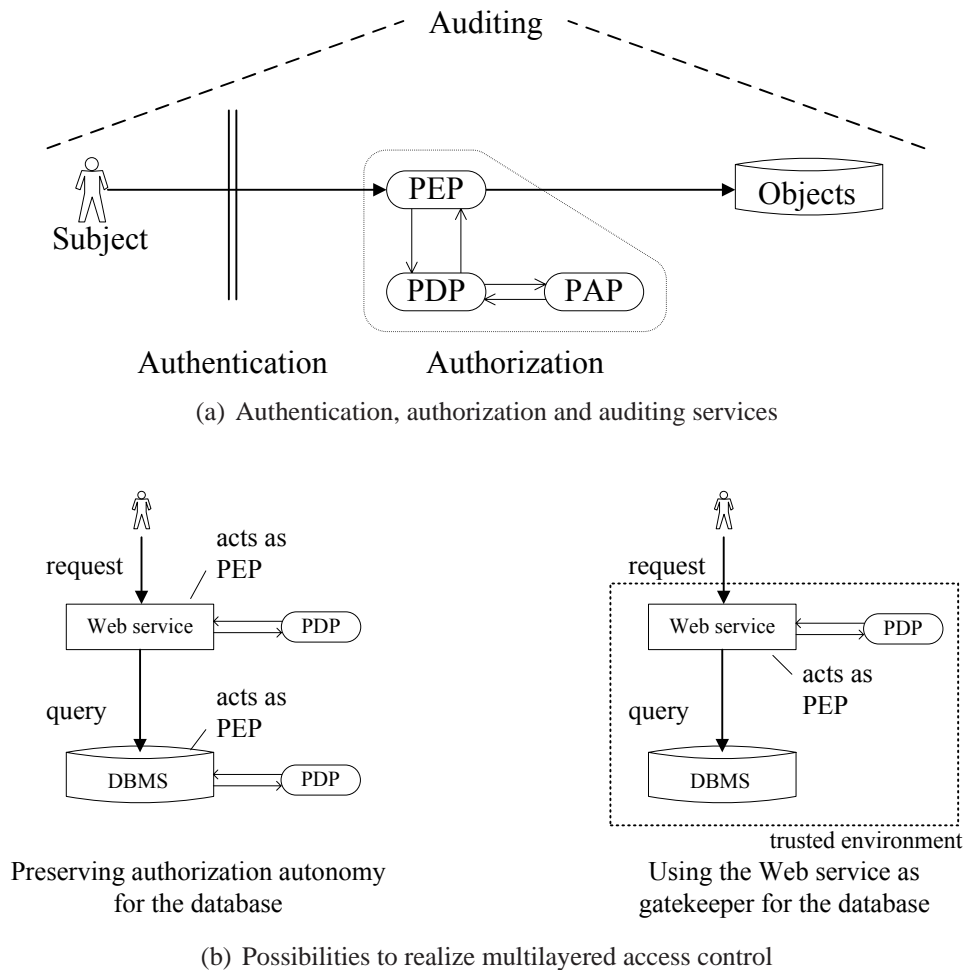


Figure 2.1: Policy enforcement strategies

will concentrate on authorization and assume that authentication has been correctly achieved. In service-oriented architectures, authentication of requesters is, for example, implemented by use of the Security Assertion Markup Language (SAML), specified by Cantor et al. (2005), or the WS-Security framework, introduced by Nadalin et al. (2006).

Authorization is the process of evaluating requests against applicable policies ascertaining that requested actions can be granted. This process is initiated by the so-called policy enforcement point (PEP), which can be the invoked service itself. The functionality for evaluating requests is provided by the policy decision point (PDP), e.g., realized as a dedicated authorization service. The PDP relies on the policy administration point (PAP) to determine the applicable policies. For instance, the PAP can realize the interface to a centralized policy repository. Furthermore, a PAP has to ensure that only legitimate entities can access policy information. From the software engineering perspective, it is recommended to separate PEP, PDP, and PAP functionality. In order to ensure maintainability, security functionality should be separated from the

application logic.

Figure 2.1(b) sketches two examples for realizing the access control of database backed Web services: In the left part of the figure, the service and the underlying database system are autonomously enforcing access control. In contrast to this, the right part of the figure illustrates a configuration where only the Web service enforces access control. As we will point out in Chapter 3 and 4, as a prerequisite for this optimization, the authorization policies of the service and the authorization configuration of the database system have to be consolidated.

As shown in Figure 2.1(a), access control is closely related to further important security services, like auditing. For auditing, user requests have to be logged. The evaluation of log information allows detecting flaws in the security system. Furthermore, it provides possibilities to detect and prove violation attempts. As most attacks and compromises are conducted from inside organizations,<sup>1</sup> auditing helps to supervise the behavior of users and to hold them accountable for their activities.

Often, authorization rules are defined in an informal way – at least at the initial phase of specifying access control. In order to compare policies and to enable their validation and evaluation by means of programs, they have to be expressed in a formal policy language. In the following, we briefly describe widely used access control schemes like mandatory access control, discretionary access control, and role based access control. Further (historical) background information concerning these access control models are, for example, provided by Castano et al. (1994), and Samarati and de Capitani di Vimercati (2001).

## 2.2 Access Control Models

### 2.2.1 Mandatory Access Control

Mandatory access control (MAC) regulates access on protection objects based on the sensitivity level of the information (also called the objects' classification) and the authorization level of subjects (the so-called clearance). Mandatory policies typically represent multilevel security schemes. The main field of application for mandatory policies are the military and governmental sectors because of stringent security requirements. The sensitivity of information can, for example, be classified as *confidential*, *secret*, and *top secret*, depending on whether their disclosure to unauthorized subjects is expected to cause some, serious, or grave damage. The classification of objects and subjects defines a partial order expressing dominance relationships. Subjects are authorized to access objects in case their clearance dominates the classification of the objects.

In order to ensure the secrecy of information, the information flow among subjects of different classification levels has to be controlled strictly. Bell and LaPadula (1973, 1976) proposed the so-called no-write-down and the no-read-up rules to regulate the propagation of sensitive information. The no-write-down paradigm dictates that subjects can only perform write access on protection objects whose classification dominates the subjects' clearance. No-read-up prohibits subjects to read information in case their clearance level does not dominate the protection objects' classification.

---

<sup>1</sup>See, for example, [Rosenberg and Remy (2004)], page 35.

While these rules ensure the secrecy of information, they cannot guarantee their integrity. Subjects with lower clearance could still indirectly initiate improper modifications to higher classified objects, leading to misinformation. Therefore, Biba (1977) proposed the no-read-down and no-write-up principles for enforcing information integrity: A subject is allowed to read an object only if the object's classification dominates the subject's clearance; and a subject is allowed to write an object only if the subject's clearance dominates the object's access class. These rules represent the contrary to the principles of the Bell LaPadula model. While the Bell LaPadula principles allow the information flow from lower to higher secrecy levels, the Biba model allows the information flow from higher to lower integrity levels. If both concepts have to be enforced, subjects and objects need to be classified individually for secrecy and integrity.

Though mandatory policies provide ways to control the information flow, they are often considered to be too restrictive and rigid for commercial scenarios, hindering operational procedures. Furthermore, MAC models can lead to the so-called polyinstantiation problem which denotes that the same real world fact is present in form of multiple instances that differ with regard to their classification, thus, violating data integrity constraints. Jajodia and Sandhu (1990, 1991) address this issue for multilevel database systems.

### 2.2.2 Discretionary Access Control

In discretionary access control (DAC) models, access rights restrict access on protection objects based on the identity of subjects or, in order to improve scalability, groups they belong to. The access matrix, which was proposed by Lampson (1974) and formalized by Harrison et al. (1976) declares for each combination of subjects and objects the set of allowed actions. It is called access matrix model as the access rights can be stored in a matrix with the columns and rows representing objects and subjects, respectively, and the entries being the granted privileges. Such policies are typically employed for operating systems. The model is called discretionary as subjects with certain control permissions are allowed passing privileges to other subjects (at their discretion). Often, the ownership paradigm is applied, denoting that the creator of an object is its owner who, by default, is granted control privileges for that object.

Several extensions of DAC models have been proposed like (hierarchical) group schemes for subjects, objects, and actions and the formulation of conditions and exceptions. Object hierarchies and privilege implications are for example treated by Rabitti et al. (1991) and group schemes for actions have been proposed by Shen and Dewan (1992). Conditions restrict the applicability of access rules. For instance, a condition can constrain access on a protection object to the working hours of a day, e.g., to the time between 8 am and 6 pm. The concept of exceptions leads to the differentiation between positive and negative access control rules, i.e., privileges and denials. Therefore, positive authorization systems, where only privileges can be defined, are to be distinguished from negative systems, where each rule represents a denial. Also mixed systems where policies can consist of a mixture of negative and positive access rules are possible. In general, policy specifications are not complete, i.e., cases can arise where neither privileges nor denials apply which have to be handled by a closure assumption. In this regard, the closed world assumption, where everything is denied unless a privilege can be inferred, or the open world assumption, where everything is allowed unless a denial is derived, can be applied. Positive sys-

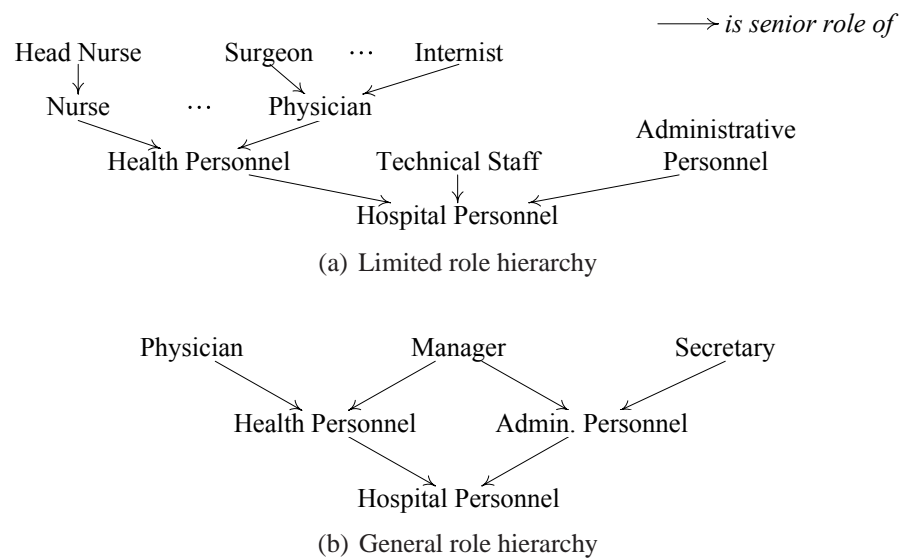


Figure 2.2: Example role hierarchies

tems are compatible with the closed world assumption and negative systems with the open world assumption. For mixed systems, both combinations are meaningful.

Mixed systems also demand for conflict resolution techniques, i.e., strategies that specify how situations when positive and negative rules apply at the same time are to be handled. Various approaches like privileges-take-precedence, denials-take-precedence or the most-specific-takes-precedence have been proposed. An overview over conflict resolution techniques is provided by Jajodia et al. (2001).

### 2.2.3 Role Based Access Control

The administration of DAC policies can easily become unmanageable if privileges are independently assigned to each user. Better scalability is provided by role based access control (RBAC) schemes which, for instance, have been proposed by Sandhu et al. (1996), Osborn et al. (2000), and Ferraiolo et al. (2001). In 2004, an official standard for RBAC has been released.<sup>2</sup> Using RBAC, privileges required for performing certain jobs are grouped by roles. Subjects acquire privileges via the indirection of being granted the needed roles. Therefore, as discussed by Sandhu (1996), (user) groups and roles describe two different concepts. Groups denote sets of users that can be described through the same attributes like their membership characteristics, while roles represent sets of privileges.

Role based access control is a means of mapping organizational structures onto access control policies. Not the identities of users determine their authorizations, but their job profiles. Users changing their jobs within a company are revoked their old roles and granted those roles which

<sup>2</sup>Confer [ANSI INCITS 359-2004].

are aligned to their new area of activity. Thus, RBAC provides high levels of flexibility, ease of administration, and, which is of particular importance, allows realizing the concept of least privilege which is a way to limit the potential risk of compromise or misuse.

In addition to the core RBAC model, roles can be organized in hierarchies defining a partial order “ $\sqsubseteq$ ”. Senior roles which are at higher levels in the hierarchy inherit all privileges that are granted to their junior roles. To give an example, the role Internist in Figure 2.2(a) is senior to Physician, denoted as Internist  $\sqsupseteq$  Physician. The other way round, Physician is called junior role of Internist.

In RBAC, two types of associations must be managed. One is the assignment of privileges to roles and the second is the assignment of roles to subjects. If  $users(r)$  represents the membership function for a role  $r$  and  $permissions(r)$  the set of privileges that are assigned to  $r$ , the following holds for two roles  $r$  and  $r'$  with  $r'$  being a junior role of  $r$ , i.e.,  $r' \sqsubseteq r$ :

- $users(r') \supseteq users(r)$  and
- $permissions(r') \subseteq permissions(r)$ .

A role  $r'$  is called *immediate descendant* of a role  $r$  if  $r' \sqsubseteq r$  and there is no  $r''$  with  $r \neq r''$  and  $r'' \neq r'$  such that  $r' \sqsubseteq r'' \sqsubseteq r$ . A role hierarchy is called *limited* if each role has at most one immediate descendant. This is the case for the example hierarchy illustrated in Figure 2.2(a). In contrast to this, *general role hierarchies* like the one shown in Figure 2.2(b) support the concept of multiple (access right) inheritance. As shown in the figure, the role Manager has two immediate descendants, namely Health Personnel and Administrative Personnel.

## 2.2.4 Administration of Authorization

The discussed access control models specify how access rights are defined and evaluated. Another important issue is the administration of access control policies, denoting who is allowed to define and modify access rights.

Regarding multilevel mandatory access control, subjects are assigned authorization levels by the security administrator. The sensitivity levels of objects are deferred based on the classification of the subjects creating the respective objects. Thus, in mandatory access control, there is typically one central authority that is responsible for administering security policies.

In contrast to this, a wide range of administration schemes exists for discretionary and role based access control. The following is an excerpt of possible administration schemes taken from [Sandhu and Samarati (1994)]:

- *Ownership*: According to the ownership paradigm, the creator of an object can grant and revoke access rights for that object.
- *Centralized*: This approach is akin to the administration of mandatory access control, meaning that one central authority is concerned with the administration of access control rules.



- *Decentralized*: The owner of an object can delegate the administration to other subjects that themselves can then grant and revoke access rights for that object.
- *Hierarchical*: The responsibility for administering policies is assigned to several administrators that are organized in a hierarchy. For example, the security officer of a company can delegate the administration to the security officers of the company's subsidiaries.
- *Cooperative*: In cooperative administration schemes, some access rights can not be granted by a single authorization entity, i.e., need to be granted by a group of authorizers, instead.

## 2.3 Access Control Requirements of Service-oriented Architectures

Service-oriented computing paradigms reveal the following characteristics which have to be taken into account when designing appropriate authorization infrastructures:

### **Autonomy of Authorization**

Web services, as the predominant technology for realizing service-oriented architectures, are fine grained, modular software components that independently enforce access control. Thus, in contrast to monolithic architectures, no single point of administration is given in service-oriented IT infrastructures.

### **Multilayered Authorization**

Service-oriented architectures can be used to integrate existing enterprise applications and legacy systems by use of standardized service interfaces. Services can on their part be combined to realize higher order services, thus, leading to composite applications. Regarding service compositions, access control is enforced in a multilayered manner.

### **Coalition-based Access Control**

Via service compositions, intra- and inter-organizational value creation chains can be realized. In order to enable inter-organizational cooperations, the authorization framework needs to support the delegation of access rights across administrative boundaries and the evaluation of authorizations within collaboration networks. Through service invocations, users like customers, suppliers, and partners are able to directly access business relevant data from outside the organization. This is what Lord (2002) refers to as "disintermediation" as requests are directly passed to the company's information system instead of being mediated by employees who supervise the execution.

### **Demand for Scalability**

The employed access control models and mechanisms need to be scalable. In particular, identity based authorization is not meaningful in dynamic coalition environments. Instead, authorizations should be inferred based on the requesters' attributes.

## **2.4 Design Principles of our Authorization Framework**

Samarati and de Capitani di Vimercati (2001) point out that the conceptual specification and realization of an access control framework is a three-phased approach consisting of the specification of the underlying security model, the (informal and formal) definition of policies, and the implementation of the policy enforcement mechanisms. Based on the overview of policy models (Section 2.2) and the previous discussion on security requirements of service-oriented architectures (Section 2.3), we now motivate the design principles of our access control framework which we present in detail in the following chapters.

### **Access Control Model**

Atluri (2001) and Lopez et al. (2004) emphasize that mandatory policies are oftentimes considered to be impractical for many scenarios other than military applications. Instead, discretionary policies are typically used to realize access control of commercial applications. Furthermore, mandatory access control can hardly be realized in distributed environments as this would require all cooperating partners to use consolidated classification schemes for subjects and objects. Otherwise, rules can become inconsistent, in case confidential and secret information is classified differently by the partners. Therefore, we employ discretionary and role based access control policies. Furthermore, the employed policy model should also support the dynamic set-up of collaboration networks. That's why we realize a distributed RBAC scheme supporting role delegations and cross-domain assignments. Thereby, access control rules can be administered in a distributed manner. Nevertheless, as trust relationships in dynamic coalitions can change, policy administration is preserved for the owners of protection objects. Thus, if necessary, interactions can be canceled at any time. Our model relies on positive authorization and the closed world assumption. This supports the reliable and clear administration of security in distributed systems, in particular dynamic coalitions.

### **Policy Language**

Policy language candidates suitable for our authorization framework must provide the following characteristics:

- They must support discretionary and role based access control schemes. Furthermore, attribute based policy specifications need to be supported.
- They need to be capable of realizing multilayered access control which demands for the specification of combined policies.

- They need to be easily integrable into existing Web service architectures.

Several good policy languages have been proposed in the past and our intention was not to reinvent the wheel by specifying yet another one. Our choice fell on XACML which is a widely adopted XML-based security standard that can be seamlessly integrated into Web service environments and fulfills the above requirements.<sup>3</sup>

### **Policy Enforcement Mechanisms**

Policy evaluation needs to be efficient and scalable as discussed in the previous section. For intra-organizational composite applications, we propose a policy consolidation technique that reduces the cost of policy enforcement. With regard to distributed Web service workflows, access control is realized as an interplay of local and distributed authorization checks. We complement this enforcement strategy with secure caching techniques making this approach applicable for large-scale dynamic service coalitions.

---

<sup>3</sup>See [Moses (2005)].



---

# Optimized Access Control for Composite Applications and Workflows

---

With the advent of service-oriented computing principles, monolithic enterprise resource planning (ERP) systems are decomposed into their basic functionalities which are then provided in the form of autonomous Web services. New business processes can be realized on demand through Web service compositions, bringing about higher levels of flexibility and adaptability. In general, the individual services or generic sub-applications autonomously enforce access control. Nevertheless, for the sake of security and efficiency, consolidated access control policies for composite applications should be provided. Such policies are based on the policies of the corresponding autonomous sub-applications and have the following properties: On the one hand, they are as restrictive as possible to block requests which do not comply with the integrated sub-applications' policies. Thus, ultimately unauthorized requests can be detected at an early stage and unnecessary service executions can be avoided. On the other hand, the combined policies must grant all privileges necessary to make the intended functionality available to legitimate users.

This chapter presents our formal model and algorithmic solutions for consolidating the access control of composite applications. The generated policies conform to the paradigm of least required privileges and, thus, allow the revision and optimization of the access control of composite applications. We demonstrate these issues by means of Web service workflows that constitute the state-of-the-art of science and technology for realizing business processes.

Parts of this chapter have already been presented at the *International Conference on Emerging Trends in Information and Communication Security* (ETRICS 2006, [Wimmer et al. (2006a)]) and at the *20<sup>th</sup> Annual IFIP WG 11.3 Working Conference on Data and Applications Security* (DBSec 2006, [Wimmer et al. (2006b)]).

This chapter is organized as follows: Section 3.1 motivates the need for policy consolidation for tightening and optimizing the access control of composite applications. In Section 3.2, we introduce our policy algebra which constitutes the basis for the consolidation approach that is

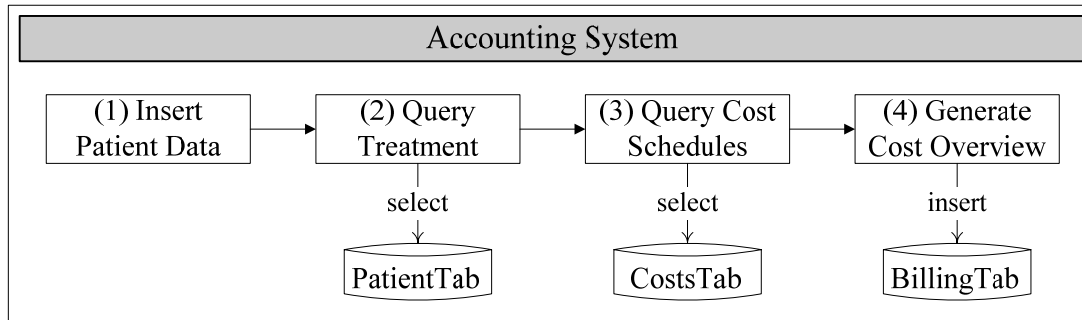


Figure 3.1: Multilayered architecture of a hospital’s accounting system

presented in Section 3.3. Algorithmic solutions for analyzing and combining access rules are discussed in Section 3.4. In Section 3.5, we describe an efficient policy enforcement technique for Web service workflows. Section 3.6 presents some related work before we summarize in Section 3.7.

### 3.1 Motivation

Composite applications rely on further sub-applications – also called sub-activities in the following – to implement their functionality. There are numerous examples including quite simple Web applications as well as large scale enterprise resource planning (ERP) systems that depend on database backends. Also, business processes that are realized as Web service workflows represent complex composite applications. In general, sub-applications themselves can constitute composite applications.

As an example of a composite application consider the hospital accounting system illustrated in Figure 3.1.<sup>1</sup> It is implemented as the sequence of four sub-activities: In order to draw up an account, first of all, the patient’s personal data like his/her name, insurance policy number, etc. has to be fed into a user interface (*Insert Patient Data*, step 1 in Figure 3.1). Afterwards, the patient’s medical record is queried and the costs of his/her therapy are determined (steps 2 and 3). After the costs for the medical treatment have been calculated, the cost overview is generated in step 4. As illustrated in the figure, the sub-activities *Query Treatment*, *Query Cost Schedules*, and *Generate Cost Overview* rely on database backends, hence representing composite applications themselves.

Suppose that Alice as an administrative employee of the hospital is responsible for balancing accounts with the health insurance funds. For being allowed to execute the accounting system, she must be granted execution rights for all of these sub-activities and requires access rights on the underlying database(s). In case of lacking some of these privileges, e.g., the permission to insert the results into the *BillingTab*, she will proceed until step 4 and be blocked then. Thus, previous steps will be executed ineffectively. Besides, ultimately unauthorized requests can even

<sup>1</sup>See Appendix A for details on the employed graphical notation.

demand for transaction rollbacks or costly compensating transactions. To avert this, one possible approach is to provide Alice with comprehensive access rights on the database. In this regard, one extreme would be to grant full access on the tables PatientTab, CostsTab, and BillingTab to her. However, the system will get vulnerable to avoidable security threats that might originate from the inside as well as from the outside of the organization: Alice intentionally or unintentionally can modify data she shall not be able to do according to her job profile. Furthermore, attackers succeeding in disclosing Alice's account acquire substantial control over corporate assets.

The solution to this problem is to grant users exactly those privileges they require to perform their work. In the literature, this concept is referred to as the *principle of least privilege* or as the *least authority paradigm* [Bishop (2002); Curtin (2001); Stiegler et al. (2006)]. Considering composite applications, fulfilling this principle is a non-trivial task in general as the access control configurations of several autonomous sub-applications have to be taken into account. The key to success is a consolidated view onto the access control of composite applications, providing answers to the following questions: (1) What are the least required privileges?, (2) Who is allowed to execute the composite application?, and (3) Are there possibilities to reduce policy evaluation costs?

Issue (1) refers to the principle of least privilege, denoting that only those privileges are granted that are required for performing the sub-activities. Following this design paradigm reduces security vulnerabilities as it guarantees that no business resources other than the ones needed by the composite application can be accessed.

Knowing the set of authorized users facilitates the detection of unintended configurations. For instance, if only highly privileged users like the hospital manager are authorized to execute the process, this might be an indication that the design of the application itself needs to be revised. We are addressing issue (2) from the *single-user / single-role* perspective, meaning that a user can execute the application by the activation of one task specific role. This complies with many business processes which are typically representing job specific functions and are thus designed for specific groups of employees. Therefore, composite applications are to be distinguished from multi-user workflows which are business processes that are executed by several users as a team.

The access control of composite applications can be optimized as follows (confer issue (3)): On the one hand, a consolidated policy allows the early-filtering of requests. Application invocations which will lead to aborts at later stages in the process due to missing privileges can be detected and averted. On the other hand, repeated and possibly redundant authorization checks by the individual sub-activities are avoided in case the authorization decision can be inferred on the composite application's layer.

Practical use cases might consider some of these issues in isolation. In Section 3.5, we propose an efficient policy enforcement strategy for Web service workflows concentrating on issues (2) and (3). Other enterprise application integration projects have their focus on issues (1) and (3). For example, the security engineering approach introduced in Chapter 4 is a methodical approach for determining least required privileges for database backed Web services.

## 3.2 Policy Model

First, we introduce the policy algebra which constitutes the basis for the formal specification of the proposed policy consolidation technique. In our model, entities like subjects are specified through characterizing attributes like role-membership, age, profession skills, etc. The policy model allows to express discretionary access control (DAC) rules and supports role based access control (RBAC) models which are suitable security concepts for almost all commercial applications. The formal syntax and semantics of our policy model are based on those introduced by Bonatti et al. (2002). We adapted and extended this model where necessary, e.g., by introducing additional operators.

### 3.2.1 Notation

#### 3.2.1.1 Predicates

Predicates represent attribute comparisons of the form (*attribute-identifier*  $\circ$  *constant*). Depending on the attribute's domain, the comparison operator  $\circ$  is in  $\{<, \leq, =, \geq, >\}$  for totally ordered sets and in  $\{\sqsubset, \sqsubseteq, =, \sqsupseteq, \sqsupset\}$  for partially ordered finite sets.

For instance, the predicate (*role* = Physician) states that the role Physician is assigned to the attribute identifier named *role*. The predicate (*role*  $\sqsupseteq$  Physician) describes the set of all possible role assignments, with *role* being equal to the role Physician or any senior role of it.

#### 3.2.1.2 Subjects, Objects, Actions, and Conditions

Let *Attr* be the set of distinguished attribute identifiers. *Attr* is subdivided into disjoint sets of subject, object, action, and environment attribute identifiers (denoted as *S-Attr*, *O-Attr*, *A-Attr*, and *E-Attr*, respectively).

A set of subjects *S* is represented by a disjunction of predicate conjunctions over *S-Attr*. That is,

$$S = ((s_{1,1} \wedge \dots \wedge s_{1,l}) \vee \dots \vee (s_{k,1} \wedge \dots \wedge s_{k,l})),$$

with each  $s_{i,d}$  ( $1 \leq i \leq k$  and  $1 \leq d \leq l$ ) being a predicate conjunction that applies to one attribute of *S-Attr*. Examples are (*age*  $\geq$  18) and (*age*  $\geq$  18  $\wedge$  *age*  $\leq$  65), representing the intervals  $[18, \infty[$  and  $[18, 65]$ , respectively. The cardinality of *S-Attr* is denoted by *l*. The elements of *S-Attr* are also called dimensions of subject specifications.

Objects *O* and actions *A* are represented in an analogue way. *S*, *O*, and *A* are inequality-free. A condition *c* is a boolean formula defined over attributes of *E-Attr* that can include user defined functions with Boolean codomain (e.g., *isWeekday*(*date*), for *date* being an element of *E-Attr*).

#### 3.2.1.3 Rules

A rule *R* is a quadruple (*S*, *O*, *A*, *c*), consisting of subjects *S*, objects *O*, and actions *A*. A rule assigns privileges specified by (*O*, *A*) to subjects *S*. The scope of the assignment is restricted



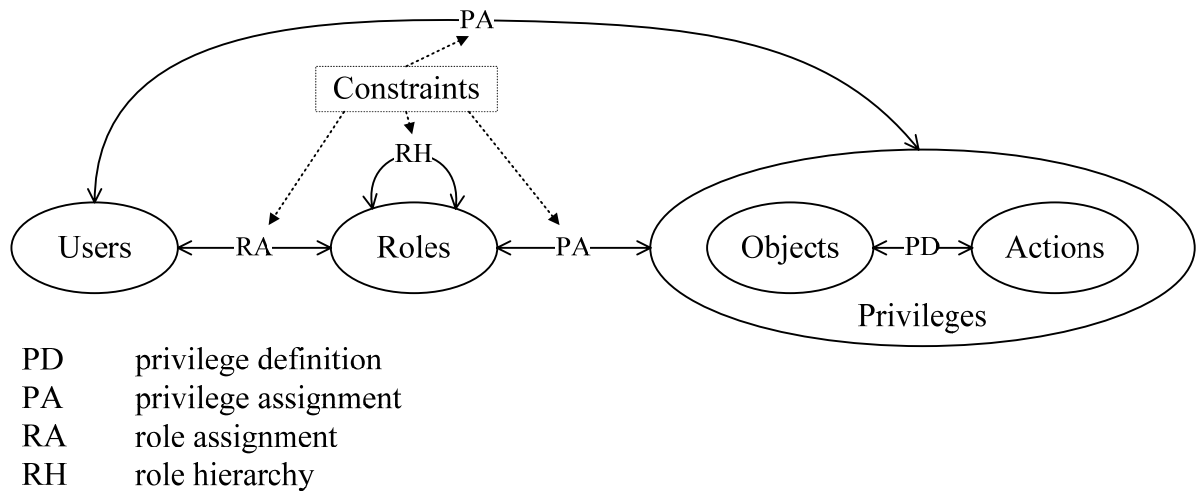


Figure 3.2: Employed policy model

through  $c$ . For example, a rule declaring that physicians with permanent employment can read and modify medical records between 8 am and 6 pm looks as follows

$$R_1 = ((role \sqsupseteq \text{Physician} \wedge \text{employment} = \text{permanent}), (table = \text{MedicalRecordsTab}), \\ (method = \text{select} \vee method = \text{update}), (time > 8 \text{ am} \wedge time < 6 \text{ pm}))$$

This simplified example already shows that the policy model supports the expression of role based access control. Figure 3.2 illustrates the different kinds of assignments that can be expressed. For any assignment, the cardinalities are *many-to-many*. For example, more than one action can be granted on one object and one action (e.g., *read*) can be granted on one or many objects (e.g., files, directories, etc.). Privileges can be assigned directly to users. Roles play two parts in our policy model. On the one hand, role attributes are part of the subject specification when being used to group privileges. On the other hand, roles can be assigned to other subjects – which then could be users (role assignment, RA) or further roles (role hierarchy, RH). Then the role identifiers will also be part of the object specification. The following rule  $R_2$  states that the role ChiefPhysician is a senior role of the role Physician:

$$R_2 = ((role = \text{ChiefPhysician}), (granted-role = \text{Physician}), (method = \text{enable}), (\text{true}))$$

That way, roles can be organized in a hierarchy. As illustrated in the figure, the applicability of the assignments PA, RA, and RH can be constrained through conditions.

### 3.2.1.4 Policies

Individual rules  $R_1, \dots, R_n$  can be aggregated in a policy  $P = \{R_1, \dots, R_n\}$ .

### 3.2.1.5 Evaluation Context

If  $D_1, \dots, D_m$  are the domains of the attributes in  $Attr$ , then  $\mathcal{E}$  is defined as  $D_1^\perp \times \dots \times D_m^\perp$ , with  $D_j^\perp = D_j \cup \{\perp\}$ . The symbol  $\perp$  stands for “unspecified”. An evaluation context  $e \in \mathcal{E}$  is a partial mapping of the attributes defined in  $Attr$ .

## 3.2.2 Semantics

### 3.2.2.1 Evaluation of Rules

An evaluation context  $e \in \mathcal{E}$  is evaluated against the individual components of rules. A subject specification  $S$  applies to  $e$  iff  $S$  maps to true w.r.t. the attribute values of  $e$ . That is,

$$\llbracket S \rrbracket_e := S(e) = (\text{true} \mid \text{false}).$$

Thereby, a predicate that is defined on attribute  $a$  will evaluate to false if  $a$  is not specified in  $e$  (i.e., equal to  $\perp$ ). The semantics of  $O$ ,  $A$  and  $c$  are defined analogously. The applicability of a rule  $R$  w.r.t.  $e$  is defined as

$$\llbracket R \rrbracket_e := \llbracket S \rrbracket_e \wedge \llbracket O \rrbracket_e \wedge \llbracket A \rrbracket_e \wedge \llbracket c \rrbracket_e.$$

As an example assume that at 1 pm Dr. Kerry Weaver, who is a chief physician, intends to access the medical records of her patients which are stored in the table `MedicalRecordsTab`. Thus, the evaluation context would look as follows:

$$e = (\text{role} = \text{ChiefPhysician}, \text{employment} = \text{permanent}, \\ \text{table} = \text{MedicalRecordsTab}, \text{method} = \text{select}, \text{time} = 1\text{pm})$$

As the role `ChiefPhysician` is a senior role of the role `Physician`, the evaluation result of  $e$  against the previously introduced role  $R_1$  is  $\llbracket R_1 \rrbracket_e = \text{true}$ .

### 3.2.2.2 Evaluation of Policies

The semantics of a policy  $P$  depends on the employed *policy evaluation algorithm* (abbreviated *pe-alg*). We define the evaluation algorithms *pe-any* and *pe-all*:

- any rule applies:  $\llbracket P \rrbracket_e^{\text{pe-any}} := \bigvee_{R \in P} \llbracket R \rrbracket_e$
- all rules apply:  $\llbracket P \rrbracket_e^{\text{pe-all}} := \bigwedge_{R \in P} \llbracket R \rrbracket_e$ .

When runtime information has to be taken into account, *pe-any* can be used to gradually perform access control. Under the assumption that for each sub-activity a unique rule is defined, access control can autonomously be performed by each sub-activity. In this case, *any* rule must be determined that authorizes the execution of the respective sub-activity. This policy enforcement strategy is illustrated in Figure 3.3(a).

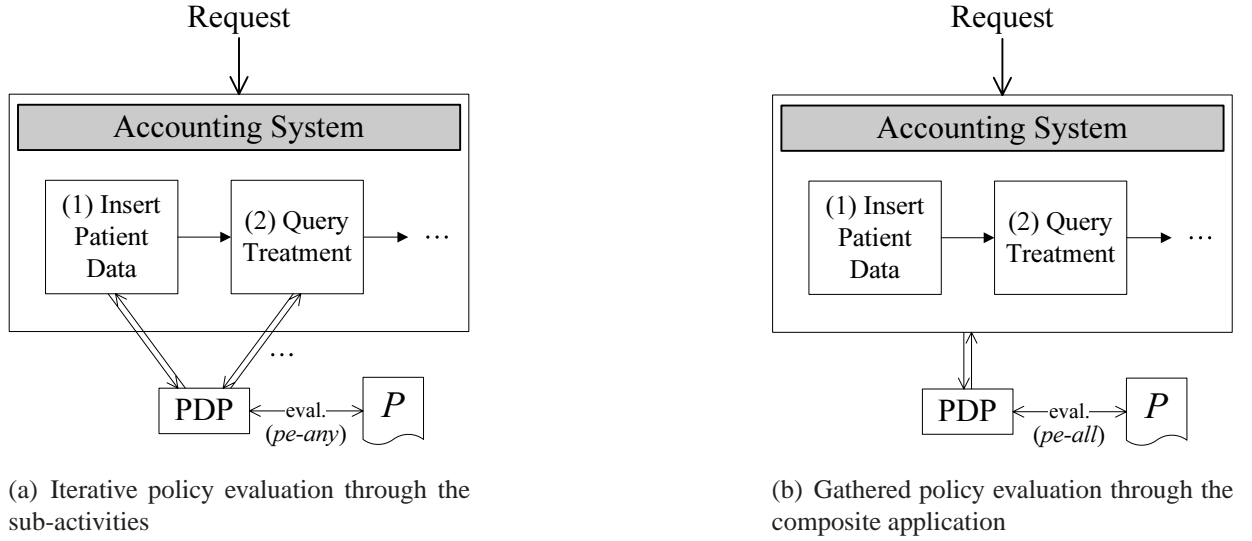


Figure 3.3: Policy enforcement strategies

In the following sections, we concentrate on centralized policy enforcements as depicted in Figure 3.3(b). In cases when access control can be performed in advance on the layer of the composite application, authorization checks for sub-activities are combined by applying *pe-all*.

In the following, we use the symbol  $\Upsilon$  to represent unrestricted policies. That means,  $\forall e \in \mathcal{E} : \llbracket \Upsilon \rrbracket_e^{pe-all} = \text{true}$ .

### 3.2.3 Policy Combining Operators

#### 3.2.3.1 Conjunction

Let  $S$  and  $S'$  be two subject specifications. The conjunction of  $S$  and  $S'$  is denoted as  $S \wedge S'$  with  $\llbracket S \wedge S' \rrbracket_e = \llbracket S \rrbracket_e \wedge \llbracket S' \rrbracket_e$ . The conjunction operator is analogously defined on objects, actions, conditions, and rules.

#### 3.2.3.2 Subtraction

The subtraction of two subject specifications  $S$  and  $S'$  is defined as  $S - S'$  with  $\llbracket S - S' \rrbracket_e = \llbracket S \rrbracket_e \wedge \neg(\llbracket S' \rrbracket_e)$ . The subtraction operator is analogously defined on objects, actions, conditions, and rules.

#### 3.2.3.3 Projection

Let  $R = (S, O, A, c)$  be a rule. The projection on the subjects of  $R$  is defined as  $\Pi_S(R) = S$ . Projections on objects, actions, conditions, and privileges are accordingly defined as

$\Pi_{\mathcal{O}}(R) = O$ ,  $\Pi_{\mathcal{A}}(R) = A$ ,  $\Pi_{\mathcal{C}}(R) = c$ , and  $\Pi_{\mathcal{O},\mathcal{A}}(R) = (O, A)$ .

Let  $P = \{R_1, \dots, R_n\}$  be a policy.  $\Pi_{\mathcal{S}}(P)$  is defined as  $\Pi_{\mathcal{S}}(P) = \{\Pi_{\mathcal{S}}(R_1), \dots, \Pi_{\mathcal{S}}(R_n)\}$ . The remaining projection operators  $\Pi_{\mathcal{O}}(P)$ ,  $\Pi_{\mathcal{A}}(P)$ ,  $\Pi_{\mathcal{C}}(P)$ , and  $\Pi_{\mathcal{O},\mathcal{A}}(P)$  are defined analogously.

We use the abbreviation  $\mathcal{S}(P) = \bigwedge_{1 \leq i \leq n} \Pi_{\mathcal{S}}(R_i)$  to denote those subjects that are granted all privileges defined in  $P$ .

### 3.2.3.4 Privilege, Rule, and Policy Relaxation

A privilege  $(O', A')$  relaxes a privilege  $(O, A)$ , denoted as  $(O, A) \sqsubseteq (O', A')$ , iff it applies to more (or the same) actions on more (or the same) objects. That is,  $(\llbracket (O, A) \rrbracket_e = \text{true})$  implies  $(\llbracket (O', A') \rrbracket_e = \text{true})$  for any evaluation context  $e$ . Accordingly, a rule  $R'$  relaxes a rule  $R$ ,  $R \sqsubseteq R'$ , iff it grants more or the same privileges to more or the same subjects under the same or less restrictive conditions. That is,  $\forall e \in \mathcal{E}$  with  $(\llbracket R \rrbracket_e = \text{true}) \Rightarrow (\llbracket R' \rrbracket_e = \text{true})$ . In the same way,  $P \sqsubseteq^{pe\text{-alg}} P'$  iff  $\forall e \in \mathcal{E} : (\llbracket P \rrbracket_e^{pe\text{-alg}} = \text{true}) \Rightarrow (\llbracket P' \rrbracket_e^{pe\text{-alg}} = \text{true})$ .

### 3.2.3.5 Reduced Policies

In order to efficiently consolidate policies of composite applications, we are focusing on reduced policies (see Section 3.3.1). Let the applied policy evaluation algorithm be *pe-all*. A policy  $P$  is called *reduced* iff

- (1)  $\forall R, R' \in P, R \neq R' : \nexists e \in \mathcal{E} : (\llbracket \Pi_{\mathcal{O},\mathcal{A}}(R) \wedge \Pi_{\mathcal{O},\mathcal{A}}(R') \rrbracket_e = \text{true})$  and
- (2)  $\forall R \in P : \mathcal{S}(P) = \Pi_{\mathcal{S}}(R)$

A policy fulfilling (2) but not (1) can be transformed into an equivalent reduced policy by eliminating overlapping rules:

Let  $R_a, R_b \in P, R_a \neq R_b : \exists e \in \mathcal{E} : (\llbracket \Pi_{\mathcal{O},\mathcal{A}}(R_a) \wedge \Pi_{\mathcal{O},\mathcal{A}}(R_b) \rrbracket_e = \text{true})$ . Substitute the two rules  $R_a, R_b$  through the three combined rules  $R_{a-b}, R_{a \wedge b}, R_{b-a}$  with

- $R_{a-b} = (\mathcal{S}(P), \Pi_{\mathcal{O},\mathcal{A}}(R_a) - \Pi_{\mathcal{O},\mathcal{A}}(R_b), \Pi_{\mathcal{C}}(R_a))$ ,
- $R_{a \wedge b} = (R_a \wedge R_b)$ , and
- $R_{b-a} = (\mathcal{S}(P), \Pi_{\mathcal{O},\mathcal{A}}(R_b) - \Pi_{\mathcal{O},\mathcal{A}}(R_a), \Pi_{\mathcal{C}}(R_b))$ .

## 3.3 Policy Consolidation

Access control policies of composite applications specify the privileges which apply to the composite applications' sub-activities. The aim of policy consolidation is to determine minimized policies that are restricted to the functionality of the composite applications.

### 3.3.1 Problem Specification

Let  $APP_1, \dots, APP_N$  be  $N \geq 1$  (autonomous) sub-activities of the composite application  $APP_0$  and  $P_i$  be the policy that applies to  $APP_i$  (for  $1 \leq i \leq N$ ). We equate the permission to execute the  $i^{\text{th}}$  sub-activity with the set of privileges needed for performing the actions of  $APP_i$ . This set is defined by  $\Pi_{\mathcal{O},\mathcal{A}}(P_i)$ . In order to enforce all of these access rights, we use *pe-all* as evaluation algorithm. We assume  $P_i$  to be a reduced policy. Thus, as defined in Section 3.2.3.5,  $P_i$  has the following two characteristics: First, the privileges defined in  $\Pi_{\mathcal{O},\mathcal{A}}(P_i)$  are disjoint. Second, the rules of  $P_i$  apply to the same set of subjects. In some cases it might be required that the privileges  $\Pi_{\mathcal{O},\mathcal{A}}(P_i)$  are granted to different groups of users under varying conditions. For instance, subjects  $S_1$  might be able to execute the application only during the day, while for subjects  $S_2$  access is restricted to the night. In such cases,  $P_i$  will not fulfill the second criterion. In order to efficiently process the set of constraints, the policy is decomposed into reduced policies which are evaluated independently. The reducedness-property allows us to give a concise definition of policy consolidation without restriction of the general case.

Let  $P_0$  be the reduced policy for  $APP_0$ . In many cases there might be no predefined policy for  $APP_0$ , meaning that  $P_0$  is equal to  $\Upsilon$ . This, for instance, is typically the case for Web service workflows as we will see in Section 3.5. Nevertheless, the application developer might as well provide a predefined policy that specifies the intended configuration. This approach can, for example, be applied regarding database backed Web services, which will be discussed in Chapter 4. The objective of policy consolidation is to evaluate  $P_0$  against the policies of the sub-applications. Its result is an optimized policy  $P^{\text{opt}}$  that fulfills the following two criteria:

#### Least privileges criterion (LP)

Each privilege defined in  $P^{\text{opt}}$  must also be defined in at least one policy  $P_i$  with  $1 \leq i \leq N$ . The privileges defined in  $P^{\text{opt}}$  must be sufficient to perform  $APP_0$  and its sub-activities.

#### Maximum set of subjects criterion (MS)

Each subject that is authorized based on the original policy configurations  $(P_i)_{0 \leq i \leq N}$  must also be authorized by  $P^{\text{opt}}$ . Each subject that is defined in  $P^{\text{opt}}$  must also be defined in at least one policy  $P_i$  with  $1 \leq i \leq N$  and in  $P_0$ .

### 3.3.2 Workflow Dependencies

The control flow of a composite application determines the execution order of its sub-activities. Figure 3.4(a) sketches the structure of a composite application. Sub-activities can be executed in sequence or in parallel.<sup>2</sup> From an access control point of view these two control patterns denote that *all* sub-activities are invoked. We represent this fact through the SEQUENCE pattern (Figure 3.4(b)). Furthermore, conditional and event based executions are possible. From the

<sup>2</sup>Iterations, i.e., loops, are discussed in Section 3.3.5.

access control perspective this denotes that only *one* sub-activity will be invoked, which we represent through the so-called SWITCH template illustrated in Figure 3.4(c). SEQUENCE and SWITCH templates can be nested to model complex workflows. Apart from these kinds of control flow dependencies further interdependencies influencing access control can exist:

- a) *Data-flow dependencies* are given if an output parameter  $x$  of a sub-activity  $APP_i$  is input to  $APP_j$  and the value of  $x$  determines the result of the evaluation of policy  $P_j$ .

As an example consider the accounting system introduced in Section 3.1. Let's assume that Alice, who is still in training, is not allowed to draw up an account if the total costs exceed 10,000 \$. As the total amount is determined in step 3 of Figure 3.1, its outcome determines Alice's authorization for the fourth sub-activity.

- b) *External dependencies* are dependencies by parameters external to the system, like time. For example,  $P_i$  and  $P_j$  might define time constraints that restrict the execution of  $APP_i$  and  $APP_j$  to disjoint time frames. That is, the conjunction of conditions defined in  $P_i$  and  $P_j$  constitute a contradiction. Nevertheless, the control-flow can be consistent due to the execution order (e.g., think of delays during long-running transactions).

In Section 3.3.3 and Section 3.3.4 we describe the consolidation of access control policies for the two patterns SEQUENCE and SWITCH. In case no interdependencies in the form of a) or b) exist, it is sufficient to perform access control solely through  $APP_0$  based on the consolidated policy  $P^{\text{opt}}$ . Otherwise, the sub-activities still need to enforce access control on their own. In any case,  $P^{\text{opt}}$  allows to revise the security configuration of  $APP_0$  by determining appropriate user and role profiles as shown in Section 3.3.6.

### 3.3.3 Analysis of SEQUENCE Patterns

For a SEQUENCE pattern to be consistent from the access control perspective, the following two conditions must be met: First, the access rights defined in  $P_0$  must include those privileges defined in the policies  $(P_i)_{1 \leq i \leq N}$ . Second, there must be at least one subject that is granted these privileges. Otherwise, the access specifications are conflicting, preventing the execution of  $APP_0$ . Formally:

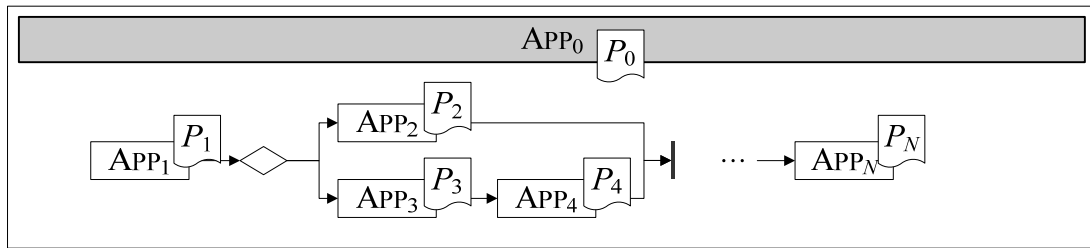
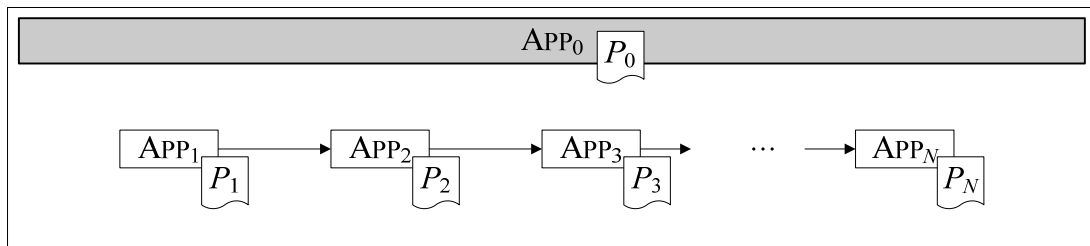
$$\forall 1 \leq i \leq N : \forall R \in P_i : \exists R' \in P_0 : \Pi_{\mathcal{O},\mathcal{A}}(R) \sqsubseteq \Pi_{\mathcal{O},\mathcal{A}}(R') \quad (3.1)$$

$$\exists e \in \mathcal{E} : \llbracket S_{\text{all}} \rrbracket_e = \text{true for } S_{\text{all}} = \bigwedge_{0 \leq i \leq N} \mathcal{S}(P_i) \quad (3.2)$$

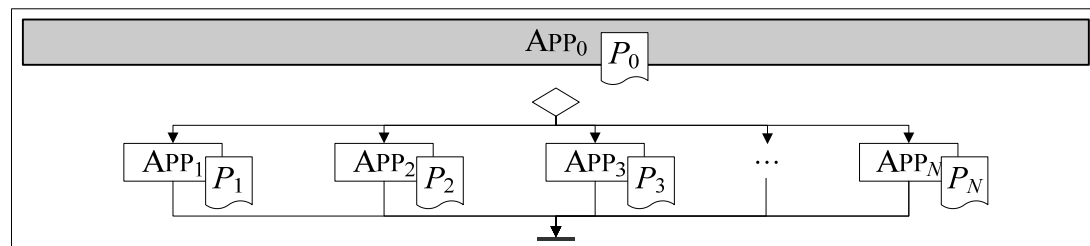
These prerequisites are schematically illustrated in Figure 3.5. The consolidated policy  $P_{(\text{all})}^{\text{opt}}$  is defined as:

$$P_{(\text{all})}^{\text{opt}} = \{(S_{\text{all}}, \Pi_{\mathcal{O},\mathcal{A}}(R), (\Pi_{\mathcal{C}}(R) \wedge \Pi_{\mathcal{C}}(R'))) \mid \forall i \in \{1, \dots, N\} : R \in P_i, R' \in P_0 : \Pi_{\mathcal{O},\mathcal{A}}(R) \sqsubseteq \Pi_{\mathcal{O},\mathcal{A}}(R')\} \quad (3.3)$$

The evaluation algorithm used for  $P_{(\text{all})}^{\text{opt}}$  is *pe-all*. If the policies  $(P_i)_{1 \leq i \leq N}$  fulfill LP, then LP can also be inferred for  $P_{(\text{all})}^{\text{opt}}$ . This is due to the privileges of  $P_{(\text{all})}^{\text{opt}}$  being restricted to those defined

(a) General design of a composite application  $APP_0$ 

(b) SEQUENCE pattern



(c) SWITCH pattern

Figure 3.4: Composite patterns

in  $(P_i)_{1 \leq i \leq N}$  and its rules being constrained through conjunctions of the respective conditions defined in these policies and  $P_0$ . Thus,  $P_{(all)}^{opt}$  will not contain privileges which do not apply to the underlying sub-activities.<sup>3</sup> Sub-activities can perform similar access on the same objects, like scans of the same tables of a database. Thus,  $P_{(all)}^{opt}$  – which aggregates the privileges defined in  $(P_i)_{1 \leq i \leq N}$  – might contain redundant access rules. Redundancies, i.e., overlapping privileges can be eliminated according to the optimization described in Section 3.2.3.5.

<sup>3</sup>Functionality directly provided by  $APP_0$  is also modeled as a sub-activity for the consolidation process.

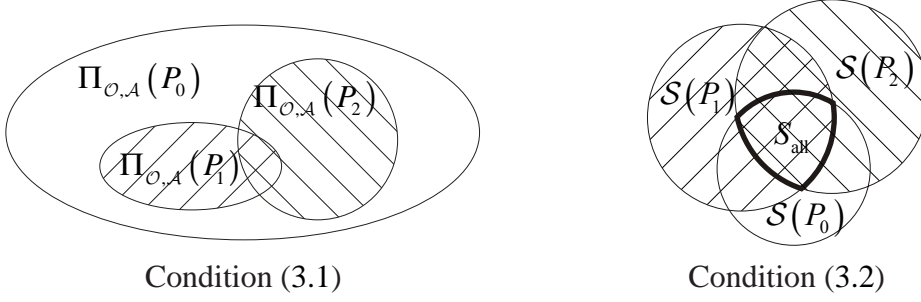


Figure 3.5: Prerequisites for the consolidation of  $P_0$  w.r.t.  $P_1$  and  $P_2$

### 3.3.4 Analysis of SWITCH Patterns

The access control configurations of SWITCH patterns can be consolidated from two different perspectives. The *full authorization* approach enforces that each subject which is defined in the consolidated policy is authorized for any of the  $(APP_i)_{1 \leq i \leq N}$ , independent of which sub-activity will actually be executed at runtime. As a consequence, the consolidated policy corresponds to  $P_{(all)}^{opt}$  defined in the previous section.

On the other hand, *partial authorization* distinguishes the different execution paths. Subjects might be authorized to execute  $APP_0$  in case a particular  $APP_i$  is invoked next, but will be blocked in any other case. Thus, in order to efficiently evaluate a SWITCH pattern, the distinguished execution branches have to be analyzed separately. Consequently, up to  $N$  security configurations have to be considered. In order to specify the optimized policy for the  $i^{\text{th}}$  branch, the policies  $P_0$  and  $P_i$  are consolidated and the following must be true:

$$\forall R \in P_i : \exists R' \in P_0 : \Pi_{O,A}(R) \sqsubseteq \Pi_{O,A}(R') \quad (3.4)$$

$$\exists e \in \mathcal{E} : \llbracket S^{(i)} \rrbracket_e = \text{true} \text{ for } S^{(i)} = S(P_0) \wedge S(P_i) \quad (3.5)$$

The consolidated policy for the  $i^{\text{th}}$  branch is defined as:

$$P_{(i)}^{opt} = \{(S^{(i)}, \Pi_{O,A}(R), (\Pi_C(R) \wedge \Pi_C(R'))) \mid R \in P_i, R' \in P_0 : \Pi_{O,A}(R) \sqsubseteq \Pi_{O,A}(R')\} \quad (3.6)$$

Again, the evaluation algorithm is *pe-all*.

Note that non-executability of an application due to conflicting and/or unsatisfiable conditions is beyond the scope of this analysis. As conditions can contain user defined functions (see Section 3.2), satisfiability is undecidable in the general case. The objectives of the proposed policy consolidation are to determine the least required privileges (LP) and the maximum set of authorized subjects (MS).

### 3.3.5 Structural Analysis

As mentioned above, from the access control perspective, control flow structures denoting the execution of all sub-activities have to be differentiated from those denoting the execution of a



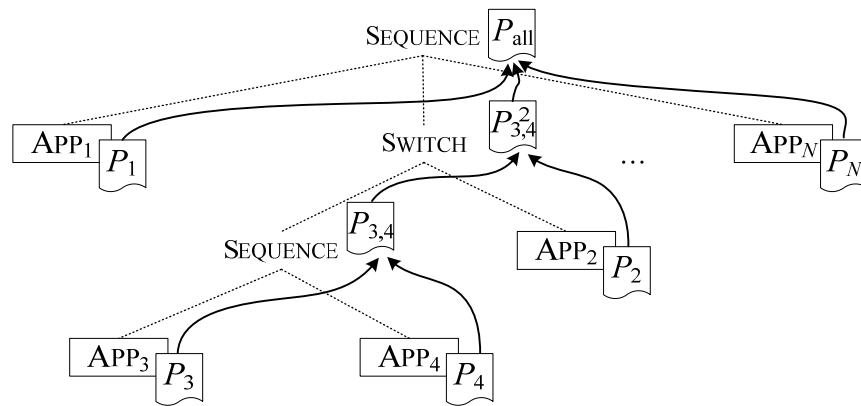


Figure 3.6: Tree representation of the composite application  $APP_0$  illustrated in Figure 3.4(a)

single sub-activity. These concepts are represented by the SEQUENCE and SWITCH patterns, respectively. SEQUENCE and SWITCH patterns can be nested. The control flow of a composite application can then be described by means of a tree whose nodes are SEQUENCE and SWITCH patterns and whose leaves represent invocations of sub-activities. Figure 3.6 shows the tree representation of the composite application illustrated in Figure 3.4(a).

Determining the consolidated policy for  $APP_0$  proceeds by means of a bottom-up analysis of the tree representation. In the given example, first the consolidated policy for the SEQUENCE node that combines  $APP_3$  and  $APP_4$  is determined by evaluating  $P_3$  and  $P_4$  as described in Section 3.3.3. The resulting policy and  $P_2$  are then input to the policy consolidation of the SWITCH node according to Section 3.3.4. Finally, the consolidated policy for  $APP_0$  is the result of the analysis of the topmost SEQUENCE node.

The complexity for performing the structural analysis depends on whether the *full authorization* or the *partial authorization* approach is employed. In case of the *full authorization* approach, the number of policy comparisons depends linearly on the number of nodes. Worst case complexity increases significantly if *partial authorizations* are determined. Considering an  $n$ -ary tree of height  $m$ , up to  $n^m$  cases have to be evaluated in the worst case. This upper bound is attained if each inner node is a SWITCH node and for each such SWITCH node the maximum number of subcases has to be considered.

When determining *partial authorization*, we also need to take into account loop nested switches, i.e., SWITCH patterns which can be executed in a loop. Figure 3.7 illustrates the basic structure of a loop nested switch. The possible execution paths can be described through the regular expression  $A(B|C)^+$ . But, regarding policy consolidation, that does not mean that an uncountable number of cases needs to be differentiated. For instance, the paths  $A \rightarrow B \rightarrow C$ ,  $A \rightarrow C \rightarrow B$ , and even  $A \rightarrow B \rightarrow C \rightarrow \dots$  are equivalent from the access control point of view. Subjects authorized to execute any of these paths need to be granted execution rights for all three sub-activities. That is, for a static analysis, execution order and reoccurrences are irrelevant. In the given example, distinguished *partial authorizations* have to be evaluated for the paths  $A \rightarrow B$ ,  $A \rightarrow C$  and  $A \rightarrow B \rightarrow C$ . In general, if a switch of  $k$  sub-activities is nested

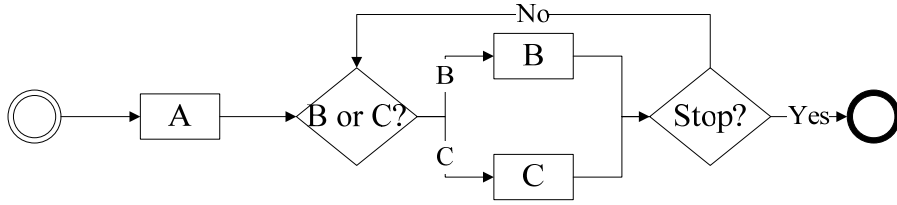


Figure 3.7: Example of a loop nested switch

within a loop, then up to

$$\sum_{i=1}^k \binom{k}{i} = 2^k - 1$$

*partial authorization* cases have to be distinguished.

The *partial authorization* approach will be of minor practical relevance if the top-level policies include an unmanageable number of case distinctions, reducing its interpretability by the software engineer. Therefore, it is reasonable to consider *partial authorization* only if the workflow size and the number of switches are limited. That means, the *partial authorization* approach will only be applied for selected scenarios. For instance, it is meaningful for business processes that consist of sub-processes which are designed for specific job profiles. For example, the financial accounting system of a company might include a sub-process for the calculation of salaries and a sub-process for the booking of goods receipt and issue. If the individual sub-processes are to be executed by different accountants, the *partial authorization* approach can be applied to determine the different job profiles.

### 3.3.6 Evaluation of the Policy Consolidation Approach

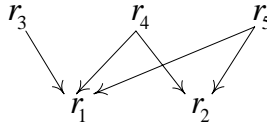
We assume that the policies  $(P_i)_{0 \leq i \leq N}$  of the sub-activities are static, meaning that they are not modified at runtime (which might be the case for some mobile application scenarios). Otherwise, in case of policy updates, the consolidation process has to be rerun. The result of the consolidation process is a policy  $P^{\text{opt}}$  that applies to  $\text{APP}_0$  and all its sub-activities (if  $P^{\text{opt}} = P_{(\text{all})}^{\text{opt}}$ ) or only to specific branches  $\text{APP}_0 \rightarrow \text{APP}_i$  (if  $P^{\text{opt}} = P_{(i)}^{\text{opt}}$ ), respectively.

#### 3.3.6.1 Static Optimization

In case external and dataflow dependencies are excluded,  $P^{\text{opt}}$  allows to adjust the access control of the composite application  $\text{APP}_0$  and to reduce policy enforcement costs. As each execution which is granted based on  $P^{\text{opt}}$  will also be granted by the sub-activities, it is sufficient to enforce access control solely at  $\text{APP}_0$ , thus, saving repeated and potentially redundant enforcements through the sub-activities. The other way round, subjects that are not authorized by the policies of the sub-activities are not authorized according to  $P^{\text{opt}}$  either. This is because (1) each subject which is defined in  $P^{\text{opt}}$  is also defined in the respective sub-activities' policies and (2) each

privilege granted to these subjects based on the sub-activities' policies is also granted based on  $P^{\text{opt}}$ . These claims are justified through the definition of policy  $P_{(\text{all})}^{\text{opt}}$  (equation (3.3)) and policy  $P_{(i)}^{\text{opt}}$  (equation (3.6)), respectively. In particular, (1) is justified through conditions (3.1) and (3.4), while (2) is true due to conditions (3.2) and (3.5). The static analysis allows to receive a consolidated view onto the set of authorized users (MS) and the least required privileges (LP), enabling the following optimizations:

**Evaluation of MS:**  $\mathcal{S}(P^{\text{opt}})$  specifies those subjects that are authorized to execute the workflow (branch) or general composite application, respectively. The aggregated information  $\mathcal{S}(P^{\text{opt}})$  allows application developers to check whether the policy complies with the intended security specifications, which is less time-consuming than an exhaustive evaluation of the sub-activities' policies. They can detect over-privileged users or conflicts if the conditions defined in Section 3.3.3 and Section 3.3.4 are not fulfilled. Furthermore, it allows to infer *least required roles* if role based access control is employed. In this regard, a least required role is a minimal role that grants process execution without demanding for further intermediary role activations. This “one role will do”-approach is especially relevant for business processes that are typically devised for specific job functions. Least required roles are unique for limited role hierarchies but not necessarily for general role hierarchies. As an example consider the following role hierarchy:



The infima of the role hierarchy are  $r_1$  and  $r_2$ . If users need to be members of  $r_1$  and  $r_2$  in order to be allowed to execute the composite application, least required roles are the least common senior roles, i.e.,  $r_4$  and  $r_5$  in the given example.

**Evaluation of LP:**  $\Pi_{\mathcal{O},\mathcal{A}}(P^{\text{opt}})$  represents the set of privileges needed for executing the composite application and corresponding sub-activities. In the meaning of a reverse security engineering, this information allows to define *task specific roles*. They are called task specific as they group exactly those privileges that are required for the composite application's functionality – which is not necessarily the case for more generic least required roles. Least required roles are roles that are already defined in a role hierarchy, while task specific roles are newly defined roles that are tailored to the security requirements of the composite application.

### 3.3.6.2 Filtering Requests

In case dataflow and temporal dependencies have to be taken into account, access control has to be performed by the autonomous sub-activities. For instance, reconsider the use case discussed in Section 3.3.2: Only after step 3 has been executed, the respective amount due is known so that Alice's authorization for step 4 can be determined. Thus, Alice cannot be authorized for

all sub-activities of the accounting system in advance. Though security checks by the individual sub-activities cannot be saved completely, consolidated policies still provide optimization capabilities: As we know that only subjects in  $\mathcal{S}(P^{\text{opt}})$  are candidates for the execution of the workflow, other requesters can be blocked immediately and unnecessary executions be avoided.

## 3.4 Algorithmic Solutions

For an implementation of the described policy consolidation technique, algorithmic solutions for the evaluation of predicate conjunctions and subtractions and for the validation of privilege relaxation are required.

### 3.4.1 Implementing the Conjunction Operator

Equations (3.2) and (3.5) introduce  $S_{\text{all}}$  and  $S^{(i)}$  as conjunctions of subject specifications. The conjunction operator is semantically equivalent to the set theoretical intersection operator. That is,  $S_{\text{all}}$  and  $S^{(i)}$  can be interpreted as the intersection of subject sets. Let  $S$  and  $S'$  be two subject specifications. According to the policy model,  $S$  and  $S'$  are represented via disjunctions of predicate conjunctions over attributes in  $S\text{-Attr}$ :

$$\begin{aligned} S &= s_1 \vee \dots \vee s_k = (s_{1,1} \wedge \dots \wedge s_{1,l}) \vee \dots \vee (s_{k,1} \wedge \dots \wedge s_{k,l}) \quad \text{and} \\ S' &= s'_1 \vee \dots \vee s'_{k'} = (s'_{1,1} \wedge \dots \wedge s'_{1,l}) \vee \dots \vee (s'_{k',1} \wedge \dots \wedge s'_{k',l}) \end{aligned}$$

The attributes in  $S\text{-Attr}$  are also called the dimensions. We assume all dimensions in  $S$  and  $S'$  to be specified. If a conjunction  $s_i$  is not constrained in dimension  $d$ , then the respective predicate  $s_{i,d}$  represents the whole domain of  $d$ . According to Section 3.2.3, the intersection of  $S$  and  $S'$  is  $S \wedge S' = \bigvee_{1 \leq i \leq k, 1 \leq j \leq k'} (\bigwedge_{1 \leq d \leq l} (s_{i,d} \wedge s'_{j,d}))$ . Nevertheless, conjunctions  $(s_{i,d} \wedge s'_{j,d})$  can be contradictory, i.e., unsatisfiable by any evaluation context. Such terms constitute unnecessary parts of a policy and shall be omitted to keep policy specifications clear. Figure 3.8 presents an pseudo-code implementation for computing a condensed representation of  $S \wedge S'$ . Algorithm `intersect` is of polynomial time complexity w.r.t. the number of conjunctive subterms and the number of dimensions  $l$ . We illustrate the algorithm by means of an example. Consider the following two subject descriptions (based on the example role hierarchy shown in Figure 2.2(a) on page 11):

$$\begin{aligned} S &= (s_1) = (\text{role} \sqsupseteq \text{Nurse} \wedge \text{yop} \geq 1) \quad \text{and} \\ S' &= (s'_1 \vee s'_2) = (\text{role} \sqsupseteq \text{Admin. Pers.} \wedge \text{yop} \geq 0) \vee \\ &\quad (\text{role} \sqsupseteq \text{Health Pers.} \wedge \text{yop} \geq 2 \wedge \text{yop} \leq 4) \end{aligned}$$

$S$  represents all subjects that are granted the Nurse role and that have at least one year of practice (abbrev. *yop*).  $S'$  represents administrative employees and all subjects that are granted senior roles of the Health Personnel role with at least two and at most four years of practice. Thus, the dimensions are *role* and *yop*. While the domain of *role* is a finite lattice (defined by the role hierarchy), the domain of *yop* is  $[0, +\infty[$ , i.e., an interval.

**Input** : Subject specifications  $S, S'$  in disjunctive normal form:

$$S = s_1 \vee \dots \vee s_k, \text{ and } S' = s'_1 \vee \dots \vee s'_{k'}$$

**Output**: The reduced representation of

$$S \wedge S' = \bigvee_{1 \leq i \leq k, 1 \leq j \leq k'} \left( \bigwedge_{1 \leq d \leq l} (s_{i,d} \wedge s'_{j,d}) \right)$$

```

1  $\Psi = \text{false};$ 
2 foreach conjunction  $s_i$  of  $S$  do
3   foreach conjunction  $s'_j$  of  $S'$  do
4     foreach dimension  $d = 1 \dots l$  do
5        $\psi_d = \text{reduce}(s_{i,d} \wedge s'_{j,d});$ 
6     endfch
7      $\Psi = \Psi \vee (\psi_1 \wedge \dots \wedge \psi_l);$ 
8   endfch
9 endfch
10 return  $\Psi;$ 

```

Figure 3.8: Algorithm intersect

As  $s_{1,\text{role}} \equiv \{\text{Nurse, Head Nurse}\}$  and  $s'_{1,\text{role}} \equiv \{\text{Admin. Pers.}\}$ ,  $s_1$  and  $s'_1$  are disjoint in the *role*-dimension. That is, the conjunction  $(s_{1,\text{role}} \wedge s'_{1,\text{role}})$  is a contradiction and the overlap in the *yop* dimension is ineffectual as the conjunctive add-on in line 7 of the algorithm in Figure 3.8 evaluates to false and can be omitted.

In contrast to this,  $s_1$  and  $s'_2$  overlap in each dimension as illustrated in Figure 3.9. The conjunction  $(yop \geq 1) \wedge (yop \geq 2 \wedge yop \leq 4)$  is reduced to  $(yop \geq 2 \wedge yop \leq 4)$ . The predicates  $s_{1,\text{role}}$  and  $s'_{2,\text{role}}$  define the two finite sets

$$\begin{aligned} \Phi_1 &= \{\text{Nurse, Head Nurse}\} \quad \text{and} \\ \Phi'_2 &= \{\text{Nurse, Head Nurse, Physician, Internist, Surgeon}\}. \end{aligned}$$

Thus,  $(s_{1,\text{role}} \wedge s'_{2,\text{role}})$  is equivalent to  $\Phi_1 \cap \Phi'_2 = \{\text{Nurse, Head Nurse}\}$ . The intersection can be represented through the predicate  $(\text{role} \sqsupseteq \text{Nurse})$ , as the role Nurse is the infimum of  $\Phi_1 \cap \Phi'_2$  according to the example role hierarchy. Thus,  $S_1 \wedge S_2 = (\text{role} \sqsupseteq \text{Nurse} \wedge yop \geq 2 \wedge yop \leq 4)$ , i.e., the intersection consists of those subjects that are granted the Nurse role and that have at least two and at most four years of practice.

### 3.4.2 Checking Privilege Relaxation

Let  $(O, A)$  and  $(O', A')$  be two privileges. As objects and actions are defined on disjoint sets of attribute identifiers (*O-Attr* and *A-Attr*, see Section 3.2.1.2) and according to the definition of privilege relaxation (Section 3.2.3.4),  $(O', A')$  relaxes  $(O, A)$  if the following holds:

$$\forall e \in \mathcal{E} : ([O]_e = \text{true} \wedge [A]_e = \text{true}) \Rightarrow ([O']_e = \text{true} \wedge [A']_e = \text{true}).$$

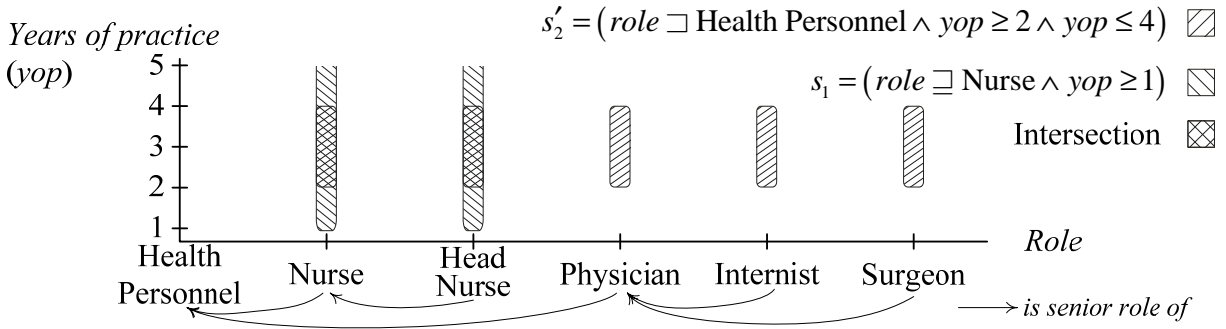


Figure 3.9: Matching conjunctive terms

Therefore, the privilege relaxation problem can be reduced to the implication problem:

“Let  $T = (t_1 \vee \dots \vee t_k)$  and  $T' = (t'_1 \vee \dots \vee t'_k)$  be disjunctions of predicate conjunctions.  $T$  implies  $T'$ , denoted as  $T \Rightarrow T'$ , if and only if every evaluation context which is satisfying  $T$  is also satisfying  $T'$ .”

Guo, Sun, and Weiss (1996)

Informally,  $T \Rightarrow T'$  means that  $T'$  is more generic than  $T$ . To evaluate whether  $T \Rightarrow T'$  holds, each predicate conjunction  $t_i$  of  $T$  is compared to the predicate conjunctions  $t'_j$  of  $T'$ . The following three cases can occur:

1.  $t_i$  implies  $t'_j$ , i.e.,  $t_i \Rightarrow t'_j$ . Then a match for  $t_i$  has been found.
2.  $t_i$  and  $t'_j$  are incomparable, meaning that  $(t_i \wedge \neg t'_j) = t_i$ . Then  $t_i$  has to be compared with the remaining predicate conjunctions of  $T'$  to find possible matches.
3.  $t_i$  and  $t'_j$  describe partially overlapping data sets: The overlap is  $(t_i \wedge t'_j)$ . The remainder  $(t_i \wedge \neg t'_j)$  is separately compared with the predicate conjunctions of  $T'$ .

Figure 3.10 shows a pseudo-code implementation of implies for evaluating predicate implications.  $T$  implies  $T'$  if all predicate conjunctions  $t_i$  of  $T$  are subsumed by  $T'$ . In this case, the remainder  $\Delta$  is equal to false. In line 6, the sub-procedure subtract is invoked which calculates the remainder of  $t_i$  w.r.t.  $t'_1$ , i.e.,  $\delta = (t_i \wedge \neg t'_1)$  given in disjunctive normal form (DNF). The individual predicate conjunctions of  $\delta$  are compared separately to the remaining conjunctions of  $T'$  through recursive invocations of implies in line 8 of the algorithm in Figure 3.10.

A pseudo-code implementation of subtract is depicted in Figure 3.11. Computing the predicate subtraction is done in a way similar to intersect by comparing the conjunctive terms  $t_i$  and  $t'_j$  in each dimension  $d$  (line 2–11). If  $t_i$  and  $t'_j$  do not overlap in any dimension  $d$ ,  $t_i$  and  $t'_j$  represent disjoint data sets and the remainder is  $t_i$ . The overall overlap of  $t_i$  and  $t'_j$  is iteratively constructed by the helping variable *work*. The non-matching parts of  $t_i$  are aggregated in  $\delta$ .

**Input** :  $T, T'$  in disjunctive normal form:  
 $T = t_1 \vee \dots \vee t_k$  and  $T' = t'_1 \vee \dots \vee t'_{k'}$   
**Output**:  $T - T'$

```

1 if  $k' = 0$  then
2   return  $T$ ;
3 endif
4  $\Delta = \text{false}$ ;
5 foreach conjunctive term  $t_i$  of  $T = (t_1 \vee \dots \vee t_k)$  do
6    $\delta = \text{subtract}(t_i, t'_1)$ ;
7   if  $\delta \neq \text{false}$  then
8      $\Delta = \Delta \vee \text{implies}(\delta, t'_2 \vee \dots \vee t'_{k'})$ ;
9   endif
10 endfch
11 return  $\Delta$ ;

```

Figure 3.10: Algorithm implies

**Input** : Predicate conjunctions  $t$  and  $t'$ :  
 $t_i = t_{i,1} \wedge \dots \wedge t_{i,l}$  and  $t'_j = t'_{j,1} \wedge \dots \wedge t'_{j,l}$   
**Output** :  $t - t'$   
**Remark**: We use the variable *work* to represent the stepwise computation of the intersection of  $t$  and  $t'$ .

```

1  $\delta = \text{false}$ ,  $\text{work} = t_i$ ; //  $\text{work} = w_1 \wedge \dots \wedge w_l$ 
2 for  $d = 1 \dots l$  do
3    $w'_d = (t_{i,d} \wedge t'_{j,d})$ ; // the overlap of  $t_{i,d}$  and  $t'_{j,d}$ 
4    $\text{work} = (w'_1 \wedge \dots \wedge w'_{d-1} \wedge w'_d \wedge w_{d+1} \dots \wedge w_l)$ ;
5   if  $w'_d = \text{false}$  then
6     return  $t_i$ ; //  $t_i$  and  $t'_j$  represent disjoint data sets
7   else if  $w'_d \neq t_{i,d}$  then
8      $\omega = t_{i,d} \wedge \neg t'_{j,d}$ ; // the remainder of  $t_{i,d}$  minus  $t'_{j,d}$ 
9      $\delta = \delta \vee (w'_1 \wedge \dots \wedge w'_{d-1} \wedge \omega \wedge w_{d+1} \dots \wedge w_l)$ 
10  endif
11 endfor
12 return DNF of  $\delta$ ; //  $\omega$  in line 8 is a predicate disjunction

```

Figure 3.11: Algorithm subtract

As an example assume a relational database with the table *Employees* which has the attributes Name, Gender, Salary, and Job (abbreviated *na*, *ge*, *sa*, and *jo*). Possible job values are the categories health personnel, administrative personnel, and technical personnel (for short *HP*, *AP*, and *TP*). Two privileges are defined on this table: The first one states that the complete table can

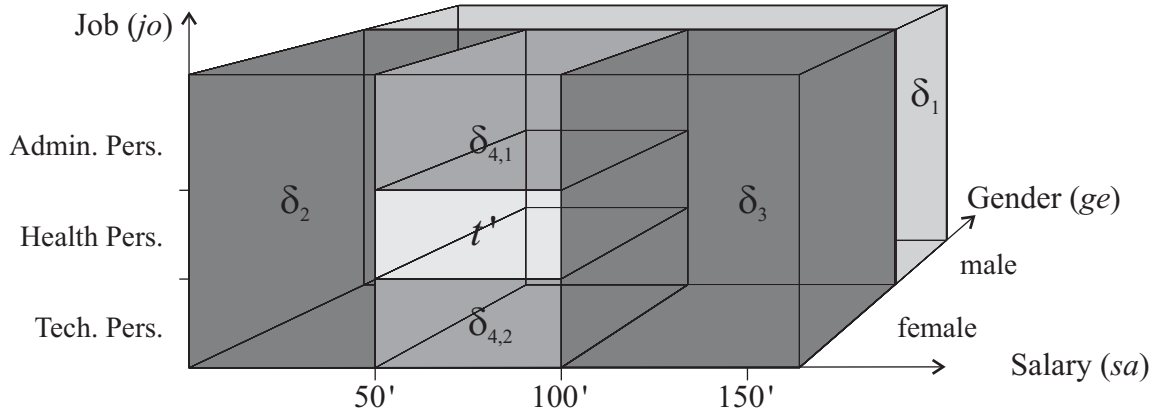


Figure 3.12: Visualization of predicate subtraction

be accessed via the select-operator. The second privilege restricts selections exclusively to the data of female health care employees that earn more than 50' \$ and less than 100' \$. We use the symbol  $\perp$  to represent unrestricted attribute values. The object specifications of both privileges are represented by the following two predicate conjunctions:

$$t = (na = \perp \wedge ge = \perp \wedge sa = \perp \wedge jo = \perp)$$

$$t' = (na = \perp \wedge ge = \text{female} \wedge sa > 50' \wedge sa < 100' \wedge jo = \text{HP})$$

Obviously,  $t$  is more generic than  $t'$ , i.e.,  $t$  relaxes  $t'$ , but not the other way round. Whether  $t'$  relaxes  $t$  can be evaluated by subtracting  $t'$  from  $t$ , i.e., by calling  $\text{subtract}(t, t')$ , and testing whether the remainder is empty. The following table shows the evaluation steps if the attributes are evaluated in the order Name, Gender, Salary, and Job:

Eval. attr.	Variable <i>work</i>	Remainders
<i>na</i>	$(na = \perp \wedge ge = \perp \wedge sa = \perp \wedge jo = \perp)$	—
<i>ge</i>	$(na = \perp \wedge ge = \text{female} \wedge sa = \perp \wedge jo = \perp)$	$\delta_1 = (na = \perp \wedge ge = \text{male} \wedge sa = \perp \wedge jo = \perp)$
<i>sa</i>	$(na = \perp \wedge ge = \text{female} \wedge sa > 50' \wedge sa < 100' \wedge jo = \perp)$	$\delta_2 = (na = \perp \wedge ge = \text{female} \wedge sa \leq 50' \wedge jo = \perp)$ $\delta_3 = (na = \perp \wedge ge = \text{female} \wedge sa \geq 100' \wedge jo = \perp)$
<i>jo</i>	$(na = \perp \wedge ge = \text{female} \wedge sa > 50' \wedge sa < 100' \wedge jo = \text{HP})$	$\delta_4 = (na = \perp \wedge ge = \text{female} \wedge sa > 50' \wedge sa < 100' \wedge jo \in \{\text{TP}, \text{AP}\})$

Figure 3.12 illustrates the comparison of  $t$  and  $t'$  in the dimensions Gender, Salary, and Job. It shows that in the Salary-dimension,  $t$  divides *work* into 3 components – the overlapping part and two remainder predicates  $\delta_2$  and  $\delta_3$ . This is the maximum number of remainder predicates that can be generated in one step if the attribute's domain is a totally ordered set (the domain of Salary is  $[0, +\infty[$ ). Things are different if the attribute's domain is a partially ordered finite set, as is the case for the dimension Job. In the figure, it is shown that  $\delta_{4,1}$  and  $\delta_{4,2}$  are the remainder



predicates of the comparison in the Job-dimension. Instead of enumerating all attribute values ( $AP$  and  $TP$ ) in distinct predicates, the remainder is internally represented as an aggregate of the form ( $jo \in \{AP, TP\}$ ) as illustrated in the table above.

Policy consolidation is performed at the time a composite application is developed or when the underlying sub-activities' policies are modified. Although policy consolidation is not considered to be a mission critical task, its usability nevertheless depends on the used algorithms' complexity. Unfortunately, the worst case time complexity of privilege relaxation is exponential with regard to the input parameter  $k'$ . The described privilege implication problem is closely related to other well known computationally hard issues like query subsumption or the satisfiability problem which, for example, have been analyzed by Rosenkrantz and Hunt (1980), Sun et al. (1989), and Guo et al. (1996). As we have to determine exact results, heuristics cannot be applied. An estimation of the complexity of implies is given in Appendix B. The worst case complexity results from the recursion in line 8 of algorithm implies and depends on the number of subterms that are generated by the predicate subtraction (see line 6 of the algorithm in Figure 3.10). In our case,  $T$  and  $T'$  stand for objects or actions of two privilege specifications. For the worst case to occur, each  $t_i$  needs to be split into the maximum number of remainder predicates during each comparison with any  $t'_j$  (estimated by the upper bound  $2l$ ). The same applies to each of the produced remainder predicates. However, the worst case is unlikely to arrive in practice, since it would demand for highly unstructured and almost unmanageable policies.

### 3.4.3 Implementing the Subtraction Operator

The semantics of the subtraction of two terms  $T$  and  $T'$  is  $\llbracket T - T' \rrbracket_e = \llbracket T \rrbracket_e \wedge \neg(\llbracket T' \rrbracket_e)$ . Thus, the subtraction operator can be realized through the already presented algorithm implies (Figure 3.10), as the remainder  $\Delta$  of  $\text{implies}(T, T')$  is equivalent to  $T - T'$ .

## 3.5 Optimizing the Access Control of Intra-organizational Web Service Workflows

So far, policy consolidation has been presented at an abstract level by means of a generic formal model. In this section, we show how it can be applied to analyze and optimize the access control of Web service workflows.

### 3.5.1 Running Example

Figure 3.13 illustrates a simplified e-health workflow which we assume to be executed when a patient is transferred to the cardiology department of a hospital.<sup>4</sup> Depending on the diagnostic

<sup>4</sup>The example workflow and the later modeled policies are fictitious and do not represent a real-world e-health business process configuration. We use this simplified workflow to demonstrate the basic principles of policy consolidation for general Web service workflows. Details about the security requirements of e-health applications are,

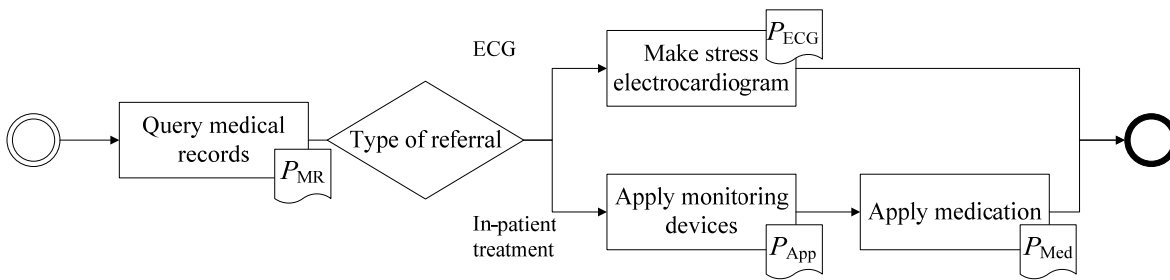


Figure 3.13: Example of an e-health (Web service) workflow

findings, either an in-patient treatment is applied or an electrocardiogram (ECG) is made in order to acquire further insight. Sub-activities of the workflow on the one hand represent practical activities that require human interaction like a medication. On the other hand, they stand for information processing tasks like an update of the stock of pharmaceuticals in the database. In the following, we concentrate on the technical aspects of the workflow and assume the subsequent informal security policies:

- The policy  $P_{MR}$  applies to the sub-activity *Query medical records*. It states:  
Health personnel with permanent employment and administrative employees are allowed to access the medical records of patients. These are stored in the table `MedicalRecordsTab` of the hospital's database.
- The policy  $P_{ECG}$  that applies to *Make stress electrocardiogram* declares:  
Nurses of the cardiology and internists are allowed to update medical records, e.g., by inserting ECG results.
- Policy  $P_{App}$  applies to *Apply monitoring devices* and is defined as:  
Internists are allowed to apply monitoring devices by marking them as in use in the table `DevicesTab`.
- Policy  $P_{Med}$  applies to *Apply medications* ( $P_{Med}$ ) and is defined as:  
Nurses and physicians can apply medications if the patient is not allergic concerning the respective pharmaceutical. This Web service updates the table `PharmaceuticalsTab`.

Access control is usually performed at the services' layer, because services are self-contained software modules that autonomously enforce security. Unfortunately, this authorization autonomy can bring about performance drawbacks: Firstly, security checks are performed redundantly. That is, authentication and authorization of the same client will be done repeatedly, which might be very costly considering, for example, certificate evaluation and verification. Secondly, further

---

for example, provided by Evered and Bögeholz (2004) and Blobel and Pommerening (1997). Neuhaus et al. (2006) and Caumanns et al. (2006) describe the telematics infrastructure of the German e-health card, which is realized as a service-oriented architecture.

performance drawbacks can emerge if services are needlessly invoked by ultimately unauthorized subjects. For instance, querying the medical records of a patient will be done unnecessarily if the workflow is called by an administration employee that is neither able to pursue the *ECG*- nor the *In-patient-treatment*-branch of the workflow. Regarding Web service transactions, even rollbacks or compensating transactions might be required.

Therefore, it is beneficial to determine the set of authorized users before executing the workflow, blocking unauthorized requests as soon as possible. Hence, this optimization is based on evaluating the MS property of consolidated policies. Considering our example, the subjects that are authorized for the workflow are those that are in the intersection of the subject sets specified in the policies  $P_{MR}$ ,  $P_{ECG}$ ,  $P_{App}$ , and  $P_{Med}$ . To give an example, subjects allowed to execute the *In-patient-treatment*-branch need to be granted privileges for the services *Query medical records*, *Apply monitoring devices*, and *Apply medication*. Consequently, these subjects are in the intersection  $\mathcal{S}(P_{MR}) \cap \mathcal{S}(P_{App}) \cap \mathcal{S}(P_{Med})$ . With regard to our informal policy specification, this applies to internists. As internists are also allowed to *Make stress electrocardiograms*, they are granted the privileges to invoke any branch within the workflow, i.e, they possess *full authorization*.

In contrast to this, nurses are only granted *partial authorization*, as they are only allowed to execute the *ECG*-branch of the workflow. Other subjects, like administrative employees, – though being able to invoke *Query medical records* – do not possess the required privileges to execute a complete branch and can be blocked right from the beginning.

### 3.5.1.1 Workflow Model

As shown by Weerawarana et al. (2005), Web service workflows can be described by use of the *Business Process Execution Language for Web Services*, BPEL4WS or simply BPEL for short. This specification introduces five patterns to model the control flow of business processes, namely *flow*, *while*, *sequence*, *switch*, and *pick*. We refer to [Andrews et al. (2003)] for their specifications. It is relevant to the consolidation of access control policies whether all or only some of the sub-activities grouped by these control patterns will be executed. As the first three control patterns require all sub-activities to be invoked,<sup>5</sup> we group them together and represent them through the SEQUENCE template, which we have introduced in Section 3.3.2. The remaining patterns specify that at most one sub-activity will be called, e.g., depending on certain conditions or the arrival of events. Thus, they can be represented by the SWITCH template.

Starting from the BPEL4WS specification, a Web service choreography can be represented through a *workflow tree* as illustrated in Figure 3.14. As shown in the figure, the workflow tree is closely related to the tree view of the XML representation. Inner nodes of the tree either represent SEQUENCE or SWITCH nodes, while leaves represent individual Web services. In order to obtain a consolidated view onto the access control settings of a workflow, we perform a bottom-up structural analysis over the workflow tree, gathering access control information.

<sup>5</sup>We assume that sub-activities of *while*-activities will be executed at least once. Furthermore, conditional execution steps that are nested in *while*-constructs need a different treatment regarding *partial authorization* as discussed in Section 3.3.5.

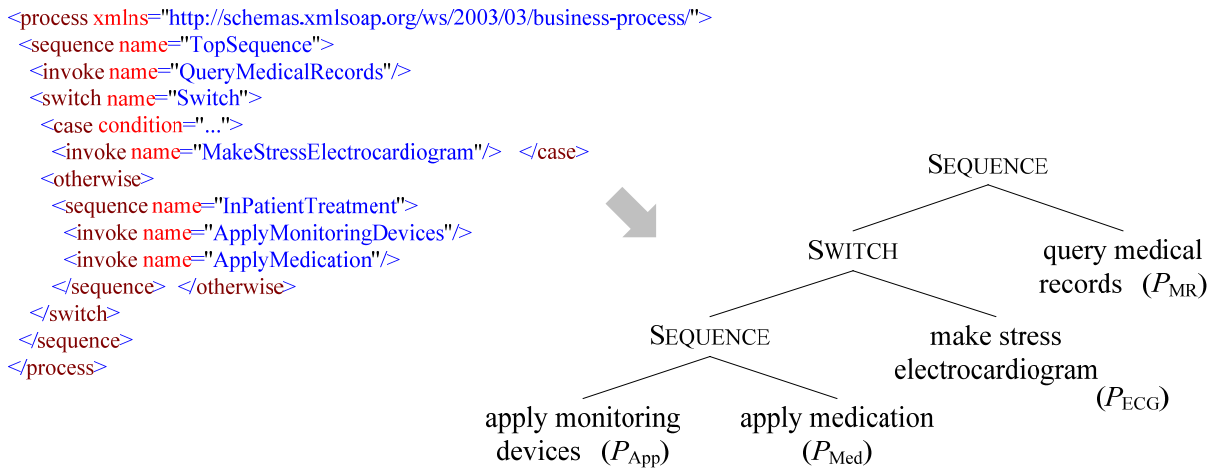


Figure 3.14: BPEL4WS-extract and workflow tree representation of the e-health process

### 3.5.1.2 Preliminary Remarks

Web services are autonomous software components that can be supplied by varying service providers. Thus, access control is administered by different authorities, in general. Moreover, services need not be exclusively used, i.e., can be integral parts of different workflows. In order to be able to consolidate access control policies, the following conditions must be met:

- (i) The access control specifications of the Web services fulfill the principle of least privilege.
- (ii) Policies can be expressed in one common policy language and rely on DAC and RBAC models.
- (iii) The subject specifications are based on the same description language.

If the first condition holds, we can infer compliance with the least privilege principle for the consolidated policies. The principle of least privilege criterion can often be attained (semi-) automatically, as we will show in the next chapter by means of database backed Web services. The second and third assumption refer to homogeneous policy representations. The proposed consolidation approach requires policies to rely on the same access control model. This precondition is typically met for intra-organizational business processes that only include local services.

Wiehler (2004) points out that enterprises will adopt Web services step by step. First, Web services will be used for intra-organizational enterprise application integration (EAI) tasks, e.g., in order to efficiently integrate legacy systems into business processes. Inter-organizational Web service workflows will at first be realized with a few selected and well-known partners until this new technology has proven to be reliable and secure. Later on, partner communities and dynamic collaborations will come along. The proposed policy consolidation approach is intended for composite applications that belong to the first phase.

## 3.5.2 Performing Policy Consolidation

### 3.5.2.1 Policy Representation

We show by means of the introduced e-health workflow, how policy consolidation for Web service choreographies proceeds. Based on the role hierarchy shown in Figure 2.2(a) on page 11 the informal access rules introduced above are represented as follows:

$$\begin{aligned}
 P_{MR} &= \{(((role \sqsupseteq \text{Health Personnel} \wedge employment = \text{permanent}) \vee \\
 &\quad (role \sqsupseteq \text{Administrative Personnel})), \\
 &\quad (table = \text{MedicalRecordsTab}), (action = \text{select}), (\text{true}))\} \\
 P_{ECG} &= \{(((role \sqsupseteq \text{Nurse} \wedge field\ of\ activity = \text{cardiology}) \vee role \sqsupseteq \text{Internist}), \\
 &\quad (table = \text{MedicalRecordsTab}), (action = \text{select} \vee action = \text{update}), (\text{true}))\} \\
 P_{App} &= \{((role \sqsupseteq \text{Internist}), (table = \text{DevicesTab}), \\
 &\quad (action = \text{select} \vee action = \text{update}), (\text{true}))\} \\
 P_{Med} &= \{(((role \sqsupseteq \text{Nurse} \vee role \sqsupseteq \text{Physician}), (table = \text{PharmaceuticalsTab}), \\
 &\quad (action = \text{select} \vee action = \text{update}), \\
 &\quad (\text{HighAnaphylaxisRisk}(\text{patient}, \text{drug}) = \text{false}))\}
 \end{aligned}$$

As each of these policies include only one rule, it can easily be shown that they are reduced policies according to the definition given in Section 3.2.3.5.

### 3.5.2.2 Structural Analysis

BPEL provides no integrated mechanism to specify security policies for business processes. Therefore, policies of inner nodes in the workflow tree are equal to  $\Upsilon$ . Hence, no relaxation tests are necessary for generating consolidated policies. Instead, privileges defined in the policies of the Web services are gathered. Therefore, we focus on determining the set of authorized users. Performing the policy consolidation for our running example starts with analyzing the policies  $P_{App}$  and  $P_{Med}$  that apply to the activities *Apply monitoring devices* and *Apply medication* which are linked in sequence as illustrated in Figure 3.13 and Figure 3.14. The subjects allowed to execute both are those granted the Internist role. The consolidation process is continued by analyzing the SWITCH node. The following cases have to be distinguished:

1. Internists are in the intersection of the subject sets that are allowed to execute both branches (*ECG* and *in-patient treatment*).
2. Nurses working at the cardiology are only granted privileges for the *ECG*-branch.

The last step is to analyze the topmost SEQUENCE node for both cases. We compute the conjunction

$$\begin{aligned}
 &((role \sqsupseteq \text{Health Personnel} \wedge employment = \text{permanent}) \vee role \sqsupseteq \text{Administrative Pers.}) \\
 &\quad \wedge (role \sqsupseteq \text{Internist}) \\
 &= (role \sqsupseteq \text{Internist} \wedge employment = \text{permanent})
 \end{aligned}$$

Thus, subjects granted the Internist role are allowed to execute the complete workflow. The applicable consolidated policy is  $P_{(all)}^{opt}$  with

$$P_{(all)}^{opt} = \{((role \sqsupseteq \text{Internist} \wedge employment = \text{permanent}), (table = \text{MedicalRecordsTab}), (action = \text{select} \vee action = \text{update}), (\text{true})), ((role \sqsupseteq \text{Internist} \wedge employment = \text{permanent}), (table = \text{DevicesTab}), (action = \text{select} \vee action = \text{update}), (\text{true})), ((role \sqsupseteq \text{Internist} \wedge employment = \text{permanent}), (table = \text{PharmaceuticalsTab}), (action = \text{select} \vee action = \text{update}), (\text{HighAnaphylaxisRisk}(\text{patient}, \text{drug}) = \text{false}))\}$$

Additionally, according to the definition of  $P_{MR}$  and the role hierarchy depicted in Figure 2.2(a), the subjects that are allowed to execute the *ECG*-branch are those that are granted the Nurse role. The consolidation for this branch results in

$$P_{(ECG)}^{opt} = \{((role \sqsupseteq \text{Nurse} \wedge employment = \text{permanent} \wedge field\ of\ activity = \text{cardiology}), (table = \text{MedicalRecordsTab}), (action = \text{update} \vee action = \text{select}), (\text{true}))\}$$

### 3.5.2.3 Interpretation

The conclusions to be drawn from the structural analysis are

1. Internists and nurses are authorized to execute the workflow: Internists are granted *full authorization*, while nurses are granted *partial authorization* restricted to the *ECG*-branch. These two roles constitute least required roles according to Section 3.3.6.

Other subjects, like those granted the *Administrative Personnel* role, can be blocked right from the beginning.

2.  $\Pi_{O,A}(P_{(all)}^{opt})$  and  $\Pi_{O,A}(P_{(ECG)}^{opt})$  entail the least required privileges for *full* and *partial-authorization*, respectively. This information can be used to define task specific roles.
3. The number of policy enforcement points (PEP) can be reduced as illustrated in Figure 3.15:

- Realizing the *full authorization*-approach, policy  $P_{(all)}^{opt}$  will be enforced on the workflow layer through the first PEP. In this case, only internists will be authorized for the execution of the workflow.
- *Partial authorizations* need to be validated by two PEPs. In addition to  $P_{(all)}^{opt}$ ,  $P_{(ECG)}^{opt}$  needs to be enforced before the invocation of the *Query medical records* sub-activity and when entering the switch.

Policy enforcement costs can be reduced if access control of the sub-activities can be delegated to the workflow management system. That is, the security system of the workflow engine / service platform has to be capable of performing access control at the workflow layer.

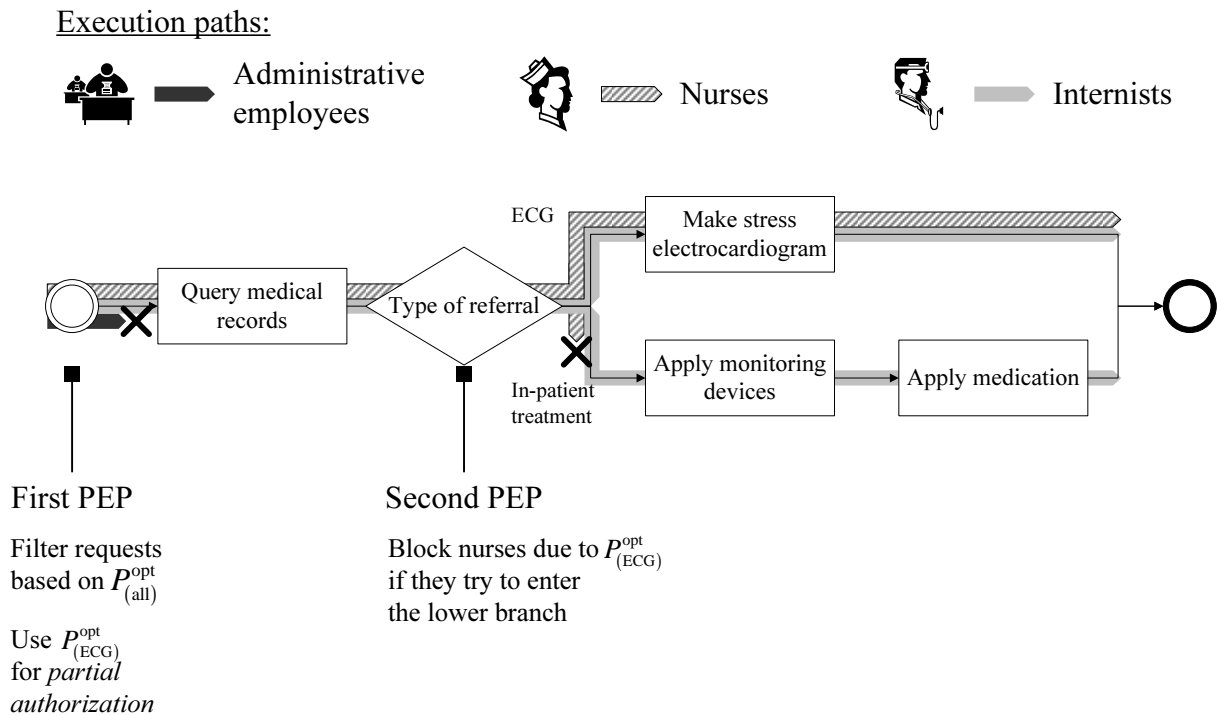


Figure 3.15: Optimizing the access control through policy enforcements at the workflow layer

Web services are self-contained software components, so that by default access control is performed at the services' layer. Nevertheless, enforcement costs can be reduced by including workflow relevant access rules into service policies. Instead of repeatedly initiating subject identification, only certain "pass-through"-credentials (e.g., realized as SAML assertions) are employed, allowing better performing security evaluation. Such "pass-through"-credentials are issued if access control at the workflow layer succeeds. Another possibility is to set up a security context between the workflow execution system and the autonomous services, e.g., by employing WS-SecureConversation [Gudgin and Nadalin (2005)]. This way, the access control of core business processes that are implemented as stateful Web service workflows can be realized efficiently. These approaches can often be realized for intra-organizational business processes, when workflow architects have the possibility to optimize the Web services' policies. In any case, policy enforcement at the workflow layer helps to reduce unnecessary service executions, transaction rollbacks, and compensating transactions.

Furthermore, policy consolidation allows to check the consistency of security configurations. Through the consolidation of the access control rules, *dead paths*, i.e., branches of the workflow that will never be executable, can be detected. Dead paths are identified via nodes with empty policies in the workflow tree.

### 3.5.3 Implementation

#### 3.5.3.1 Policy Consolidation Library

We implemented the proposed policy consolidation approach in Java, providing a library that is based on the policy and workflow specification standards XACML and BPEL:

**XACML** The eXtensible Access Control Markup Language is an XML based declarative policy language. It allows to formulate discretionary access control rules and to model the core RBAC functionality. XACML will be presented in Section 4.2.2. As we will show there, our policy model can seamlessly be expressed in XACML: The core components of rules, i.e., subjects, objects, and actions, are described through conjunctions of predicates. XACML Rule and Policy (respectively PolicySet) elements correspond to the introduced rule and policy definitions. Due to the extensibility of XACML, the rule combining algorithms presented in Section 3.2.2 can be implemented as well.

**BPEL** The Business Process Execution Language for Web Services (BPEL4WS or simply BPEL) is the emerging standard for specifying Web service choreographies. The language is currently getting standardized by OASIS and supposed to be renamed to WSBPEL.

Figure 3.16 illustrates the functionality of our policy consolidation tool. It generates consolidated workflow policies based on the original policies of the integrated Web services and a BPEL specification of the control flow:

1. First, the workflow tree is generated based on the BPEL description and the SEQUENCE and SWITCH patterns are identified.
2. For the individual Web services, the applicable policies are identified. XACML policies can be defined in a hierarchy. For the internal Java representation, hierarchies are flattened and distinguished rules according to our policy model are generated.
3. Then, policy consolidation as described in Section 3.3 is performed.
4. Finally, the resulting consolidated workflow policy is transformed into its XACML representation.

#### 3.5.3.2 Integration into SAP Research's Workflow Management System

The policy consolidation approach has been integrated into SAP Research's workflow management system consisting of the three components Maestro, Nehemiah, and Gabriel. Maestro is used to model business processes by specifying sub-activities and their interdependencies, i.e., the control flow [Aiouche (2005)]. Via drag-and-drop, components like sub-activity nodes and control flow nodes can be inserted and connected. Nehemiah is the workflow management engine that allows to execute business processes which have been designed using Maestro [Lippe (2004)]. At runtime, Nehemiah allows to supervise the state of the workflow by keeping track



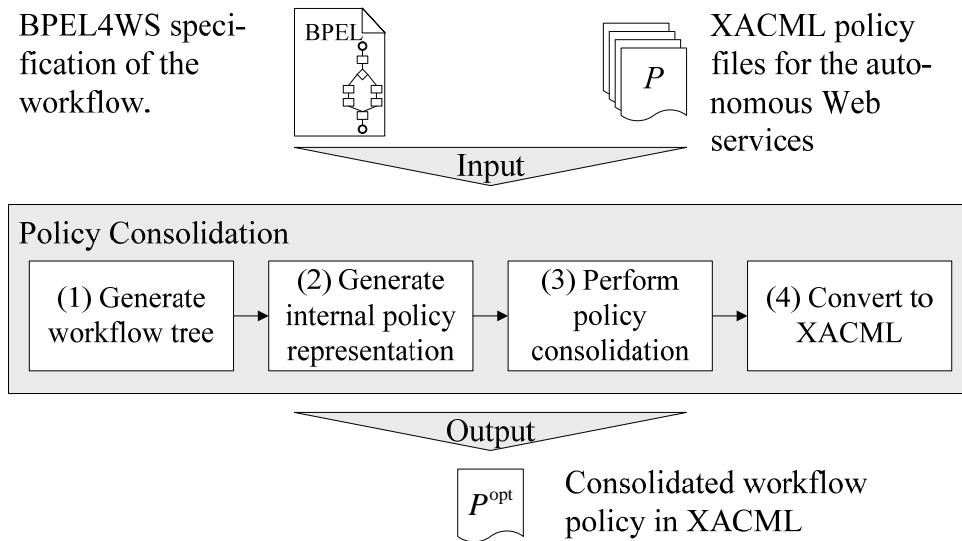


Figure 3.16: Processing steps of the policy consolidation prototype

of the active sub-activities. Thus, Maestro and Nehemiah are used for workflow modeling and activation. Individual sub-activities, on the other hand, are modeled and activated by Gabriel [Schaad and Spadone (2005)]. By use of Gabriel the roles needed to execute sub-activities are specified. At design time, sub-activity profiles are defined that describe which actions have to be performed when executing a certain task. For instance, an action can be the invocation of a Web service. When modeling a workflow with Maestro, sub-activity nodes can be associated with the corresponding sub-activity by means of the profile. Furthermore, subjects like roles and users can be modeled and these subjects can be granted the privileges required to execute the corresponding sub-activities.

The relationships between the three tools are illustrated in Figure 3.17. At runtime, Nehemiah interacts with Gabriel giving notification about active sub-activities. Then, Gabriel figures out users that are allowed to execute the respective sub-activities. These sub-activities can then be reserved by legitimate users. After a sub-activity has been executed, Gabriel notifies Nehemiah to continue with the business process execution.

Nehemiah supports the execution of multi-user workflows, denoting that sub-activities can be executed by varying users. We complemented the workflow execution engine with the special treatment of single-user workflows and composite applications, integrating the full-authorization approach [Albutiu (2006)]. Figure 3.18 illustrates the interplay of SAP Research's workflow management tools and the policy consolidation library. The consolidation is initiated by Nehemiah evaluating the workflow specification provided by Maestro. The access control policies of the individual sub-activities are queried from Maestro. The consolidated policy is used by Nehemiah to enforce access control for single-user workflows. Sub-activities of a single-user workflow are still activated by Gabriel, but, policy enforcement at this layer is no longer necessary. Instead, policy enforcement proceeds at the workflow level. The single-user execution

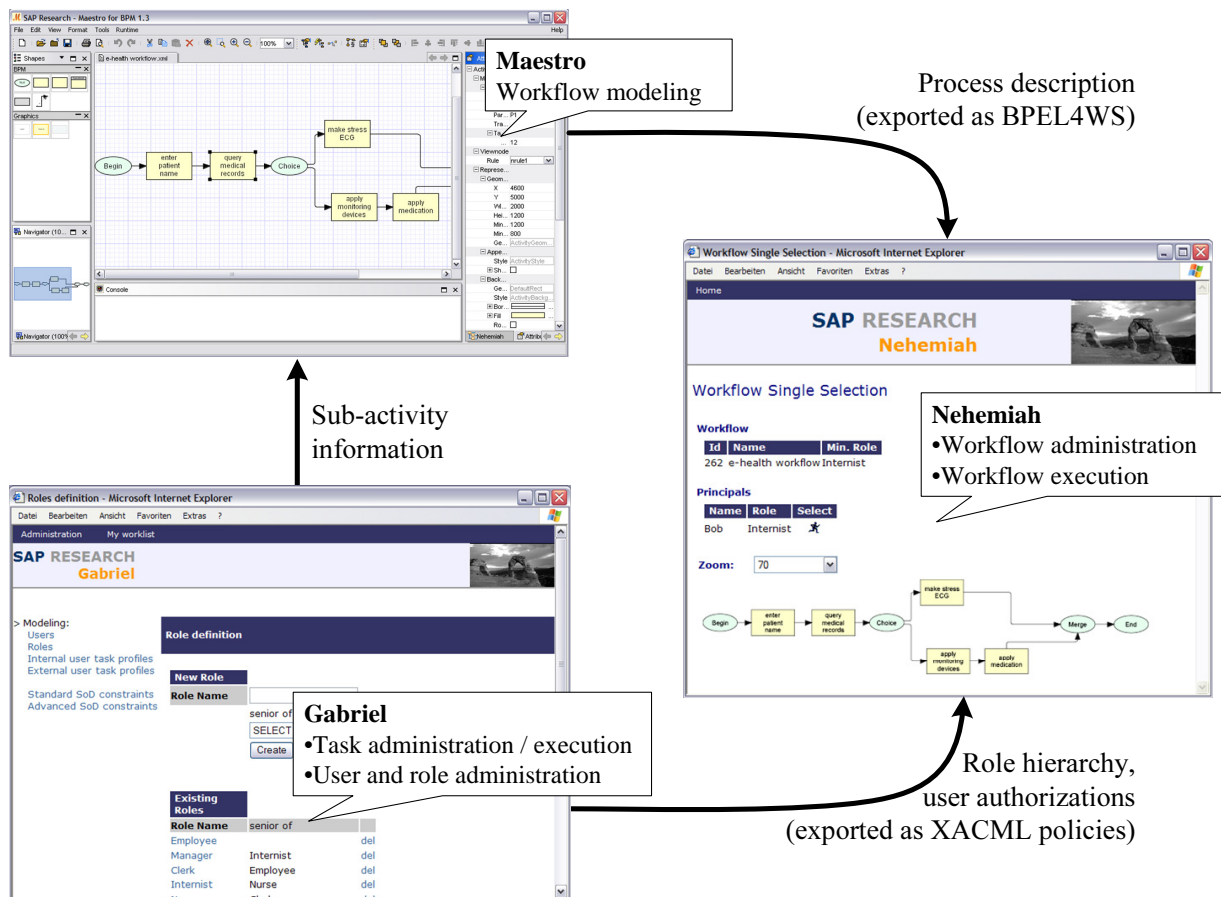
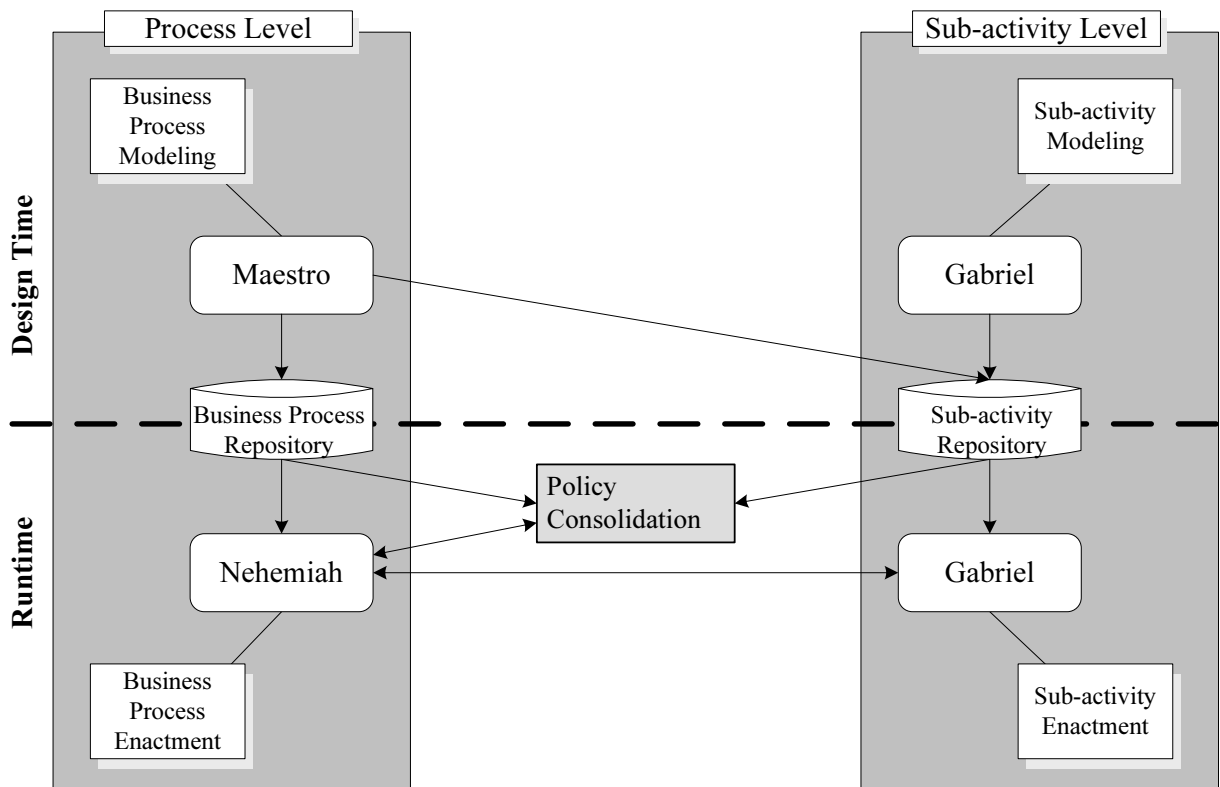


Figure 3.17: SAP Research's workflow management system

is conceptually closely related to SAP Guided Procedures which is a flexible workflow environment easing the development of (collaborative) business processes by use of reusable templates. Weilbach and Herger (2005) emphasize that SAP Guided Procedures excels in usability and auto-configuration. Our proposed policy consolidation approach allows to detect authorization templates for single-user workflows. These templates can be used to safeguard the execution of single-user workflows.

### 3.6 Related Work

In Section 3.2, we defined the policy model that constitutes the formal basis of our proposed policy consolidation technique. Bonatti et al. (2000, 2002) and Wijesekera and Jajodia (2001, 2002, 2003) present policy algebras that allow combining and comparing access control policies on a formal basis, building upon previous work on the composition and secure behavior of program modules by Abadi and Lamport (1993) and Jaeger (1999). Backes et al. (2004, 2003) present an



(adapted from [Schaad and Spadone (2005)])

Figure 3.18: Policy consolidation within SAP Research's workflow management tool-suite

algebra for the composition of enterprise privacy policies. Syntax and semantics of our model are closely related to the model proposed by Bonatti et al. We extended it where necessary by adding additional operators and definitions of policy and rule relaxation.

The model is very expressive by allowing entities like subjects and objects to be expressed in an attribute based way, rather than relying on identity based descriptions. Therefore, the model is well applicable for (Web) applications that demand for fine-grained access control. The individual attributes are defined on partially or totally ordered sets. Thus, for instance, general and limited role hierarchies as introduced in Section 2.2.3 as well as partial orders on objects and actions can be defined. According to the database object hierarchy, privileges defined on generic levels (e.g., a complete table in a database) include privileges for subordinate levels (like columns or individual tuples). Our definition of privilege and rule relaxation is also related to early work on access control for object-oriented database systems. Rabitti et al. (1991, 1988) and Fernandez et al. (1994) introduce concepts like authorization inheritance into the specification and enforcement of access control rules. The access control schemes proposed by Richardson et al. (1992) and Ahad et al. (1992) realize the encapsulation concept, denoting that objects are accessed by use of methods. Users can be granted privileges for executing these methods but need not be granted comprehensive privileges for the objects themselves.

Access control policies of composite applications are composed of rules that codify the individual access rights relating to the underlying sub-activities. The enforcement of such policies depends on the applied evaluation algorithm. Logical frameworks for the specification and enforcement of access control are, for example, presented by Jajodia et al. (1997a,b). If negative or mixed authorization should be employed, which could be expressed in our model as well by means of the subtraction operator, conflict resolution techniques have to be employed. Jajodia et al. (2001) and Bertino et al. (2001, 1999b) deal with policy evaluation strategies for mixed authorization systems. A high diversity of evaluation algorithms is also offered by the XACML standard [Moses (2005)]. Different policy evaluation algorithms have been presented in this chapter when defining the consolidation of the access control policies of composite applications. Concerning the optimization of workflows including autonomous sub-activities like individual Web services, we focused on positive authorization which is suitable for almost all enterprise applications as stated by Atluri (2001).

Our work is also related to research on models for the specification and analysis of workflow processes [Altunay et al. (2005); Kang et al. (2001)]. Adam et al. (1998) use Petri-nets to model and evaluate control flow dependencies and to infer secure execution configurations. Policy consolidation allows to optimize the access control of workflow systems by providing a consolidated policy that can be enforced at the workflow layer. The enforcement of access rules at the workflow layer is also proposed by Gudes et al. (1999). But in contrast to our approach, the authors focus on multi-user workflows where each step is executed by different individuals. Access control models and architectures for workflow systems are, for example, proposed by Atluri and Huang (1996), Huang and Atluri (1999) and Bettini et al. (2002). Bettini et al. identify temporal constraints that can cause inconsistencies restricting the executability of workflows. Temporal constraints must also be considered when interpreting the result of the static policy analysis. Nevertheless, even if dynamic dependencies have to be evaluated at runtime, policy consolidation still offers optimization potential. This technique allows to improve the performance of workflow systems since unauthorized execution attempts can be filtered at an early stage and runtime policy enforcement costs can be reduced based on consolidated policies.

Atluri et al. (1997, 2000) examine the security requirements of multilevel secure workflows, i.e., workflows where the individual sub-activities can be assigned to varying security levels. Bertino et al. (1999a) introduce a formal model and framework for the enforcement of static and dynamic separation of duties in workflows. Schaad et al. (2005) describe separation of duty requirements by means of a case study based on the legislative procedure in Austria. These approaches are somehow orthogonal to ours, as we are performing the consolidation process from the *single-user/single-role* viewpoint. That is, we are focusing on composite applications that are implemented for specific jobs and, thus, do not rely on user/role switching.

In Section 3.5, we showed how the consolidation approach can be applied to Web service compositions. Khalaf and Leymann (2003) give an overview over service composition strategies. Robinson et al. (2006) describe how the access control configuration of business processes can be inferred from the control flow. With the aim of fulfilling the principle of least privilege, policies are enabled and disabled depending on the workflow state. In all probability, due to widely accepted standards, like XML, SOAP, UDDI, and WSDL (to name the most important ones),

enterprise application integration challenges (EAI, [Linthicum (2001, 2003); Ruh et al. (2000)]) will increasingly be solved by use of Web services. We also built our prototypical implementation on the predominant workflow and access control standards in this area, especially BPEL4WS and XACML. Identifying the set of authorized users of composite applications for access control optimization is also addressed by Rits et al. (2005). They present an approach for determining task-specific user profiles and roles by analyzing the source code of composite applications. The approach we present in this chapter is more generic and implementation independent. It also supports the reverse engineering of appropriate user/role profiles by determining the least required privileges. Compliance with the principle of least privilege can be inferred if the policies of the underlying sub-activities are preprocessed and minimized. How this can be achieved for database backed Web services is shown in the next chapter.

### 3.7 Conclusion

Security and usability are often experienced as counteracting objectives [Cranor and Garfinkel (2005)]. However, the policy consolidation approach presented in this chapter is an example for both being able to be synergistic as well. Through the consolidation of the access control configurations of autonomous sub-activities, authorization complexity is reduced. Application developers and security officers receive a top-level view that allows them to detect possible security vulnerabilities more easily. For example, over-privileged user accounts can be substituted by task-specific role profiles and predefined access rights for a composite application can be tailored to the respective functionality. Furthermore, access control that is performed on the workflow layer based on a consolidated policy can reduce policy enforcement costs. Additionally, it helps to avoid transaction rollbacks and compensating transactions through detecting and blocking insufficiently authorized execution attempts early.

Policy consolidation was introduced on the basis of a formal policy algebra that allows DAC and RBAC models to be expressed. Due to the attribute based description of entities like subjects and resources, the model is very expressive and useful for business applications that demand for fine-grained access control specifications. The calculation of consolidated policies is reduced to a structural analysis of the workflow tree. Through a bottom-up analysis, intermediary consolidated policies are calculated for the inner nodes of the workflow tree until a consolidated workflow policy is obtained, finally. The resulting policy either applies to the complete workflow or to the execution of selective workflow branches, depending on whether the *full* or *partial authorization* approach is employed.

In order to consolidate policies, privilege relaxation checks and subject intersections have to be calculated, for which algorithmic solutions were presented. The complexity of policy consolidation is subdivided into the complexity that is determined by the structural analysis and the complexity of policy comparisons. The complexity of the structural analysis depends on whether *partial authorizations* have to be analyzed and the number of SWITCH nodes that are included in the control flow. The complexity of privilege relaxation checks is closely related to the predicate implication problem. Whether the exponential worst case complexity arises, depends on the degree of partial overlaps between the conjunctive subterms of privilege specifications. The

worst case is attained if policies are described in a rather unstructured manner what is assumed to arrive rarely in practice.

The benefits of policy consolidation for composite applications were demonstrated by the example of Web service workflows. We emphasized that for Web service workflows typically no predefined policies have to be adjusted. Instead, the focus of consolidation is on determining the set of authorized users. In the following chapter, we will also give an example for the second consolidation objective by presenting a security engineering approach for tightening the access control of database backed Web services. Our aim is to avoid security vulnerabilities that arise due to failure to comply with the principle of least privilege.

---

# Security Engineering for Database Backed Web Services

---

Many enterprise services rely on database backends to store and process business relevant data. Thus, (the contents of) databases in many cases constitute the heart of a company's information processing systems, determining its value and success. With the advent of the service-oriented computing paradigm, a change in the design of enterprise resource planning (ERP) systems is recognizable. Database functionality is appreciably provided over Web service interfaces used to realize intra- and increasingly also inter-organizational value creation chains. In general, database systems in this case are no longer guarded by the security system of a wrapping monolithic middleware system. Hence, new approaches are required to ensure security of database systems that are integrated over lightweight Web services.

The access control of database backed services can be defined from two extreme viewpoints. On the one hand, security can be enforced by the underlying database systems which then have to realize application specific access control. On the other hand, authorization can be enforced solely by the services. This requires the services to run under control of the database administrator as service-to-database connections are established by use of highly privileged database profiles. None of these approaches appears to be satisfactory. Instead, application specific security has to be realized at the application layer, i.e., the services, while autonomy of authorization for the underlying database systems is preserved at the same time. Therefore, multilayered access control appears to be promising. But, in case access control policies of services are defined without correlation to the database policies, authorization mismatches are likely to be induced. This demands for a methodical approach that achieves the consolidation of both security configurations.

The remainder of this chapter is structured as follows: In Section 4.1, we discuss the security requirements of database Web services, i.e., services that rely on database interactions for providing their functionality. Section 4.2 presents an overview of access control concepts for Web services and database systems. In Section 4.3, we present our security engineering approach for

consolidating the access control of database backed Web services, before introducing our prototypical realization in Section 4.4. Finally, we give an overview over related work in Section 4.5 and conclude in Section 4.6. Parts of this chapter have already been published in [Wimmer et al. (2004)] and [Wimmer et al. (2005a)].

## 4.1 Motivation

Currently, a change of the architecture of ERP applications is becoming apparent. Because of increased demand for flexibility, adaptability, and the need to quickly react to market shifts, more and more enterprises are migrating to service-oriented architectures. Nevertheless, this architectural change also brings about new challenges concerning the security of business applications and respective underlying resources. In the past – and still today –, databases have often been integrated into large-scale monolithic ERP systems, like SAP R/3-installations. Classical ERP systems represent single points of administration, typically supplied with a robust security system which has been evaluated over the years in practice. Usually, database systems are fully integrated into the middleware layer, meaning that in many cases, database connections are established over ‘root’-like accounts and fine-grained authorization is enforced on the middleware and application layer.

Applying this approach one-to-one to database backed Web services – in the following also referred to as *database services* for short – can result in significant security vulnerabilities for the underlying resources. Web services are lightweight software components – possibly even mobile code – that can be executed on various service hosts. Typically, services only demand a restricted portion of the data processing capabilities of database systems. Hence, providing comprehensive access on databases is not meaningful from the security viewpoint. Nonetheless, service-to-database connections are often established by the use of over-privileged database accounts. The reason for this is that determining adequate database user profiles for a given service functionality is a non-trivial task, in general. Moreover, more generic accounts can be reused for the realization of varying services, reducing database administration effort, thus, easing service implementations.

Figure 4.1 sketches an example of two services accessing the same database. Each service requires access to only an extract of the database content, expressed via the *access corridors* AC1 and AC2. The term ‘access corridor’ represents the privileges which are granted by the employed database connections. In the figure, both corridors overlap. Thus, it appears beneficial to use only one database account like `dbuser` for realizing both service interfaces. In case security leaks of the services or the service platform are disclosed, confidential data (which is not provided via the services) is in danger of being accessible, like ▼ and ■ in the figure. Two extreme approaches concerning the specification and enforcement of the access control of database services can be observed.

On the one hand, following the database-centric approach, access control for database services can be realized by the underlying database systems. Consequently, security enforcements on the services’ side are not implemented and policy evaluation costs (like the browsing of XML policy files) can be saved. However, the application’s security policy has to be implemented by



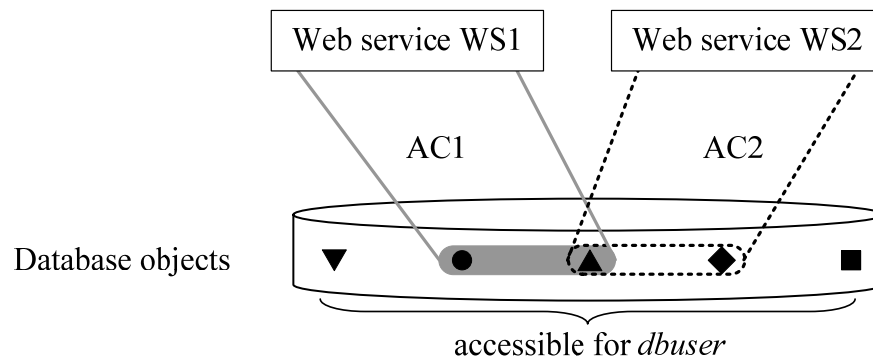


Figure 4.1: Reducing security vulnerabilities through access corridors

the database which appears to be impractical in many cases. Moreover, service implementations can demand for fine-grained access control, which cannot be realized satisfactorily, depending on the employed database system.

On the other hand, authorization can be enforced solely on the application layer. This service-centric approach brings about the security risks discussed above. The common practice lies in-between these two extreme approaches, denoting a multilayered access control scheme. Consequently, two access control systems have to be taken into consideration, namely the security system of the service and the security system of the database. But, in case the two are administered independently, authorization mismatches are likely to occur.

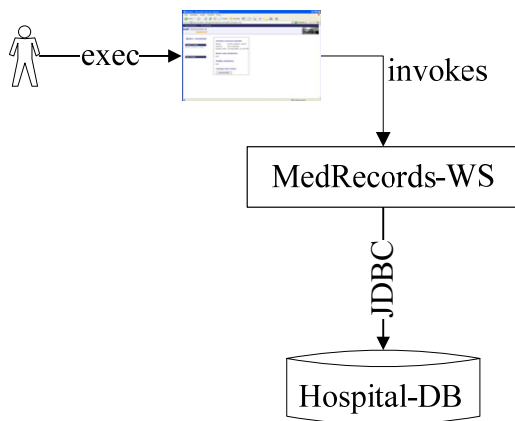
In this chapter, we propose a best-practice for the implementation of database services. Based on the service specification, Web service policies, policy enforcement points, and optimal database user profiles can be generated semi-automatically. To ensure the compatibility of service policies with database security policies, we perform a policy compliance check that relies upon the techniques presented in the previous chapter.

### Running Example

We will demonstrate our security engineering approach by means of a simplified e-health scenario. Figure 4.2 sketches a Web service portal called *MedRecords-WS* which physicians can use to change the medication of patients. The signature of `updateTreatment` and the corresponding update statement are illustrated in the upper right corner of the figure. The lower half shows an extract of the hospital's database. For the sake of simplicity, we assume patients and physicians to be identified by their names.

Usually, access control takes place in two phases: First, requests are authorized by the Web service. Afterwards, a connection to the underlying database is established that is authorized by the database system. Thereby, service clients typically differ from the database accounts used to set up the service-to-database connection.

For the implementation of (database) Web services, the ability to realize fine-grained access control (FGAC) is of particular importance. For instance, the authorization of subjects often

**Web service method:**

```

updateTreatment (
  p <string>,    // patient name
  d <string>,    // disease
  m <string>)    // new medication
  
```

**SQL-query:**

```

update MedicalRecords
set Medication = m
where Patient = p
and Diagnosis = d;
  
```

**Extract of the database schema:**

Physicians	
Name	Department
John Carter	Emergency Room
Kerry Weaver	Cardiology
Mark Greene	Surgery
⋮	⋮

Patients	
Name	Health Insurance
Philip Watters	ABC Insurance
Kate Austin	Private HI
⋮	⋮

MedicalRecords			
Patient	Diagnosis	Medication	AttendingPhysician
Philip Watters	Cold	Cough Syrup	John Carter
Kate Austin	Corn	Band-aid	Kerry Weaver
⋮	⋮	⋮	⋮

Figure 4.2: Architecture of a simple database service

depends on their attributes like age, reliability, and customer relationship. Furthermore, the applicability of access rules can depend on conditions. In our scenario we want to realize the use case that only attending physicians can decide about the medication of patients.<sup>1</sup>

In the next section, we give an overview of the access control schemes of (relational) database management systems. Afterwards, we discuss policy enforcement techniques for Web services and propose a best-practice for the implementation of database backed services.

<sup>1</sup>FGAC requirements of e-health applications, in particular in the context of the German e-health card, are discussed by Caumanns (2006).

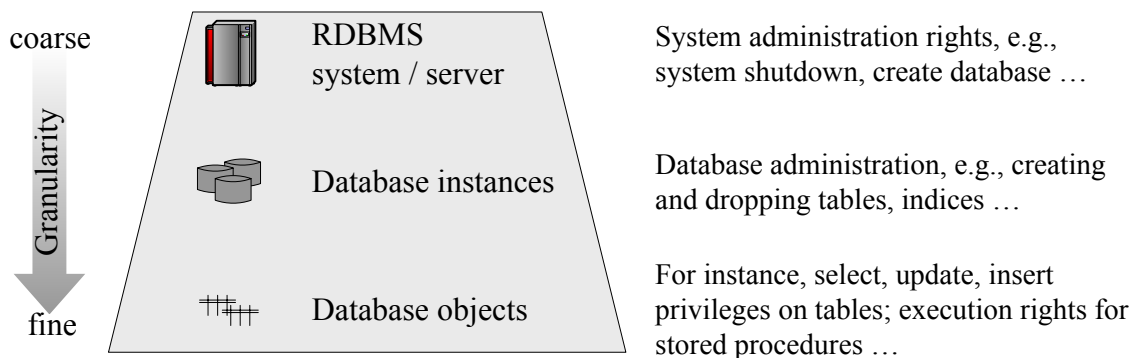


Figure 4.3: Access control granularity levels of RDBMS

## 4.2 Access Control of Database Systems and Web Services – the two Poles Apart

### 4.2.1 Access Control of Database Management Systems

Recent Gartner reports state that there is still an “ongoing demand for relational database management systems” [Petty (2006)] in the upcoming years. According to Graham (2006), the top 3 vendors accounting for approximately 85 percent of the total market are Oracle, IBM, and Microsoft. In the following, we give a brief overview of the basic access control concepts of relational database management systems (RDBMS), focusing on the three commercial RDBMS IBM DB2 UDB 8.2, Microsoft SQL Server 2005, and Oracle Database 10g. Details about the presented security concepts can be found in the listed literature and software manuals.

In order to perform activities on a database, database users must be granted respective privileges. This can be done either explicitly or implicitly, e.g., via group or role membership. Regarding the three RDBMS, privileges can be assigned at different granularity levels. These are illustrated in Figure 4.3. At the topmost level, users can be granted system administration rights, like the privileges to start or shut down the RDBMS server or to create or delete user accounts. The medium level is the database instance level. For example, users can be allowed to modify the database scheme by creating, altering or dropping tables. At the lowermost and most fine-grained level, users can be granted access to individual database objects like tables, columns of tables, and stored procedures. In general, privileges granted at higher levels in the hierarchy are more powerful and can include privileges for underlying levels. To give an example, system administration privileges imply privileges to create and access databases and respective database objects.

#### 4.2.1.1 Access Control Concepts of IBM DB2 UDB 8.2

In IBM DB2, users are authenticated by the security system of the operating system or dedicated security systems like Kerberos. According to Orhanovic et al. (2004), users can be granted

privileges by the following three possibilities:

1. They are the owners of the respective database objects. Owners are granted the `CONTROL-` privilege, which allows them to perform any action on it, like modifying or deleting it.
2. They are explicitly authorized. In order to get explicitly authorized, users need to be assigned privileges by authorized users by use of the `grant`-statement. For instance, the user `carter` (which is the database profile of the physician John Carter) can be granted read and write access on the table `MedicalRecords` via

```
grant select, update, insert on MedicalRecords to carter;
```

By use of the `with grant` option, John Carter will also be authorized to delegate the privileges. Some privileges can be refined and restricted to single columns. For instance, if John Carter shall only be allowed to update the diagnosis of a treatment, the following statement can be used:

```
grant update (Diagnosis) on MedicalRecords to carter;
```

3. They are implicitly authorized. For instance, privileges can be granted to all users (through assigning them to `PUBLIC`). Furthermore, users can inherit privileges by being members of authorized (operating system) groups.

Fine-grained access control can be realized by use of restricted views which allow restricting access on tables to certain rows (horizontally) or columns (vertically). Assume that John Carter is a surgeon and that we want to restrict access on the `MedicalRecords` table for him to information on patients that are treated by surgeons. This can be achieved by use of the following view:

```
create view SurgicalTreatments as
select mr.*
from MedicalRecords mr, Physicians p
where mr.AttendingPhysician = p.Name
and p.Department = "Surgery";
```

```
grant select on SurgicalTreatments to carter;
```

Using the keyword `CURRENT_USER`, views can be defined restricting access to records that refer to the currently logged on user.

```
create view OwnTreatments as
select mr.*
from MedicalRecords mr
where mr.AttendingPhysician = CURRENT_USER;
```

```
grant select on OwnTreatments to carter;
```

As a prerequisite for this example to work, the database accounts need to be stored in the `AttendingPhysician` column (i.e., `carter` instead of John Carter as shown in Figure 4.2).

It is also possible to grant predefined bundles of privileges to users by assigning them certain database authority profiles. For example, `SYSADM` is the highest authority with full access to

the RDBMS. On the database level, the DBADM authority grants comprehensive access rights on a specific database. Authorities and individual privileges can be withdrawn using the `revoke` command. A comprehensive presentation of the access control features of IBM DB2 is given by DB2's SQL Reference (1993 – 2004).

#### 4.2.1.2 Access Control Concepts of Microsoft SQL Server 2005

The access control of Microsoft SQL Server 2005 covers the basic access control concepts of IBM DB2. That is, the ownership paradigm is realized and users can be granted privileges explicitly and implicitly as stated above. Additionally, SQL Server 2005 realizes (hierarchical) RBAC. On the one hand, there exist predefined server roles, like the SYSADMIN role which is quite similar to DB2's SYSADM authority. Furthermore, also predefined database roles exist. For example, DB\_DATAREADER and DB\_DATAWRITER grant read and write access on all tables of a database. On the other hand, the security system of SQL Server 2005 supports the specification of new user-defined roles. By use of the `grant` statement, privileges are assigned to roles. Roles are assigned to users by means of the `sp_addrolemember` method. Thus, job specific roles can be realized, easing the administration of security policies. For instance, in the context of our example, distinguished roles like Surgeon, Internist, etc. (see Figure 2.2 on page 11) can be defined. To give an example, the role Surgeon is assigned to the database user `carter` via:

```
sp_addrolemember Surgeon, carter;
```

Using RBAC, access on the view `SurgicalTreatments` can be granted to the role Surgeon, instead of granting this privilege to each individual surgeon.

One specific feature of Microsoft SQL Server 2005 is its support of negative authorization by means of the `deny` statement. The syntax of `deny` is similar to the `grant` statement. In case of conflicts, i.e., situations where permissions and denials apply at the same time, denials take precedence.

To sum up, according to Bauder (2006), database accounts can be classified into one of the following groups:

- Accounts with access to all tables, views and procedures of a database. For them, authorization can be arranged by use of predefined database roles.
- Accounts with access to many but not all tables, views and user defined procedures of a database. Then, predefined database roles like DB\_DATAREADER and DB\_DATAWRITER can be used as starting basis and respective authorizations can be denied.
- Accounts with access to only a restricted set of database objects. The respective privileges can be assigned to these accounts individually or by means of user-defined roles.

Further details about the access control of Microsoft SQL Server 2005 are, for example, provided by DeBetta (2004), Dröge and Raatz (2005), and online at MSDN<sup>2</sup>.

---

<sup>2</sup><http://msdn.microsoft.com/sql/learning/Security/>

### 4.2.1.3 Access Control Concepts of Oracle Database 10g

Oracle Database supports the basic authorization concepts provided by IBM DB2. Like SQL Server 2005, it also provides role based access control. Advanced access control configurations are possible by use of Oracle VPD and OLS:

- With *Virtual Private Database* (VPD), data in a database can be masked so that users can only access data that are relevant for their job profiles and comply with their classification. VPD allows to restrict access through declarative policies. At runtime, VPD policies are transcribed into `WHERE`-clauses in queries. In principle, policy enforcement is similar to constraining access through restrictive views. It's advantage is that the specification of access control is separated from the application logic.
- *Oracle Label Security* (OLS) realizes multilevel security concepts. In OLS, database objects like rows of a table are classified (e.g., as sensitive or confidential) and users are assigned to security levels. The enforcement of OLS policies refers to mandatory access control as described in Section 2.2.1.

Details about the access control features of Oracle Database 10g are, for example, provided by Loney (2005), Fröhlich et al. (2005), and Haas (2006).

## 4.2.2 Access Control Mechanisms for Web Services

Priebe et al. (2005) state that modern e-commerce and e-government applications (as well as other Web-based fields of applications like e-science and e-health) demand for flexible and fine-grained access control schemes. This is due to the multitude and heterogeneity of users and the diversity of resources (e.g., different granularity levels of information). A modern policy language which is well suited for specifying the access control of Web services is the eXtensible Access Control Markup Language, XACML [Moses (2003, 2005)]. XACML is a declarative policy language that allows to express DAC and RBAC rules and supports positive, negative, and mixed authorization. Since it is based on XML, XACML seamlessly fits into the Web services' technology stack. That is, existing parser technologies can be utilized and basic language constructs are well known to the community.

XACML provides a request/response mechanism. The enforcement process complies with the ISO/IEC 10181-3 framework. In Figure 4.4, this approach is shown for the security enforcement of Web services. Services act as policy enforcement points, PEP. Incoming SOAP-requests are transformed into appropriate XACML-requests, eventually enriched with additional context information. This, for instance, can be done by the Web service itself or through a dedicated Context Handler. XACML-requests are then passed to the policy decision point, PDP. The PDP functionality is typically provided by the underlying Web service platform. The PDP evaluates requests against applicable policies that are supplied by a policy administration point (PAP). Finally, decisions are sent back to the Web service that either executes as demanded or aborts and returns an error message in case authorization failed.

Figure 4.5 shows the XACML policy model. An XACML Policy consists of a Target, a set of Rules, and optional Obligations specifying access conditions. Rules contain most of the logic of

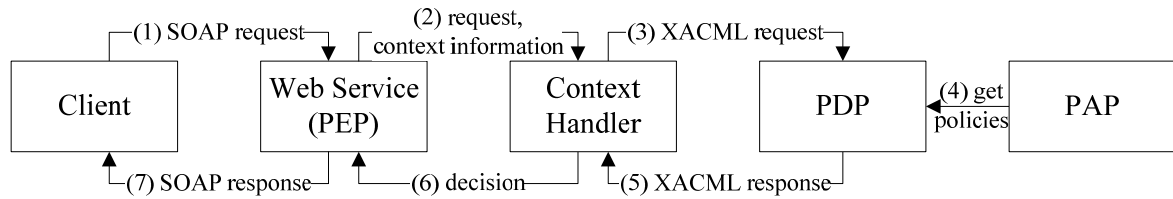


Figure 4.4: Control flow of the policy enforcement process

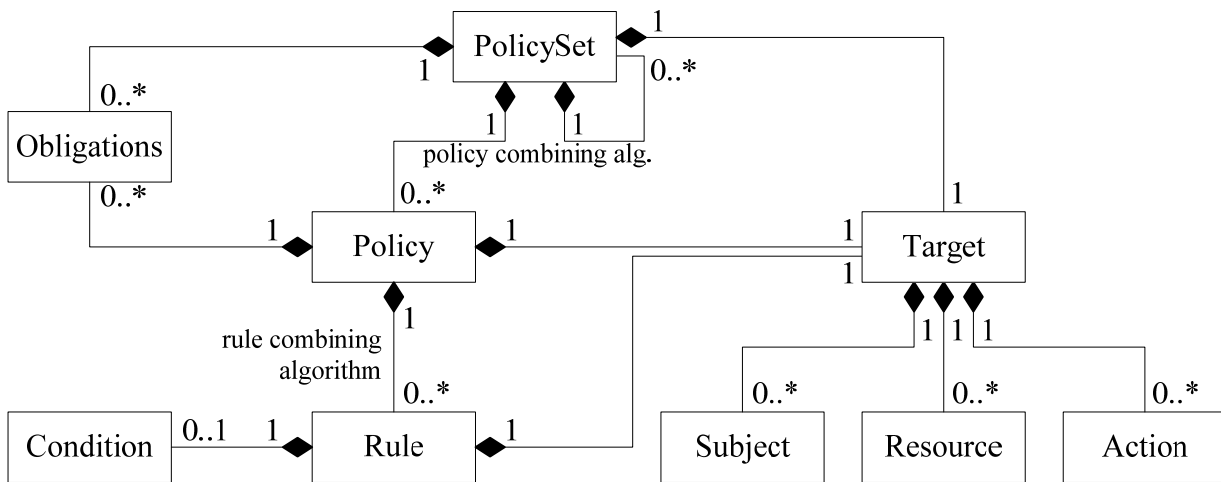
a Policy. A single Rule is set up by a Target, an optional Condition, and an Effect which is either Permit or Deny. The way several rules are combined in a Policy is specified by the rule combining algorithm. This is needed in particular to resolve conflicts in mixed authorization systems. To give some examples, denials take precedence by use of the algorithm deny-overrides, while permissions take hold when using permit-overrides. In case a processing error occurs during policy evaluation, the result will be Indeterminate. A Policy evaluates to NotApplicable if none of its Rules applies. The XACML standard defines several combining algorithms and allows the specification of new ones. Several Policy instances can be aggregated in PolicySets that are evaluated according to the policy combining algorithm, which is the policy-analogue to the rule combining algorithm.

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. Conditions can be quite complex and allow arbitrary Boolean terms to be expressed.

The Target of a Policy or Rule is composed of Subjects, Resources, and Actions. The Target thus constitutes the actor, the module to be accessed, and the operation to be performed. The child-elements of Subjects are Subject nodes. Each individual Subject is described through a conjunction of attribute comparisons. In case (part of) a Target is undetermined, i.e., unrestricted, AnySubject, AnyResource, and AnyAction are used. The refinement of a Rule takes place by imposing further constraints on its Target, thus, restricting the Rule's applicability.

Policies are evaluated against requests. An XACML Request is composed of one or many Subject elements that represent the initiator of a Request, e.g., the identity and/or characterizing attributes of human requesters. Further elements of a Request are the Resources and the requested Action that together specify the activity, i.e., the resource that shall be accessed and the way the access shall be performed. A Request is evaluated against applicable Policy and Rule instances whose Target matches with the Request. A Subjects specification is evaluated disjunctively, while an individual Subject consists of SubjectMatch conjunctions that specify its attributes. Thus, for the match of the Subjects defined in a Target with the attributes of a Request, at least one of the Subject elements must apply, while for the match of one Subject it is necessary that all associated attributes match with the Request. Analogous considerations apply to Resources and Actions.

XACML is a quite expressive declarative policy language and well suited for realizing the access



(adapted from [Moses (2005)])

Figure 4.5: XACML policy model

control of Web services. Apart from the core XACML specification, further additional specifications exist. Anderson (2005) presents the core and hierarchical role based access control profile of XACML, which is of particular relevance for our work. Other security standards like SAML can as well be employed to specify and enforce security of Web services (see Section 4.5). Since our policy model introduced in Section 3.2 can seamlessly be represented in XACML, we use XACML as policy language and enforcement mechanism for our prototype.

According to Section 3.2.1, subjects, objects, and actions are defined through attribute comparisons given in disjunctive normal form (DNF). Thus, their specification and semantics match with those of Subjects, Resources, and Actions of the XACML standard: The individual conjunctive subterms of a DNF are represented as child elements, i.e., Subject, Resource, and Action elements. Elementary attribute comparisons are represented by the XACML counterparts SubjectMatch, ResourceMatch, and ActionMatch for attributes in *S-Attr*, *O-Attr*, and *A-Attr*, respectively. Analogously, conditions of rules are represented by Condition elements in XACML. Thus, rules and policies of our policy model can be represented by XACML Rule and Policy elements. The evaluation of policies as described in Section 3.2.2 can be expressed as well, because XACML supports the definition of new rule combining algorithms. Therefore, unless explicitly indicated, we equate terms like policy and subject with the corresponding XACML element tags, i.e., Policy and Subject, and vice versa.

### 4.2.3 Access Control of Database Web Services

In this section, we compare different approaches for realizing the access control of database services. The approaches can be evaluated with regard to (1) their support for fine-grained access control (FGAC), (2) possibilities to flexibly adopt changes of the security policy, (3) the implementation costs, and (4) the expected risk of security flaws – which appears to be the most



crucial evaluation criterion. We compare the approaches by means of the previously introduced example of the service *MedRecords-WS*, whereby the following authorization constraint should be realized: Physicians shall only be granted access to medical records of patients they are the attending physicians of. Access to any other medical record has to be denied.

#### 4.2.3.1 Access Control as Part of the Application Logic

One possible approach is to include security checks into database query statements. Assume that *wsuser* is the client invoking the service method. The client is only allowed to change the medication of a patient if he or she is the attending physician. Therefore, we modify the SQL statement illustrated in Figure 4.2 to

```
update MedicalRecords
set Medication = m
where Patient = p and Disease = d
      and AttendingPhysician = wsuser;
```

This approach can be characterized as follows:

- (1) In general, FGAC is supported, as it is not restricted by the expressiveness of any policy language or the capabilities of an underlying policy model. Nevertheless, as this approach is not based on an established (formal, declarative) policy framework, application implementation is made difficult, thus, demanding for versed developers.
- (2) Modifications of the security policy are hard to realize, as these imply the modification and recompilation of the service.
- (3) Security functionality is only realized on the side of the service. Hence, database administration effort is low. But this approach typically brings about significant overhead for the implementation of the service if the security functionality has to be coded by hand.
- (4) The risk of security holes is relatively high, as the approach demands for an experienced programmer, who has to realize all security requirements and needs to be aware of all possible exceptional cases – which is a nontrivial and error-prone task.

We discuss this approach only for the sake of completeness, as it does not follow good coding principles. Implementing access control as part of the application logic makes the tasks of verification, modification and adequate enforcement difficult.

#### 4.2.3.2 Enforcing Access Control through the Database System

An alternative is to shift access control for database services to the underlying database management system (DBMS). Then, depending on the security system of the employed DBMS, access control mechanisms like role based access control, user restricted views or DBMS specific concepts like Oracle VPD can be used. This approach can be classified as follows:

- (1) The way FGAC is supported depends on the employed DBMS. The previous overview of the access control models of IBM DB2, Microsoft SQL Server, and Oracle Database shows that database systems support varying access control concepts and provide different possibilities to realize FGAC. Also, concepts that are supported by several systems can vary with regard to the employed syntax, so that access control schemes are not platform independent, in general.

In many scenarios, the FGAC functionality offered by database systems is inappropriate, i.e., not sufficient for realizing the required application security. For example, FGAC rules that evaluate properties of the requesters, while their plain identity is of secondary interest, usually cannot be modeled. As another example consider the following condition: A first-year resident should only be allowed to change the therapy of patients in case a chief physician agrees to the resident's decisions. This constraint cannot be enforced with common database access control mechanisms since it specifies the evaluation of security credentials of more than one subject.

- (2) Changes of a service's security policy imply changes of the access control scheme of the underlying DBMS. Although modifications are possible, they might be hampered if the service developer only has restricted administrative privileges for the DBMS.
- (3) The same applies to service deployment. Administration overhead is caused when the application developer does not have database administration rights and, thus, needs to cooperate with the DBMS administrator.

Furthermore, considering our sample database service, implementing the security functionality can be impractical: Let's assume that fine-grained access control is realized by means of views that are restricted to the identity of the logged on user – as an example consider the view *OwnTreatments* presented in Section 4.2.1.1. In the given example, each user of *MedRecords-WS* requires an account for the database. This clearly disqualifies this approach for applications with dynamic user basis, e.g., e-business applications where users can register and deregister on demand. Moreover, it is no useful concept for hiding the DBMS to the service users.

- (4) In case this approach is carried through thoroughly, security vulnerabilities for the database and the service can be avoided as it relies upon the security system of the underlying DBMS.

Altogether, this approach can provide a high level of security for the underlying database system if the database connections are restricted to the least required privileges. Unfortunately, realizing the security functionality can be impractical as it demands service developers and database administrators to cooperate closely. It's main disadvantages are on the one hand restricted FGAC capabilities and on the other hand a potentially high administration overhead. The latter can also induce security shortcomings in cases when relaxed database security policies are used in order to ease the implementation.

### 4.2.3.3 Performing Multilayered Access Control

Multilayered access control describes the process of enforcing database specific security requirements through the database and application specific authorization through the service. In the context of our example Web service *MedRecords-WS*, the connection to the database is established by use of a (potentially more generic) database account, say *dbuser*. This account is at least granted the privileges to set up connections to the database and to perform updates on the table *MedicalRecords*. Our example FGAC rule declares that only attending physicians can alter a patient's medication. This application specific rule is enforced at the service layer. In XACML, this rule can be expressed by means of a Condition:

```
<Condition FunctionId="string-equal">
  <Apply FunctionId="string-one-and-only">
    <SubjectAttributeDesignator DataType="string"
      AttributeId="wsuser" />
  </Apply>
  <Apply FunctionId="string-one-and-only">
    <AttributeSelector RequestContextPath=
      "//attendingPhysician[./patientName/text()=patient]/text()" />
  </Apply>
</Condition>
```

The Condition checks whether the service requester's identity (which is the value of the attribute *wsuser*) is equal to the patient's attending physician. The example is simplified and does not fulfill all of our requirements as the relationship between patient and physician is retrieved from an XML document (e.g., the XACML Request document), while in the given scenario this information is stored in a database. How security policies and databases can be linked is discussed in Section 4.4. For the moment, we put this issue on hold and focus on the pros and cons of this approach:

- (1) As presented in Section 4.2.2, modern declarative policy languages like XACML are well suited for the specification and enforcement of FGAC rules.
- (2) In case the security requirements of database services change, service policies have to be updated and exchanged. Thereby, in case the service-to-database interaction remains unchanged, modifications of the database security schemes as well as the service implementation are not required. Therefore, policy updates are well supported by this approach.
- (3) The implementation is straightforward, as business logic and security functionality are concisely separated. On the database side, only little administration and implementation effort is necessary, as generic database accounts can be used to establish service-to-database connections.
- (4) Security vulnerabilities for the DBMS may arise if database accounts providing more privileges than required with regard to the service functionality are used.

Enforcing security on the services' layer supports fine-grained access control and, moreover, is flexibly adaptable. Nevertheless, security risks for the underlying database systems can arise, in

case the database connection does not fulfill the least privileges paradigm. In other words, the service-to-database interaction then realizes an access corridor which is too relaxed for providing a reliable security configuration.

### **4.3 Security Engineering for Database Web Services – Bridging the Gap**

None of the proposed approaches presented in Section 4.2.3 clearly outperforms the others. Each of them features characteristics that limit their use in practice. The first suggestion of including security checks in the program logic disqualifies itself due to the expected security risks that arise from intermingling security functionality and application logic. The second proposal of shifting access control to the database side implies high administration effort and will thus be impractical for many applications. Furthermore, the possibility to realize FGAC is restricted to the access control capabilities of the employed database system. Fine-grained access control can well be realized by the third approach, which appears to be most preferable. Unfortunately, security leaks for the database systems arise if the service-to-database interaction is run over insecure access corridors. In case security vulnerabilities of the service or the service platform are exploited, information not provided through the service interfaces will also become accessible. Thus, the question remains what a best practice for the specification and enforcement of database service access control could look like.

In the following, we present a pragmatic approach for designing the access control of database Web services. The basic concept is to perform multilayered access control as proposed by the third approach. This allows to meet the application specific security requirements. In order to reduce the discussed security threats for the underlying database systems, access on databases has to be restricted with regard to the service functionality. We do this by generating adequate database accounts based on the service specification, hence, realizing aligned access corridors. We determine the corresponding least required privileges through the analysis of the service-database interactivity. This information is used to semi-automatically prepare the service policies. Through a comparison of database and Web service policies – based on the techniques presented in Chapter 3 – policy compliance is ensured.

#### **4.3.1 Determining the Least Required Privileges**

Before presenting our process model for specifying the access control of database Web services, we first of all give an overview of the security requirements which can automatically be determined: On the one hand, by analyzing the service specification, we are able to generate initial Web service policies and database profiles. On the other hand, access control configurations of database systems can be preprocessed to enable policy compliance checks.

According to our definition, database backed Web services are services that strongly depend on the data processing capabilities of underlying database systems. Thus, service methods are closely related to database operations, e.g., given as XQuery / XPath statements which are exe-

cuted on XML databases or SQL queries which are run on relational database systems. Instead of writing service policies from scratch, security policies that follow the principle of least privilege can be generated by preprocessing query statements.

In the following, we describe how access control information can be extracted from SQL statements. We use an attribute grammar<sup>3</sup> that represents an extract of the SQL grammar<sup>4</sup> to formalize our approach. Thereby, we focus on manipulative queries, i.e., selections, updates, insertions, deletions, and function calls / stored procedure calls, which we consider to be the type of queries that are mainly used by database services. Privileges required to execute data definition statements (like create table statements) can be determined in an analogous way. Our attribute grammar provides the following characteristics:

- Select, insert, and update privileges are determined on the granularity of columns.
- Delete privileges are determined on the granularity of tables.
- Execute privileges refer to functions and procedures.

For the sake of simplicity, the presented grammar handles qualified attribute names, i.e., columns are specified in the form `table-alias.column` and database objects belong to the same schema. Otherwise, schema information has to be provided or retrieved in a preprocessing step.

When specifying semantic rules, recurrences of nonterminals on the right hand side of a production are numbered. For example, given a production  $P : T \rightarrow xTyT$  and a synthesized attribute  $a$  of the nonterminal  $T$ , we can define the following semantic rule:  $T.a = T_1.a + T_2.a$ . This rule denotes that  $T.a$  is determined through the sum of the  $a$  values of  $T$ -occurrences on the right hand side of  $P$ . This notation corresponds to the one used by Aho et al. (1986). Our attribute grammar includes the following attributes:

**privs** is a synthesized attribute representing a set of privileges, i.e., tuples  $(O, A)$ . In other words,  $T.privs$  consists of all privileges required to execute the operations that are inferred from the nonterminal  $T$ .

**tids** is a synthesized attribute consisting of (table-alias, table-name)-tuples.

**lexval** is a synthesized attribute representing the lexical value of nonterminals.

**act** is an inherited attribute specifying the action (i.e., select, insert, update, delete, or execute) that is specified by the current context.

**env** is an inherited attribute consisting of the (table-alias, table-name) pairs that are in the current scope of the nonterminal.

Table 4.1 presents the attribute grammar for manipulative SQL query statements. The start symbol of our attribute grammar is `SQL_LISTS`. Thus, the least required privileges of manipulative SQL statements are specified by `SQL_LIST.privs`.

<sup>3</sup>Confer [Kühnemann and Vogler (1997)].

<sup>4</sup>See, for example, [Date and Darwen (1997)].

*Remark:* In order to shorten the specification of the attribute grammar, we omitted the declaration of semantic rules for lexical attributes. The values of lexical attributes are determined by the lexical fields that are deduced from the corresponding nonterminals.

#### 4.3.1.1 Example

Assume that due to a bonus program every patient with private insurance shall be treated by the chief physician Jeffrey Geiger. Based on the database schema illustrated in Figure 4.2, this adaptation is realized by the following SQL statement:

```
update MedicalRecords m
set m.AttendingPhysician = "Jeffrey Geiger"
where m.Patient in (select p.Name
                   from Patients p
                   where p.HealthInsurance = "Private HI");
```

Figure 4.6 shows the parse tree for this query statement. The dependency graph in Figure 4.7 illustrates the evaluation of the attribute values. Inherited attributes are propagated top-down. They represent context information like table aliases (*env*) and operation types (*act*). The following abbreviations are used in the figure:

$$m = (m, \text{MedicalRecords}), \quad p = (p, \text{Patient}),$$

$$s = \text{select}, \text{ and } u = \text{update}$$

Synthesized attributes are evaluated in a bottom-up manner. The least required privileges to perform the update are specified by the *privs*-value of the topmost *SQL\_LIST* node. In our example, four privileges are required, namely:

$$P_1 = ((\text{table} = \text{MedicalRecords}, \text{column} = \text{AttendingPhysician}), \text{action} = \text{update})$$

$$P_2 = ((\text{table} = \text{MedicalRecords}, \text{column} = \text{Patient}), \text{action} = \text{select})$$

$$P_3 = ((\text{table} = \text{Patients}, \text{column} = \text{Name}), \text{action} = \text{select})$$

$$P_4 = ((\text{table} = \text{Patients}, \text{column} = \text{HealthInsurance}), \text{action} = \text{select})$$

Productions	Semantic Rules
P <sub>1</sub> : SQL_LIST → SQL';' P <sub>2</sub> : SQL_LIST → SQL_LIST SQL';'	SR <sub>1</sub> : SQL_LIST.privs = SQL.privs SR <sub>2</sub> : SQL_LIST.privs = SQL_LIST <sub>1</sub> .privs ∪ SQL.privs
P <sub>3</sub> : SQL → QUERY_STMT P <sub>4</sub> : SQL → UPDATE_STMT P <sub>5</sub> : SQL → INSERT_STMT P <sub>6</sub> : SQL → DELETE_STMT P <sub>7</sub> : SQL → CALL_STMT	SR <sub>3</sub> : SQL.privs = QUERY_STMT.privs SR <sub>4</sub> : SQL.privs = UPDATE_STMT.privs SR <sub>5</sub> : SQL.privs = INSERT_STMT.privs SR <sub>6</sub> : SQL.privs = DELETE_STMT.privs SR <sub>7</sub> : SQL.privs = CALL_STMT.privs
P <sub>8</sub> : QUERY_STMT → QUERY_TERM P <sub>9</sub> : QUERY_STMT → QUERY_STMT ('union'   'union all'   'intersect'   'except') QUERY_TERM P <sub>10</sub> : QUERY_TERM → QUERY_SPEC P <sub>11</sub> : QUERY_TERM → '(' QUERY_SPEC ')' P <sub>12</sub> : QUERY_SPEC → 'select' ('all'   'distinct')? SCALAR_EXP_LIST TABLE_EXP	SR <sub>8</sub> : QUERY_STMT.privs = QUERY_TERM.privs SR <sub>9</sub> : QUERY_STMT.privs = QUERY_STMT <sub>1</sub> .privs ∪ QUERY_TERM.privs SR <sub>10</sub> : QUERY_TERM.privs = QUERY_SPEC.privs SR <sub>11</sub> : QUERY_TERM.privs = QUERY_SPEC.privs SR <sub>12,1</sub> : QUERY_SPEC.privs = SCALAR_EXP_LIST.privs ∪ TABLE_EXP.privs SR <sub>12,2</sub> : SCALAR_EXP_LIST.env = TABLE_EXP.tids ∪ QUERY_SPEC.env SR <sub>12,3</sub> : TABLE_EXP.env = QUERY_SPEC.env
P <sub>13</sub> : TABLE_EXP → 'from' TABLE_REF_LIST (WHERE)? (GROUP_BY)? (HAVING)? (ORDER_BY)?  P <sub>14</sub> : TABLE_REF_LIST → TABLE_REF  P <sub>15</sub> : TABLE_REF_LIST → TABLE_REF_LIST ',' TABLE_REF	SR <sub>13,1</sub> : TABLE_EXP.privs = TABLE_REF_LIST.privs ∪ WHERE.privs ∪ HAVING.privs SR <sub>13,2</sub> : TABLE_EXP.tids = TABLE_REF_LIST.tids SR <sub>13,3</sub> : WHERE.env = TABLE_REF_LIST.tids ∪ TABLE_EXP.env SR <sub>13,4</sub> : HAVING.env = TABLE_REF_LIST.tids ∪ TABLE_EXP.env SR <sub>13,5</sub> : TABLE_REF_LIST.env = TABLE_EXP.env SR <sub>14,1</sub> : TABLE_REF_LIST.privs = TABLE_REF.privs SR <sub>14,2</sub> : TABLE_REF_LIST.tids = TABLE_REF.tids SR <sub>14,3</sub> : TABLE_REF.env = TABLE_REF_LIST.env SR <sub>15,1</sub> : TABLE_REF_LIST.privs = TABLE_REF_LIST <sub>1</sub> .privs ∪ TABLE_REF.privs SR <sub>15,2</sub> : TABLE_REF_LIST.tids = TABLE_REF_LIST <sub>1</sub> .tids ∪

Productions	Semantic Rules
<p>P<sub>16</sub>: TABLE_REF → TABLE</p> <p>P<sub>17</sub>: TABLE_REF → '(' QUERY_STMT ')' NAME</p> <p>P<sub>18</sub>: TABLE → NAME ('as')? NAME</p>	<p>TABLE_REF.tids</p> <p>SR<sub>15,3</sub>: TABLE_REF.env = TABLE_REF_LIST.env</p> <p>SR<sub>15,4</sub>: TABLE_REF_LIST<sub>1</sub>.env = TABLE_REF_LIST.env</p> <p>SR<sub>16</sub>: TABLE_REF.tids = TABLE.tids</p> <p>SR<sub>17</sub>: TABLE_REF.privs = QUERY_STMT.privs</p> <p>SR<sub>17,1</sub>: QUERY_STMT.env = TABLE_REF.env</p> <p>SR<sub>18,1</sub>: TABLE.tids = {(NAME<sub>2</sub>.lexval, NAME<sub>1</sub>.lexval)}</p>
<p>P<sub>19</sub>: WHERE → 'where' SEARCH_CND</p> <p>P<sub>20</sub>: SEARCH_CND → SEARCH_CND ('and'   'or') SEARCH_CND,</p> <p>P<sub>21</sub>: SEARCH_CND → 'not' SEARCH_CND</p> <p>P<sub>22</sub>: SEARCH_CND → '(' SEARCH_CND ')'</p> <p>P<sub>23</sub>: SEARCH_CND → PREDICATE</p>	<p>SR<sub>19,1</sub>: WHERE.privs = SEARCH_CND.privs</p> <p>SR<sub>19,2</sub>: SEARCH_CND.env = WHERE.env</p> <p>SR<sub>20,1</sub>: SEARCH_CND.privs = SEARCH_CND<sub>1</sub>.privs ∪ SEARCH_CND<sub>2</sub>.privs</p> <p>SR<sub>20,2</sub>: SEARCH_CND<sub>1</sub>.env = SEARCH_CND.env</p> <p>SR<sub>20,3</sub>: SEARCH_CND<sub>2</sub>.env = SEARCH_CND.env</p> <p>SR<sub>21,1</sub>: SEARCH_CND.privs = SEARCH_CND<sub>1</sub>.privs</p> <p>SR<sub>21,2</sub>: SEARCH_CND<sub>1</sub>.env = SEARCH_CND.env</p> <p>SR<sub>22,1</sub>: SEARCH_CND.privs = SEARCH_CND<sub>1</sub>.privs</p> <p>SR<sub>22,2</sub>: SEARCH_CND<sub>1</sub>.env = SEARCH_CND.env</p> <p>SR<sub>23,1</sub>: SEARCH_CND.privs = PREDICATE.privs</p> <p>SR<sub>23,2</sub>: PREDICATE.env = SEARCH_CND.env</p>
<p>P<sub>24</sub>: PREDICATE → SCALAR_EXP COMPARISON SCALAR_EXP</p> <p>P<sub>25</sub>: PREDICATE → SCALAR_EXP COMPARISON '(' QUERY_SPEC ')'</p> <p>P<sub>26</sub>: PREDICATE → SCALAR_EXP ('not')? 'between' SCALAR_EXP 'and' SCALAR_EXP</p>	<p>SR<sub>24,1</sub>: PREDICATE.privs = SCALAR_EXP<sub>1</sub>.privs ∪ SCALAR_EXP<sub>2</sub>.privs</p> <p>SR<sub>24,2</sub>: SCALAR_EXP<sub>1</sub>.env = PREDICATE.env</p> <p>SR<sub>24,3</sub>: SCALAR_EXP<sub>2</sub>.env = PREDICATE.env</p> <p>SR<sub>25,1</sub>: PREDICATE.privs = SCALAR_EXP.privs ∪ QUERY_SPEC.privs</p> <p>SR<sub>25,2</sub>: SCALAR_EXP.env = PREDICATE.env</p> <p>SR<sub>25,3</sub>: QUERY_SPEC.env = PREDICATE.env</p> <p>SR<sub>26,1</sub>: PREDICATE.privs = SCALAR_EXP<sub>1</sub>.privs ∪ SCALAR_EXP<sub>2</sub>.privs ∪ SCALAR_EXP<sub>3</sub>.privs</p> <p>SR<sub>26,2</sub>: SCALAR_EXP<sub>1</sub>.env = PREDICATE.env</p> <p>SR<sub>26,3</sub>: SCALAR_EXP<sub>2</sub>.env = PREDICATE.env</p> <p>SR<sub>26,4</sub>: SCALAR_EXP<sub>3</sub>.env = PREDICATE.env</p>



Productions	Semantic Rules
P <sub>27</sub> : PREDICATE → SCALAR_EXP ('not')? 'like' SCALAR_EXP	SR <sub>27,1</sub> : PREDICATE.privs = SCALAR_EXP <sub>1</sub> .privs ∪ SCALAR_EXP <sub>2</sub> .privs SR <sub>27,2</sub> : SCALAR_EXP <sub>1</sub> .env = PREDICATE.env SR <sub>27,3</sub> : SCALAR_EXP <sub>2</sub> .env = PREDICATE.env
P <sub>28</sub> : PREDICATE → COLUMN_REF 'is' ('not')? 'null'	SR <sub>28,1</sub> : PREDICATE.privs = COLUMN_REF.privs SR <sub>28,2</sub> : COLUMN_REF.act = <i>select</i> SR <sub>28,3</sub> : COLUMN_REF.env = PREDICATE.env
P <sub>29</sub> : PREDICATE → SCALAR_EXP ('not')? 'in' '(' QUERY_SPEC ')'	SR <sub>29,1</sub> : PREDICATE.privs = SCALAR_EXP.privs ∪ QUERY_SPEC.privs SR <sub>29,2</sub> : SCALAR_EXP.env = PREDICATE.env SR <sub>29,3</sub> : QUERY_SPEC.env = PREDICATE.env
P <sub>30</sub> : PREDICATE → SCALAR_EXP ('not')? 'in' '(' SCALAR_EXP_LIST ')'	SR <sub>30,1</sub> : PREDICATE.privs = SCALAR_EXP.privs ∪ SCALAR_EXP_LIST.privs SR <sub>30,2</sub> : SCALAR_EXP.env = PREDICATE.env SR <sub>30,3</sub> : SCALAR_EXP_LIST.env = PREDICATE.env
P <sub>31</sub> : PREDICATE → ('not')? 'exists' '(' QUERY_SPEC ')'	SR <sub>31,1</sub> : PREDICATE.privs = QUERY_SPEC.privs SR <sub>31,2</sub> : QUERY_SPEC.env = PREDICATE.env
P <sub>32</sub> : PREDICATE → SCALAR_EXP	SR <sub>32,1</sub> : PREDICATE.privs = SCALAR_EXP.privs SR <sub>32,2</sub> : SCALAR_EXP.env = PREDICATE.env
P <sub>33</sub> : SCALAR_EXP → SCALAR_EXP ('+'   '-'   '*'   '/') SCALAR_EXP	SR <sub>33,1</sub> : SCALAR_EXP.privs = SCALAR_EXP <sub>1</sub> .privs ∪ SCALAR_EXP <sub>2</sub> .privs SR <sub>33,2</sub> : SCALAR_EXP <sub>1</sub> .env = SCALAR_EXP.env SR <sub>33,3</sub> : SCALAR_EXP <sub>2</sub> .env = SCALAR_EXP.env
P <sub>34</sub> : SCALAR_EXP → COLUMN_REF	SR <sub>34,1</sub> : SCALAR_EXP.privs = COLUMN_REF.privs SR <sub>34,2</sub> : COLUMN_REF.act = <i>select</i> SR <sub>34,3</sub> : COLUMN_REF.env = SCALAR_EXP.env
P <sub>35</sub> : SCALAR_EXP → LITERAL	
P <sub>36</sub> : SCALAR_EXP → '(' SCALAR_EXP_LIST ')'	SR <sub>36,1</sub> : SCALAR_EXP.privs = SCALAR_EXP_LIST.privs SR <sub>36,2</sub> : SCALAR_EXP_LIST.env = SCALAR_EXP.env
P <sub>37</sub> : SCALAR_EXP → SCALAR_EXP 'as' NAME	SR <sub>37,1</sub> : SCALAR_EXP.privs = SCALAR_EXP <sub>1</sub> .privs SR <sub>37,2</sub> : SCALAR_EXP <sub>1</sub> .env = SCALAR_EXP.env
P <sub>38</sub> : SCALAR_EXP → FUNCTION_CALL	SR <sub>38,1</sub> : SCALAR_EXP.privs = FUNCTION_CALL.privs

Productions	Semantic Rules
<p>P<sub>39</sub>: SCALAR_EXP_LIST → SCALAR_EXP</p> <p>P<sub>40</sub>: SCALAR_EXP_LIST → SCALAR_EXP_LIST ', SCALAR_EXP</p>	<p>SR<sub>38,2</sub>: FUNCTION_CALL.env = SCALAR_EXP.env</p> <p>SR<sub>39,1</sub>: SCALAR_EXP_LIST.privs = SCALAR_EXP.privs</p> <p>SR<sub>39,2</sub>: SCALAR_EXP.env = SCALAR_EXP_LIST.env</p> <p>SR<sub>40,1</sub>: SCALAR_EXP_LIST.privs = SCALAR_EXP_LIST<sub>1</sub>.privs ∪ SCALAR_EXP.privs</p> <p>SR<sub>40,2</sub>: SCALAR_EXP_LIST<sub>1</sub>.env = SCALAR_EXP_LIST.env</p> <p>SR<sub>40,3</sub>: SCALAR_EXP.env = SCALAR_EXP_LIST.env</p>
<p>P<sub>41</sub>: UPDATE_STMT → 'update' TABLE 'set' ASSIGN_LIST (WHERE)?</p> <p>P<sub>42</sub>: ASSIGN_LIST → COLUMN '=' SCALAR_EXP</p> <p>P<sub>43</sub>: ASSIGN_LIST → ASSIGN_LIST ', COLUMN '=' SCALAR_EXP</p>	<p>SR<sub>41,1</sub>: UPDATE_STMT.privs = ASSIGN_LIST.privs ∪ WHERE.privs</p> <p>SR<sub>41,2</sub>: ASSIGN_LIST.env = TABLE.tids</p> <p>SR<sub>41,3</sub>: WHERE.env = TABLE.tids</p> <p>SR<sub>42,1</sub>: ASSIGN_LIST.privs = COLUMN.privs ∪ SCALAR_EXP.privs</p> <p>SR<sub>42,2</sub>: COLUMN.act = <i>update</i></p> <p>SR<sub>42,3</sub>: COLUMN.env = ASSIGN_LIST.env</p> <p>SR<sub>42,4</sub>: SCALAR_EXP.env = ASSIGN_LIST.env</p> <p>SR<sub>43,1</sub>: ASSIGN_LIST.privs = ASSIGN_LIST<sub>1</sub>.privs ∪ COLUMN.privs ∪ SCALAR_EXP.privs</p> <p>SR<sub>43,2</sub>: COLUMN.act = <i>update</i></p> <p>SR<sub>43,3</sub>: ASSIGN_LIST<sub>1</sub>.env = ASSIGN_LIST.env</p> <p>SR<sub>43,4</sub>: COLUMN.env = ASSIGN_LIST.env</p> <p>SR<sub>43,5</sub>: SCALAR_EXP.env = ASSIGN_LIST.env</p>
<p>P<sub>44</sub>: INSERT_STMT → 'insert into' TABLE '(' COLUMN_LIST ')' VAL_OR_QUERY,</p> <p>P<sub>45</sub>: COLUMN_LIST → COLUMN</p> <p>P<sub>46</sub>: COLUMN_LIST → COLUMN_LIST ', ' COLUMN</p>	<p>SR<sub>44,1</sub>: INSERT_STMT.privs = COLUMN_LIST.privs ∪ VAL_OR_QUERY.privs</p> <p>SR<sub>44,2</sub>: COLUMN_LIST.env = TABLE.tids</p> <p>SR<sub>44,3</sub>: VAL_OR_QUERY.env = TABLE.tids</p> <p>SR<sub>45,1</sub>: COLUMN_LIST.privs = COLUMN.privs</p> <p>SR<sub>45,2</sub>: COLUMN.act = <i>insert</i></p> <p>SR<sub>45,3</sub>: COLUMN.env = COLUMN_LIST.env</p> <p>SR<sub>46,1</sub>: COLUMN_LIST.privs = COLUMN_LIST<sub>1</sub>.privs ∪ COLUMN.privs</p> <p>SR<sub>46,2</sub>: COLUMN.act = <i>insert</i></p>

Productions	Semantic Rules
<p>P<sub>47</sub>: VAL_OR_QUERY → 'values' '(' SCALAR_EXP_LIST ')'</p> <p>P<sub>48</sub>: VAL_OR_QUERY → QUERY_SPEC</p>	<p>SR<sub>46,3</sub>: COLUMN_LIST<sub>1</sub>.env = COLUMN_LIST.env</p> <p>SR<sub>46,4</sub>: COLUMN.env = COLUMN_LIST.env</p> <p>SR<sub>47,1</sub>: VAL_OR_QUERY.privs = SCALAR_EXP_LIST.privs</p> <p>SR<sub>47,2</sub>: SCALAR_EXP_LIST.env = VAL_OR_QUERY.env</p> <p>SR<sub>48,1</sub>: VAL_OR_QUERY.privs = QUERY_SPEC.privs</p> <p>SR<sub>48,2</sub>: QUERY_SPEC.env = VAL_OR_QUERY.env</p>
<p>P<sub>49</sub>: DELETE_STATEMENT → 'delete from' TABLE (WHERE)?</p>	<p>SR<sub>49,1</sub>: DELETE_STATEMENT.privs = WHERE.privs ∪ {(table = <i>t</i>, action = <i>delete</i>)   <i>t</i> ∈ TABLE.tids}</p> <p>SR<sub>49,2</sub>: WHERE.env = TABLE.tids</p>
<p>P<sub>50</sub>: CALL_STATEMENT → 'call' FUNCTION_CALL</p> <p>P<sub>51</sub>: FUNCTION_CALL → NAME '(' (SCALAR_EXP_LIST)? ')'</p>	<p>SR<sub>50</sub>: CALL_STATEMENT.privs = FUNCTION_CALL.privs</p> <p>SR<sub>51,1</sub>: FUNCTION_CALL.privs = {(procedure = NAME.lexval, action = <i>execute</i>)} ∪ SCALAR_EXP_LIST.privs</p> <p>SR<sub>51,2</sub>: SCALAR_EXP_LIST.env = FUNCTION_CALL.env</p>
<p>P<sub>52</sub>: COLUMN_REF → COLUMN</p> <p>P<sub>53</sub>: COLUMN_REF → NAME '.*'</p> <p>P<sub>54</sub>: COLUMN → NAME '.' NAME</p>	<p>SR<sub>52,1</sub>: COLUMN_REF.privs = COLUMN.privs</p> <p>SR<sub>52,2</sub>: COLUMN.act = COLUMN_REF.act</p> <p>SR<sub>52,3</sub>: COLUMN.env = COLUMN_REF.env</p> <p>SR<sub>53</sub>: COLUMN_REF.privs = {(table = <i>t</i>, action = <i>a</i>)   (NAME.lexval, <i>t</i>) ∈ COLUMN_REF.env ∧ <i>a</i> = COLUMN_REF.act}</p> <p>SR<sub>54</sub>: COLUMN.privs = {((table = <i>t</i>, column = NAME<sub>2</sub>.lexval), action = <i>a</i>)   (NAME<sub>1</sub>.lexval, <i>t</i>) ∈ COLUMN.env ∧ <i>a</i> = COLUMN.act}</p>
<p>P<sub>55</sub>: ORDER_BY → 'order by' ORDER_LIST</p> <p>P<sub>56</sub>: GROUP_BY → 'group by' GROUP_LIST</p> <p>P<sub>57</sub>: HAVING → 'having' SEARCH_CND</p> <p>P<sub>58</sub>: ORDER_LIST → COLUMN_REF ('asc'   'dsc')?</p> <p>P<sub>59</sub>: ORDER_LIST → ORDER_LIST ',' COLUMN_REF ('asc'   'dsc')?</p> <p>P<sub>60</sub>: GROUP_LIST → COLUMN_REF</p> <p>P<sub>61</sub>: GROUP_LIST → GROUP_LIST ',' COLUMN_REF</p> <p>P<sub>62</sub>: LITERAL → (STRING   INTEGER   FLOAT   'null')</p>	<p>SR<sub>57,1</sub>: HAVING.privs = SEARCH_CND.privs</p> <p>SR<sub>57,2</sub>: SEARCH_CND.env = HAVING.env</p>

Productions	Semantic Rules
P <sub>63</sub> : NAME → [ <sup>'a'</sup> - <sup>'z'</sup> <sup>'A'</sup> - <sup>'Z'</sup> ] <sup>+</sup> ( [ <sup>'a'</sup> - <sup>'z'</sup> <sup>'A'</sup> - <sup>'Z'</sup> ] <sup>+</sup>   NUMBER ) <sup>*</sup> P <sub>64</sub> : STRING → <sup>'"</sup> ( [ <sup>'a'</sup> - <sup>'z'</sup> <sup>'A'</sup> - <sup>'Z'</sup> ] <sup>+</sup>   <sup>' '</sup> ) <sup>'"</sup> P <sub>65</sub> : INTEGER → ( [ <sup>'+'</sup> , <sup>'-'</sup> ] ) <sup>?</sup> NUMBER P <sub>66</sub> : NUMBER → ( <sup>'0'</sup>   [ <sup>'1'</sup> - <sup>'9'</sup> ] <sup>[<sup>'0'</sup>-<sup>'9'</sup>]<sup>*</sup> )            P<sub>67</sub>: FLOAT → ( [<sup>'+'</sup>,<sup>'-'</sup> ] )<sup>?</sup> NUMBER <sup>'.'</sup> [<sup>'0'</sup>-<sup>'9'</sup>]<sup>+</sup>            P<sub>68</sub>: COMPARISON → <sup>'&lt;'</sup>   <sup>'&lt;='</sup>   <sup>'='</sup>   <sup>'&gt;='</sup>   <sup>'&gt;'</sup>   <sup>'&lt;&gt;'</sup> </sup>	

Table 4.1: Attribute grammar for extracting access control information from manipulative SQL queries



Figure 4.6: Parse tree

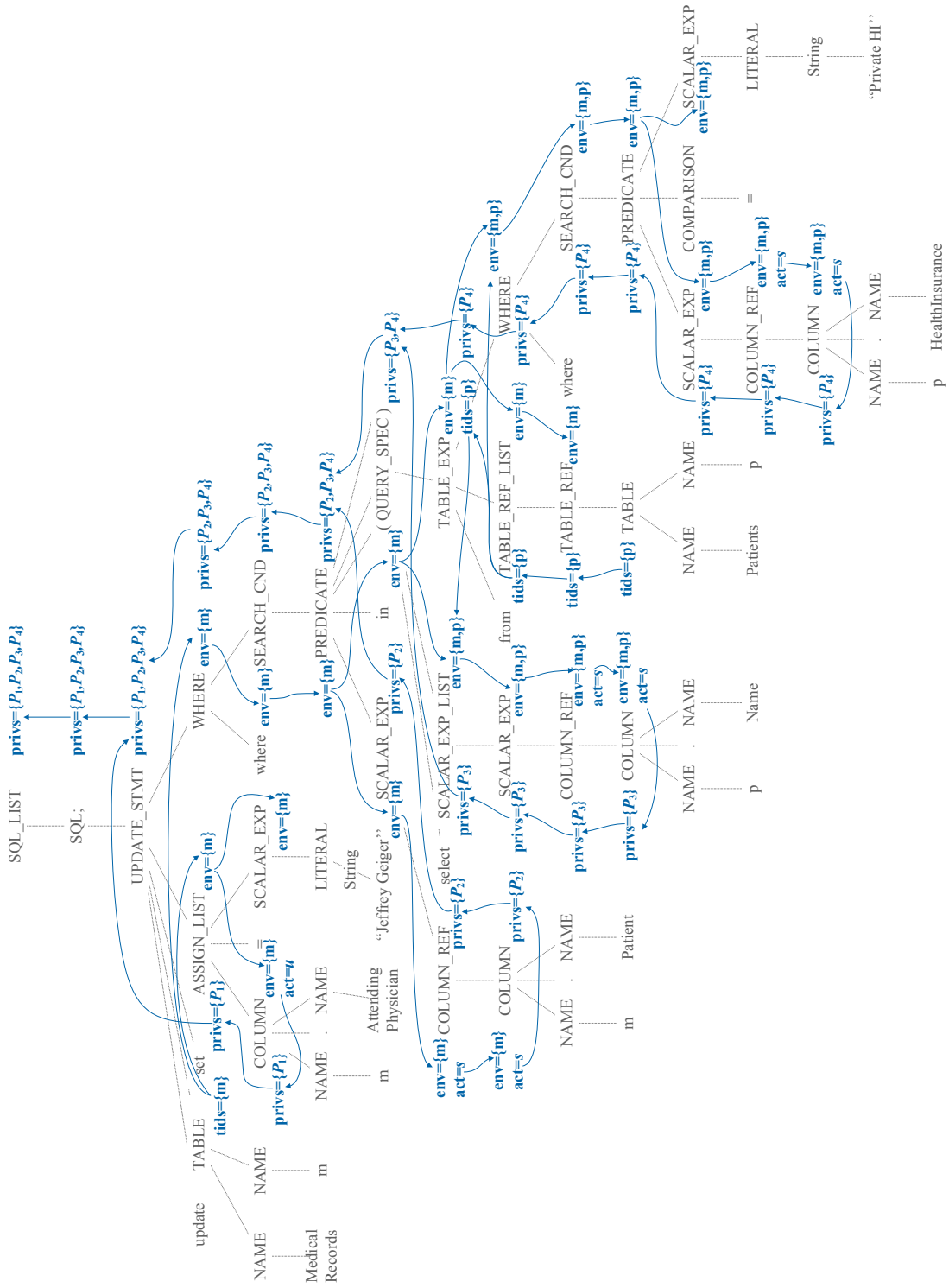


Figure 4.7: Dependency graph

### 4.3.2 Automated Policy Generation

The extracted access control information is used to generate XACML policies. In particular, database objects like tables and columns constitute the Resources of a Rule's Target, while operations like selections and updates represent the Actions. Besides policies, XACML requests and corresponding security pointcuts can be generated. Service developers can use the generated code and policies as starting basis for implementing the security functionality of database services. For example, by including subject specifications and additional constraints, security policies can be refined and adjusted.

Furthermore, the determined least required privileges can be used to specify adequate database user profiles by generating corresponding database specific `grant` statements (see Section 4.2.1). The extracted privileges represent the least required “vertical” access control rights, i.e., select, update, and insert privileges on individual columns of tables. Our approach can be extended to also determine statically defined “horizontal” access control restrictions, i.e., restrictions on rows of tables. In the above example, read access on the Patients table can be restricted to rows whose value of the attribute HealthInsurance is “Private HI”. Thus, access on tables can be restricted via adequate views. Vertical and horizontal access control information in combination allow to realize optimized access corridors for database services.

Web service policies and database profiles are determined through a static analysis of the service-to-database interaction. Using dynamic SQL, SQL statements can be constructed and executed at runtime.<sup>5</sup> In cases when nothing is known about the SQL statements, e.g., programs providing access to arbitrary tables of a database, this approach is not applicable. Obviously, the least required privileges are not known in such a scenario and the traditional approach has to be applied, granting the union of privileges for all possible types of supported queries. Nevertheless, a static analysis can be performed when the input and the output of statements are known beforehand and only parameter values are set during program execution (like the names of patients when querying their medical records).

As stated above, the extracted least required privileges allow the realization of optimized access corridors for database services. In order to improve performance, the repeated setup and termination of database connections is circumvented by connection pooling, i.e., applications can request ready-to-use connections of such pools and return them after use. Such connection pools can be realized for each application, i.e., each application can have its own pool of connections that follow the principle of least privilege. Some application servers like, for instance, BEA WebLogic<sup>6</sup>, also support applications to be deployed side-by-side within the same application server process, thus, sharing the same connection pools. In general, following this approach, the principle of least privilege is given up from the point of view of one application.

In order to minimize security risks, connection pools should only be shared by applications that demand similar authorizations. For instance, administrative applications should share other pools than those used by data processing services. The generation of service clusters demands for a metric to compare privileges. Measuring the dissimilarity of privileges  $P_1$  and  $P_2$  can, for

---

<sup>5</sup>See [Cannan and Otten (1993)].

<sup>6</sup>See the BEA WebLogic Server and WebLogic Express manual, version 8.1 (2006).

example, be done by use of the subtraction operator, e.g., determining  $\delta(P_1, P_2) = (P_1 - P_2) \cup (P_2 - P_1)$ . Thereby, privileges in  $\delta(P_1, P_2)$  should be weighted depending on whether the misuse of them is expected to cause some, serious or grave damage. The exact formal definition of such a metric is beyond the scope of this thesis, but, nevertheless, represents an interesting direction for future research.

### 4.3.3 Extraction of Database Policies

How access rules can be extracted from a DBMS depends on the employed database system. Taking Oracle Database as an example, privileges are stored in data dictionary views like `user_tab_privs`. In order to provide a clear manageable policy structure, we generate policies of the following categories:

**Permission policies:** The rules of permission policies are representing individual privileges. That is, resource and action declarations of such rules specify database objects and the way they can be accessed. Section C.1 illustrates the representation of formal privilege specifications in XACML. Targets of such policies are not restricted, using `AnySubject`, `AnyResource`, and `AnyAction` as presented in Section 4.2.2. Thus, permission policies (respectively the privileges they represent) can be granted to arbitrary users and/or roles. This is done by means of base policies.

**Base policies:** Policies of this category assign privileges to users or roles. The grantees' attributes are included in the subject specification of the Target. The granted privileges are referenced by permission policies which are included by the use of `PolicyIdReference` (cf. the XACML specification, [Moses (2005)]). Section C.2 describes the XACML layout of base policies.

**Role assignment policies:** This type of policy is used to assign roles to users or senior roles, thus, supporting hierarchical RBAC. Following the recommendation of Anderson (2005), policies are generated consisting of rules whose target is composed as follows: The granted role constitutes the Resource, while the grantee (any subject specification or senior role) is given by the Subject. The Target's Action is *enable* which is the fixed keyword used for role assignments. In Section C.3, we show the XACML representation of formal role assignments.

Referring to the policy model illustrated in Figure 3.2 on page 21, permission policies correspond to privilege definitions (PD) and privilege assignments (PA) are represented by base policies. Finally, role assignments (RA) and role hierarchies (RH) are expressed by role assignment policies.

### 4.3.4 Engineering Adaptable Access Control Policies

The reliable definition of service policies can proceed from two different directions. On the one hand, initial service policies can be defined based on the authorization settings of the database account used to establish the service-to-database connections. These can later on be refined.



On the other hand, service policies can be generated based on the specification of the service's functionality.

#### 4.3.4.1 Adapting Database Policies

This approach is particularly useful for revising the access control of existing database services, i.e., scenarios when service and database interaction are already implemented. Then, the privileges granted to the database accounts used for interacting with the database can be extracted and preprocessed as XACML policies. These define the maximum set of privileges that can be granted to the Web service clients. The privileges are stored in permission policies as described above. The Web service developer is then able to select those permission policies that apply to the respective service methods and use them for defining the service's policy. The privileges to execute service methods are assigned to Web service users via base policies. Thereby, the developer is able to use the FGAC capabilities of XACML.

#### 4.3.4.2 Defining Policies – Starting from the Service's Point of View

Figure 4.8 illustrates the process of defining the access control of database services when starting from scratch. Web service interfaces are typically defined by means of WSDL. As database services interact with one or more underlying databases, service operations are closely related to database queries, e.g., formulated in XQuery, XPath or SQL. If the queries are included in the service specification, the least required privileges can be extracted automatically (step 2) as shown in Section 4.3.1. These can be employed to specify a database profile that is tailored to the service functionality (step 3a). Alternatively, the database connection can be predefined (step 3b). In any case, the database account's authorizations can be extracted automatically (step 4), which will later be used for compliance checks with the service policies. The right branch of the process diagram illustrates the specification of initial versions of service policies based on the extracted authorization information (step 5). These policies can manually be refined (step 6), e.g., by restricting the service policy through additional conditions and subject declarations.

Now that the access control policies of the service as well as the policies for the employed database account are available, a compliance check (step 7) can be performed. It is checked if the database policies cover the authorization requirements of the service, i.e., whether the database policies are a relaxation of the service policies (cf. Section 3.2.3.4).

- In case some service privileges are not relaxed by the database policy, the service is inoperable denoting that some of the queries are not executable. Thus, the service-to-database connection has to be adjusted, e.g., by realizing the account proposed in step 3a.
- If the service privileges are subsumed by the database policy, but the database policies do not follow the principle of least privilege, the database profile should be confined in order to avoid security vulnerabilities. Again, this can be achieved by performing step 3a.
- If the database policies already fulfill the least required privileges principle, the access control configuration is optimal from the databases' security viewpoint.

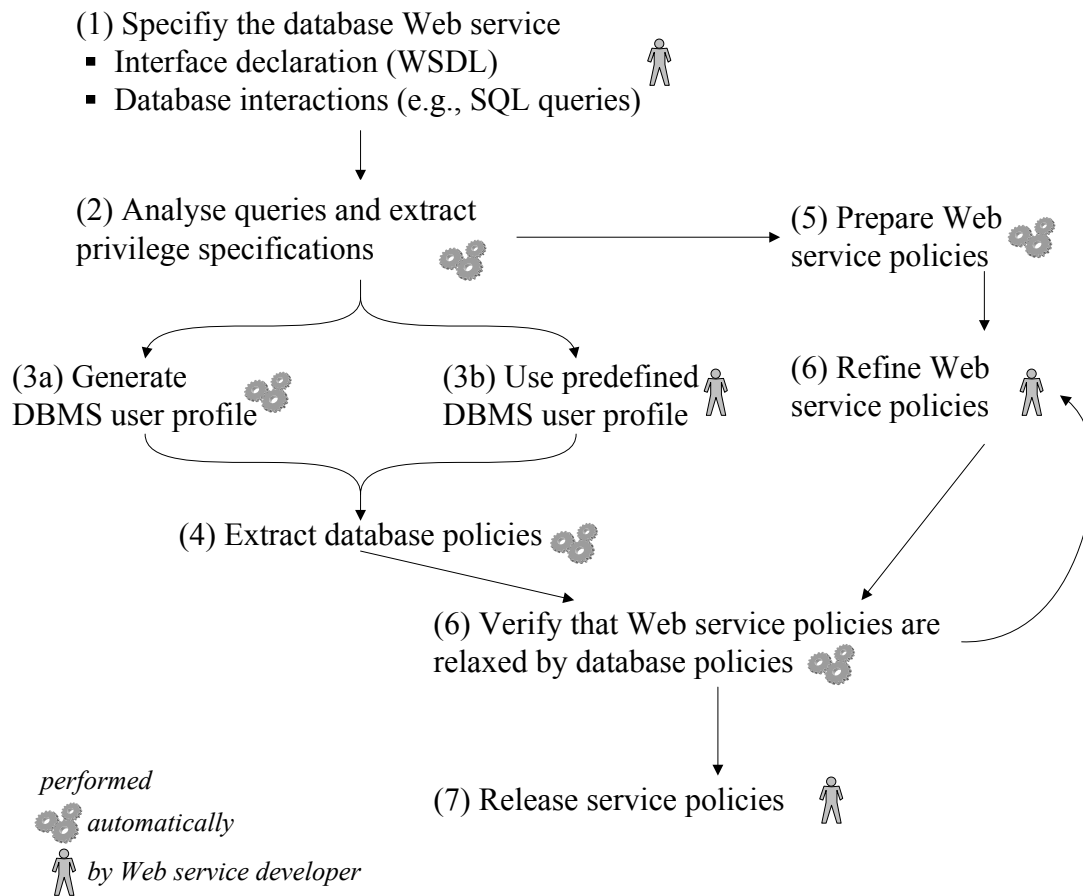


Figure 4.8: Semi-automatic security engineering for database Web services

In case compliance is achieved, the service policies can be released. As illustrated in the diagram, the refinement process can be repeated iteratively and additional application specific rules can be implemented. Each modification demands for a revalidation of policy compliance.

The benefits of the presented methodical approach are that security risks for the underlying database systems can be averted and fine-grained access control concepts can be realized. As illustrated in the figure, many of the steps can be performed automatically. That is, the development process can be supported by tools as presented in Section 4.4, easing the implementation of database services.

#### 4.3.4.3 Adaptive Policy Management

Web service environments can be highly dynamic. For example, dynamic service compositions allow the replacement of sub-activities. Moreover, also the group of authorized users can change over time. For example, consider services which are only executable for software engineers during the development phase before being released publicly. As another example, suppose a Web

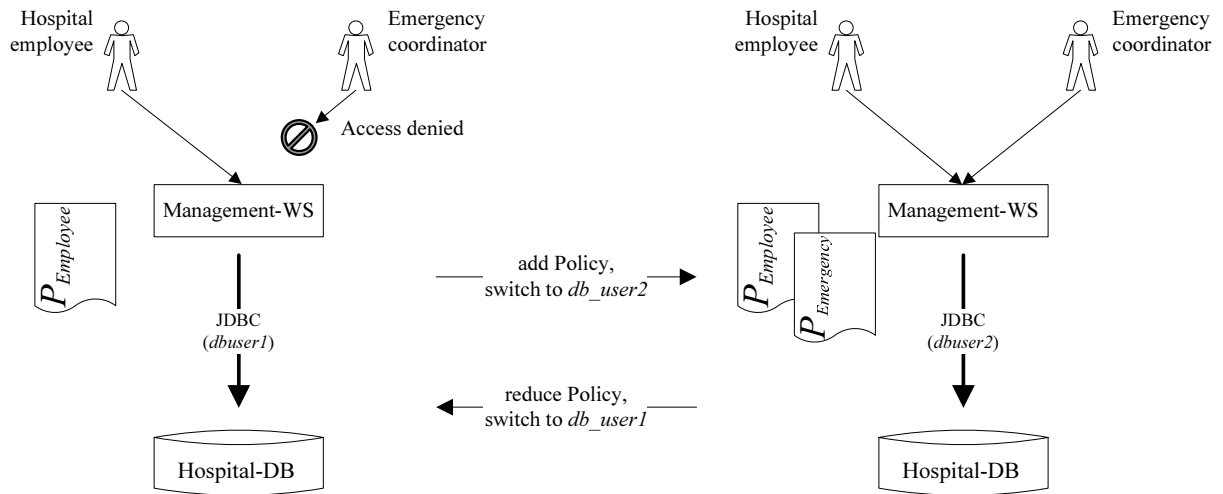


Figure 4.9: Adaptive policy management

service called *Management-WS* which allows administrative employees to manage the allocation of beds and the assignments of attending physicians (i.e., the service provides two methods for these functionalities). As shown in Figure 4.9, the *Employee Policy* regulates authorizations for hospital employees. Connections to the hospital's database system are established via the account of  $dbuser1$ .

Under certain circumstances, like emergency situations, designated experts might require gathering information about human resources and free capacities from institutions like police and fire departments, aid organizations, and hospitals in order to manage catastrophic events. Given the initial configuration, access for external clients is denied. By temporarily adding an appropriate policy that is tailored to the requirements of such *Emergency coordinators*, access can be granted as long as the emergency situation persists. After normality is achieved, the additional policy can be revoked again. In Figure 4.9, an additional switch of the database connection (from  $dbuser1$  to  $dbuser2$ ) is shown, too. This approach is advisable for cases in which the required privileges for the database Web service vary in both situations – e.g., for realizing mandatory access control policies.

Thus, instead of developing partially redundant services for different situations, existing implementations can be reused by dynamically adapting the access control configuration.

## 4.4 Implementation

We integrated the presented access control optimizations for database Web services into our research Web service platform ServiceGlobe [Keidl et al. (2002)]. ServiceGlobe is a lightweight, distributed, and extensible Web 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* that are participating in ServiceGlobe federations. Service hosts are standard Internet servers that are additionally running the ServiceGlobe runtime engine.

In order to support the execution of mobile code, ServiceGlobe's security system provides protection mechanisms to monitor and control the execution of potential erroneous or even malicious code. As presented by Seltzsam et al. (2001) and Seltzsam (2005), security for service hosts is based on the Java security architecture, in particular the Java sand-box mechanism.<sup>7</sup> Secure messaging including authentication of requesters and secure communication is provided by WS-Security that relies on XML-Encryption and XML-Signature and specifies how security information is embedded into SOAP-messages.

ServiceGlobe's security system provides the core policy evaluation functionality. That is, ServiceGlobe acts as policy decision point (PDP) as illustrated in Figure 4.4. The implementation is based on Sun's XACML implementation.<sup>8</sup> We extended Sun's XACML engine with support for core and hierarchical role based access control and specified the (hierarchical) representation of database objects and actions in XACML. Moreover, we realized the policy engineering approach which was presented in Section 4.3. That includes the preparation of Web service policies based on the service specification, the extraction of database access control configurations, the specification of database user profiles to realize optimized access corridors, and the comparison of XACML policies.

We evaluated the feasibility of our approach by means of database Web services that rely on Oracle Database instances and provided a tool called WSDL2SG which eases the development of database Web services. Its input are service interface descriptions in the form of WSDL documents and corresponding specifications of the database interactions (also in XML format). Of course, because of security and privacy concerns, declarations of the database functionality and service specifications are separated. WSDL2SG generates initial Web service policies and a service framework, i.e., a Java based implementation that complies to the service description. In case the Web service developer assigns a database account that shall be used to run the interaction with the underlying database, WSDL2SG extracts corresponding database authorizations and performs a policy compliance check. Furthermore, the basic policy enforcement functionality is automatically prepared, meaning that policy enforcement points are inserted at the entry points of service methods. This also includes the preparation of corresponding XACML requests, thus, realizing context handler functionality according to Section 4.2.2.

The core XACML specification does not specify the evaluation of attribute values that are stored in data sources other than XML documents. When implementing fine-grained access control rules for database services, it is sometimes necessary to evaluate information provided by database systems. We realized this requirement by adjusting the functioning of `AttributeSelector` elements which are part of the XACML specification. Our version allows to connect to database systems and to query data, which can then be processed during further policy evaluation steps. Figure 4.10 illustrates an XACML Condition that evaluates whether John Carter is an attending physician of

---

<sup>7</sup>An overview over the security mechanisms of Java is, for example, provided by Posegga (1998).

<sup>8</sup>Project page: <http://sunxacml.sourceforge.net/>

```

<Condition FunctionId="string-equal">
  <Apply FunctionId="string-one-and-only">
    <AttributeSelector
      RequestContextPath="dbUri=HospitalDB
                          dbUser=dbuser
                          dbPassword=pwd
                          dbDriver=oracle.jdbc.OracleDriver
                          dbFrom=MedicalRecords
                          dbSelect=AttendingPhysician
                          dbWhere=Patient
                          rqWhere=patient-name"
      DataType="string"/>
    </Apply>
    <AttributeValue DataType="string">John Carter</AttributeValue>
  </Condition>

```

Figure 4.10: Example for integrating database queries into XACML policies

the patient who is identified by patient-name in the XACML request.<sup>9</sup> For this purpose, the following query will be executed and its result be compared with the string “John Carter”:

```

select AttendingPhysician
from MedicalRecords
where Patient = <patient-name>;

```

In order to post this query, a connection to the database needs to be established. The required connection information is included in the RequestContextPath attribute.<sup>10</sup> As policies are not publicly accessible and policy evaluation is performed by the trusted ServiceGlobe runtime environment, database connections are protected against malicious user requests.

As stated above, ServiceGlobe acts as PDP for (database) Web services. Policy evaluation proceeds as illustrated in Figure 4.11: First of all, the requester’s role memberships are evaluated (step 1) to determine all roles that are granted to the user. The user-role and role-role assignments are encoded in the role assignment policies that are introduced in Section 4.3. Based on the requester’s attributes (represented by the subject attributes in the request) and the gathered role information, the applicable base policies are determined (step 2). Next (in step 3), the base policies’ references to the applicable permission policies – i.e., the granted privileges – are resolved. Finally, in step 4, the request is evaluated against the preprocessed policy information. This can involve database queries to resolve AttributeSelector specifications as presented above. In case the policy applies to the request, service execution is granted.

<sup>9</sup>In order to improve readability, we shortened the DataType attribute identifiers. We also simplified the example through restricting the Condition to the physician John Carter. However, it can easily be adjusted to apply to arbitrary requesters.

<sup>10</sup>Proceeding this way, the XACML specification does not need to be modified. In future versions, we intend to extend the XACML schema for providing better ways to consistently integrate database queries into policies.

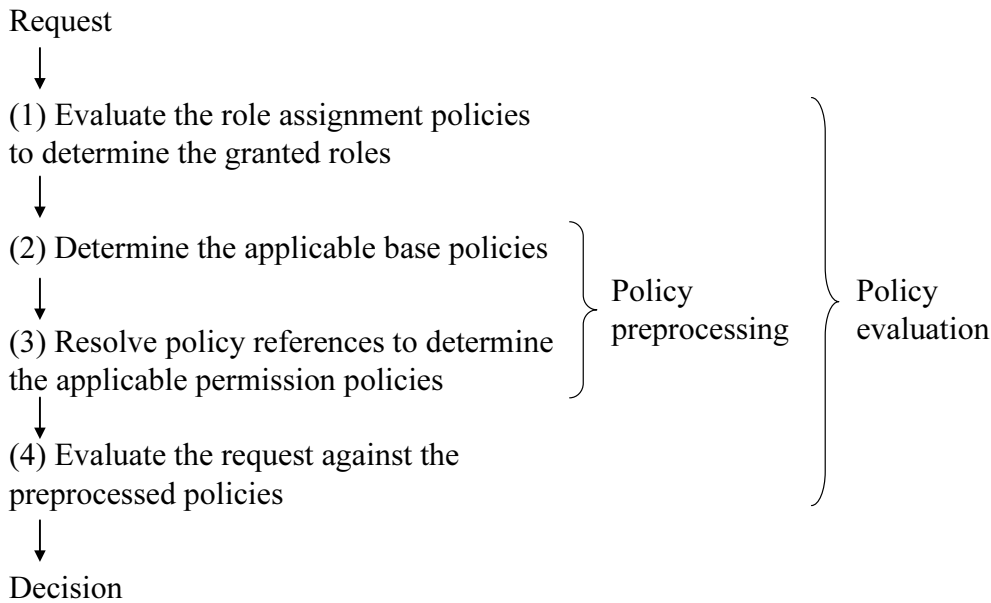


Figure 4.11: Policy evaluation process of the ServiceGlobe security system

## 4.5 Related Work

In order to reliably integrate database functionality into service-oriented IT infrastructures, the access control of database services has to be consolidated with the security configuration of database systems. Castano et al. (1994) give an overview of access control models of database management systems, including discretionary access control (DAC) and mandatory access control (MAC). While MAC is essential in environments with very stringent security requirements such as the military, commercial applications typically rely on DAC models. In Section 4.2.1, we briefly discussed the access control of the leading commercial RDBMS Oracle Database, IBM DB2, and Microsoft SQL Server. Despite many similarities, the syntax for declaring access control rules is database specific, in general. Therefore, access control configurations are typically not platform independent. Hence, a security engineering approach as described in this chapter is necessary to ease the reliable integration of database systems and the reconfiguration of database services.

Bhatti et al. (2004, 2003) present an XML-based access control specification language for Web services. Despite their substantial research contributions, generally accepted standards are of paramount importance for platform independent Web service applications. That is, as Web services are used to realize cross-organizational application integration processes (we will return to this aspect in detail in the next chapter), service providers have to agree on common security standards. Galbraith et al. (2002), O'Neill (2003), and Rosenberg and Remy (2004) provide comprehensive overviews over security models and specifications for Web services. Widely accepted standards are the Security Assertions Markup Language (SAML, [Cantor et al. (2005)]) and the Web Services Security framework (WS-Security, [IBM and Microsoft (2002);

Nadalin et al. (2006)]. These specifications focus on the exchange of security credentials, enabling authentication, authorization, message integrity, and confidentiality. For the specification and enforcement of access control we employ XACML. Several extensions of the core specification exist. Anderson (2005) shows how RBAC rules can be specified in XACML. A profile for the specification of Web service policies is presented by Moses (2003). Anderson and Lockhart (2005) focus on the interaction of XACML and SAML, describing how SAML protocol mechanisms can be used to query XACML policies from policy administration points.

Many Web service platforms and development toolkits like Bea WebLogic, IBM WebSphere, and the freely available Java Web Services Developer Pack (Java WSDP) employ Java security technology, supporting the realization of sophisticated authentication and authorization functionality for Web service applications. Implementing security functionality in an unstructured manner and without tool support often leads to mixing business logic and security logic. Kehr et al. (2001) and Walter et al. (2004) present an approach for securing Web applications through the use of tamper-resistant hardware like smart cards which store security credentials and provide encryption processing capabilities. For them, the separation of business logic and security logic is also in the main focus. Our security engineering approach decouples access control from the service implementation, thus, improving code quality and easing code maintenance. The tools presented in this chapter facilitate the identification of security requirements and the implementation of access control functionality – which, according to Gutiérrez et al. (2005), are fundamental characteristics of a reliable implementation of Web services' security.

Our approach is also related to Grid technology like the proposed Grid Services architecture of the Open Grid Forum (OGF<sup>11</sup>). The main objective of Grid initiatives, in particular of the Data Access and Integration working group (DAIS-WG), is to share resources in wide area networks, connecting numerous data providers and individual consumers. The Community Authorization Service (CAS, [Welch et al. (2003)]) of the Open Grid Services Architecture (OGSA) allows to set up Grid collaborations by enforcing security through a trusted authority. To do so, resource administrators can delegate the authorization management to the community. Keahey and Welch (2002) and Keahey et al. (2003) describe an approach for the specification and enforcement of fine-grained access control in Grid environments. However, resource administrators have to ensure that the deployed access control policies conform with the authorization of the underlying resources on their own. They point out that declarative policy languages like XACML would be meaningful alternatives to their preliminary policy language.

The DAIS-WG working group is concerned with the specification of interfaces for data resources, enabling the sharing of data in Grid environments. Distinguished specifications exist for the integration of relational database systems (WS-DAIR) and for XML data sources (WS-DAIX), see [Antonioletti et al. (2005a,b,c, 2006)]. Users are able to post their own queries to OGSA-DAI Grid services which are closely related to the specifications issued by DAIS-WG. For example, they are able to retrieve data from relational databases that are wrapped by respective Grid services. As Grid services rely on the Web services technology stack, security mechanisms for Web services can seamlessly be applied to them. The security system of the

---

<sup>11</sup>Project page: <http://www.ogf.org/>

OGSA-DAI framework offers several possibilities for administering security of Grid services.<sup>12</sup> Based on the security framework of the Globus<sup>13</sup> toolkit, authorizations for OGSA-DAI Grid services can be configured. Furthermore, authorizations can be administered for resources like XML and relational databases or files that are integrated by Grid services. Privileges can also be granted on the level of individual operations. Moreover, distinguished database accounts can be used for different clients by use of so-called role map documents. These specify resource-specific mappings from X.509 certificate credentials to database usernames and passwords. Our approach complements access control administration of Grid services by supporting the extraction of the least required privileges for the underlying databases. Compliance of service and database policies is ensured through policy comparisons as presented in Chapter 3.

## 4.6 Conclusion

In this chapter, we analyzed the access control of database services. Database services are Web services that rely on database systems for providing their functionality. We discussed and evaluated the different possibilities of realizing the access control of such services. The security enforcement can be shifted either more on the side of the underlying database systems or on the side of the services, i.e., the application layer. Our conclusion is that only the thorough consolidation with the access control of the underlying database systems constitutes a reliable and flexible way of implementing the multilayered security functionality of database services.

In this regard, we presented our security engineering approach for the specification of the access control of database services. Service policies and policy enforcement points are semi-automatically generated. Security vulnerabilities for the underlying database systems are avoided through the automatic specification of adequate database user profiles. That way, access to the database systems is restricted to those privileges required for providing the service functionality. Hence, security and privacy threats for database systems in case of service infiltrations are reduced.

This approach is also well suited for the specification of fine-grained access control (FGAC) rules which is required by many Web based applications. In order to realize FGAC, we take advantage of modern declarative policy languages like XACML. By verifying that the access control policies of database services constitute valid refinements of the corresponding database policies, services can be deployed reliably. Moreover, database Web services can operate as gatekeepers for the underlying database systems by blocking unauthorized requests on the service layer, thus, reducing the threat of denial of service attacks on the underlying databases.

A further benefit of our security engineering approach is the strict separation of security logic and application logic. Because of the application logic not being mixed with the security functionality, a concise software design is enabled. This design also supports dynamic modifications of security requirements by simply adapting the respective access control policies instead of demanding for changes of the service implementation or the database schema.

---

<sup>12</sup>Project page: <http://www.ogsadai.org.uk/documentation/ogsadai-wsrf-2.2/doc>

<sup>13</sup>Project page: <http://www.globus.org>



---

## Access Control in Dynamic Service Coalitions

---

In the previous chapters, we presented best practices for the reliable integration of business applications and resources into service-oriented IT infrastructures. Wiehler (2004) refers to this kind of integration processes as the initial phase of adopting service-oriented computing paradigms by enterprises. In this chapter, we extend our policy framework by role delegation and cross-domain assignment capabilities. These concepts allow the specification and enforcement of dynamic access control policies and the set-up of inter-organizational collaborations that, according to Wiehler, constitute oncoming application areas of Web services.

Our proposed security architecture supports both, tightly as well as loosely coupled federations. For the realization of loosely coupled Web service federations, the authorization autonomy of the participating organizations has to be preserved. In order to enable dynamic federations, the employed authorization infrastructure needs to be scalable and has to provide efficient policy enforcement mechanisms. In our framework, access control within collaboration networks proceeds as an interplay of local and distributed policy enforcement steps. On the one hand, scalability and performance are provided through the realization of role based access control. On the other hand, we devise caching strategies for the goal-oriented validation of assignments. The use of caches allows to optimize distributed authorization checks by reducing communication and policy evaluation costs. As authorizations must not succeed based on outdated cache entries, we employ caching techniques that provide the required strong cache consistency and analyze them with regard to their applicability in the authorization context.

First, we motivate the need for cross-domain assignments and role delegations in Section 5.1. These concepts demand for an extension of our policy model, which we present in Section 5.2. In Section 5.3, we describe our policy enforcement strategy supporting dynamic service coalitions. In Section 5.4, we present caching strategies for the optimization of distributed policy evaluation processes. The support of loosely and tightly coupled federations by means of our proposed authorization architecture is discussed in Section 5.5. Section 5.6 presents related work, before we summarize in Section 5.7. Excerpts of this chapter have been presented in [Wimmer et al. (2005a,b)] and [Wimmer and Kemper (2005)].

## 5.1 Motivation

Henning Kagermann, CEO of SAP, emphasizes that

*“Organizations have to pay special attention to processes that are based on a service-oriented IT architecture: They consist of a number of enterprise services whose origin is not completely known. In addition, they transcend traditional company borders and often run in distributed environments. Customers, suppliers, partners – they all play a part in innovative business scenarios.”*  
(from SAP Info vol. 136, 2006)

This citation points out that Web service compositions will increasingly integrate remote services and resources. A prerequisite for the realization of inter-organizational workflows is a flexible authorization architecture supporting the delegation of trust across closed trust domains and the enforcement of access control in service federations.

Figure 5.1 illustrates three collaborating hospitals, namely the Cook County General hospital (CCG), the Chicago Hope hospital (CH) and the Sacred Heart hospital (SH).<sup>1</sup> Each hospital offers services for the administration of the medical records of their patients. To simplify matters, we assume that each hospital provides a service interface that complies to *MedRecords-WS* which was introduced in Figure 4.2 on page 54. We showed that the consolidation of the access control of multilayered applications based on the policy model and the techniques presented in Chapters 3 and 4 allow the reliable integration of applications into service-oriented architectures. Since our policy model supports attribute-based specifications of access rules, it is well suited for the security requirements of Web service scenarios.

So far, the presented policy model provides static administration concepts for closed trust domains, denoting that only security officers can change the policy configuration of an organization. Nevertheless, in many real world scenarios, ordinary users have to be able to change policies as well, e.g., for the purpose of delegating jobs to other users. For example, the physician Kerry Weaver of the Cook County General hospital may want to delegate the treatment of some of her patients to one of her colleagues, say John Carter. The reason for this can be that John Carter is specialized on particular diseases, or just because Kerry Weaver is busy and has to hand over part of her workload. In order to realize this delegation, John Carter needs to be assigned the respective privileges to access the medical records of the transferred patients. Since Kerry Weaver herself is not able to delegate the permissions on her own, the security officer has to add corresponding delegation rules to the policy repository of the CCG hospital. In order to overcome this limitation and to enable a flexible policy management, we add the concept of (role) delegation to our policy model.

A further extension of our policy model is the support of cross-domain assignments. By means of cross-domain assignments, users acquire authorizations in foreign domains. Thus, by passing over privileges to federating parties, inter-organizational collaboration networks can be realized. As an example consider the service *PatientInfo-WS* shown in Figure 5.1. This service provides access on the medical records of patients housed in any of the three cooperating

---

<sup>1</sup>In the examples, names of hospitals and physicians are borrowed from the television shows ER, Chicago Hope, and Scrubs.

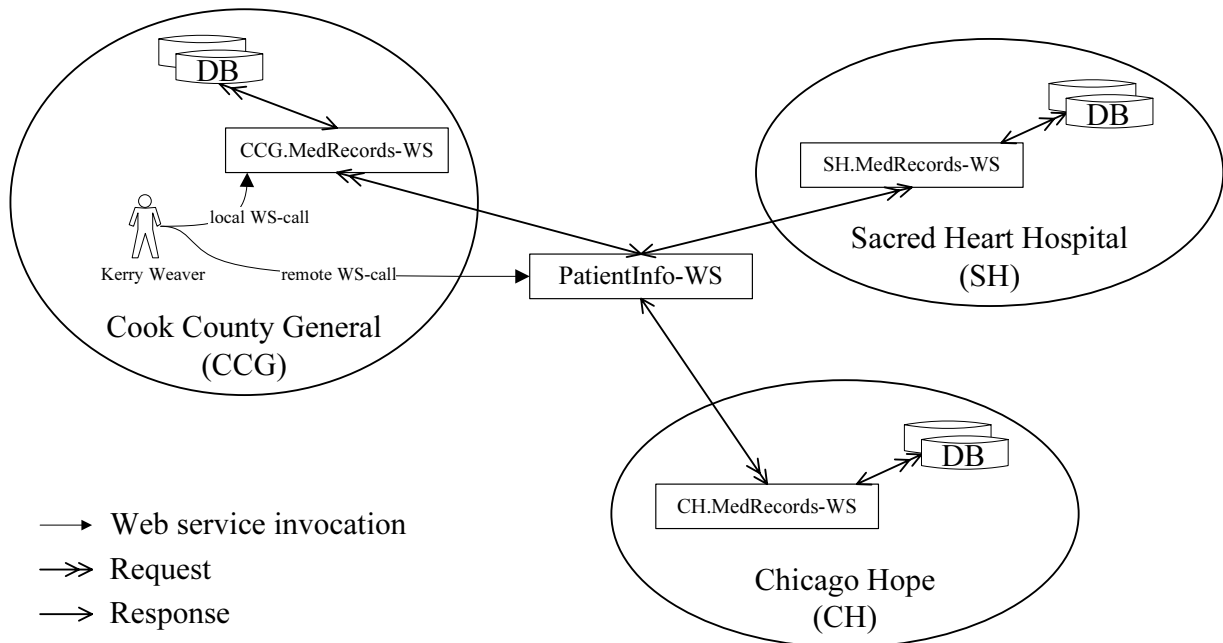


Figure 5.1: Intra-organizational and remote service executions

hospitals. Such a service portal allows to share results of new medications, e.g., in the course of a research collaboration, or to exchange diagnoses of patients who are treated in more than one hospital. As illustrated in the figure, *PatientInfo-WS* relies on the local *MedRecords-WS* instances. That means, users querying *PatientInfo-WS* require permissions for the local services, too. We distinguish two approaches for realizing access control in coalition environments. On the one hand, access control can be performed by *PatientInfo-WS*, which then operates as a central policy enforcement point. That is, any request authorized for *PatientInfo-WS* is implicitly authorized for the local services. As a precondition, the cooperating organizations have to transmit application specific policies to the central enforcement point. This configuration is referred to as tightly coupled and typically demands for a high level of trust among the cooperating parties, since sensitive access control information needs to be shared. On the other hand, cooperating organizations retain their authorization autonomy in loosely coupled systems, meaning that access control is enforced by the individual domains. In our scenario, this implies a distributed access control strategy. Loosely coupled systems are usually employed for short-term coalitions or coalitions with dynamic trust relationships.

Cross-domain assignments in combination with role delegations result in networks of distributed authorization interdependencies. In such settings, requests are authorized if applicable delegation chains can be derived. In case of loosely coupled systems, access control is performed in a distributed manner. Though distributed policy evaluation strategies can be realized in highly dynamic networks with changing trust relationships, they usually offer less performance than their tightly coupled counterparts. This is due to additional communication costs. In order to optimize distributed policy evaluations, we devise a caching strategy for the goal-oriented vali-

dation of assignments. As authorizations must not be based on outdated cache entries, we employ caching techniques that provide the required strong cache consistency and analyze them with regard to their applicability in the authorization context.

## 5.2 Extended Policy Model

### 5.2.1 Terminology and Notation

In this section, we extend our policy model presented in Section 3.2 by cross-domain assignment and role delegation capabilities. A cross-domain assignment grants access on resources to subjects of foreign domains. In this context, a *domain* is an entity with the following characteristics:

1. It is uniquely identifiable.
2. It administrates resources, user identities and roles. That is, resources, e.g., database systems, are uniquely assigned to the domain that hosts them. Analogously, domains manage identities of registered users, for instance, the employees of a company. Moreover, a domain provides its own local role management.
3. A domain autonomously administrates the security policies for its assigned resources.

Domains represent closed trust realms. Concerning the use case illustrated in Figure 5.1, the hospitals represent three distinguished domains. Domains can also be organized in hierarchies. For instance, the CCG domain can be subdivided into sub-domains for its departments, like cardiology, surgery, or emergency room. Each sub-domain administrates its own resources like the medical records of treated patients. Through the concept of delegation, administration of shared resources can also be granted to authorities of the superordinate domain.

*Delegation* is the process whereby an entity authorizes another entity to access certain resources. The delegating entity is also called grantor and the delegated entity is referred to as grantee. Delegation allows users to act on behalf of someone else. For example, a chief physician delegates jobs (and therewith the required privileges) to subordinates so that the workload is balanced among the employees. Due to delegation, grantees can perform the assigned jobs without referring back to the grantor. Delegations can be restricted temporarily. For instance, shared access on therapy results within the hospital network can be constrained to the duration of the joint research collaboration.

In Section 3.2, we defined access rules as tuples  $(S, O, A, c)$  specifying the authorized subjects  $S$ , the accessible objects  $O$ , and the granted operations  $A$ . The assignment is restricted through the condition  $c$ . In order to support role delegations, we have to protocol the entities that released delegations, i.e., the *issuers*. This information is needed when delegations are revoked. Therefore, issuers need to be uniquely identifiable, like individual users that are identified by their user-id. Hence, we extend the specification of a rule  $(S, O, A, c)$  by the issuer flag, meaning that a rule is represented by a tagged tuple  $(S, O, A, c)I$ , with  $I$  representing the issuer. In case a rule is generally accepted – i.e., it is not issued by an individual user –, the issuer is the local domain.

In the following, we use a specific notation for representing assignments, improving readability and easing the description of the distributed policy evaluation strategy. However, the formal notation does not introduce new policy semantics and can be expressed in the extended policy model.

The assignment of a privilege  $(O, A)$  to subjects  $S$ , issued by  $I$  and constrained by the optional condition  $c$  is represented through

$$[S \rightarrow_P (O, A)]_c I \quad (5.1)$$

Thus, assertion (5.1) corresponds to the rule  $(S, O, A, c)I$ . We provide special notations for the assignment of privileges to roles and for the representation of role hierarchies. The assignment of a privilege  $(O, A)$  to a role  $r$  is represented by

$$[(role = r) \rightarrow_P (O, A)]_c I \quad (5.2)$$

The assignment of a role  $r$  to users that are characterized by subject attributes  $S$ , constrained by  $c$  and issued by  $I$ , is expressed by

$$[S \rightarrow_R (granted-role = r)]_c I \quad (5.3)$$

Analogously, dependencies in role hierarchies can be expressed. Via the following assertion,  $r_{\text{senior}}$  is declared to be a senior role of the role  $r_{\text{junior}}$

$$[(role = r_{\text{senior}}) \rightarrow_R (granted-role = r_{\text{junior}})]_c I \quad (5.4)$$

An example for representing expressions (5.3) and (5.4) in our policy model is shown in Section 3.2.1.3, e.g., by means of rule  $R_2$ .

Assertions in the form of (5.1) to (5.4) realize a static rights management which – apart from the issuer information – was already supported by our original policy model. The following assertions provide role management privileges, i.e., represent the rights to delegate and revoke roles, which are the basis for a dynamic rights management:

$$[S \xrightarrow_R^{\text{delegate}} (d-role = r)]_c I \quad (5.5)$$

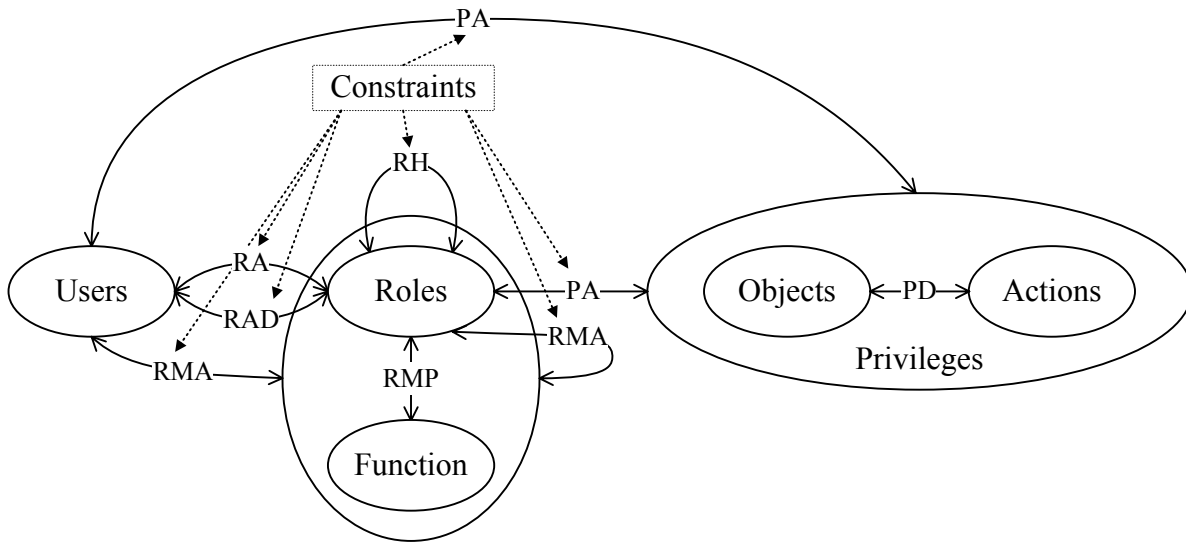
$$[S \xrightarrow_R^{\text{revoke}} (d-role = r)]_c I \quad (5.6)$$

The revocation of role assignments can follow various strategies. For instance, revocations can be performed in a cascading or non-cascading manner. An overview over revocation strategies is given in Section 5.2.4. In our policy model, expressions (5.5) and (5.6) are represented analogously to role assignments. For instance, expression (5.5) is represented by  $(S, (d-role = r), (method = \text{delegate}), c)I$ .

Role delegation and revocation privileges can also be granted to further roles. The following assertions provide role  $r'$  with administration rights for role  $r$ :

$$[(role = r') \xrightarrow_R^{\text{delegate}} (d-role = r)]_c I \quad (5.7)$$

$$[(role = r') \xrightarrow_R^{\text{revoke}} (d-role = r)]_c I \quad (5.8)$$



PD	privilege definition	RAD	delegated role assignment
PA	privilege assignment	RMP	role management permission
RH	role hierarchy	RMA	role management assignment
RA	role assignment		

Figure 5.2: Extended policy model supporting role delegations

Cross-domain assignments require users (respectively user identities), resources, and roles to be uniquely identifiable. This is achieved through domain affiliations that are denoted through prefix notation. For instance, the assertion

$$[(uid = CCG.Kerry\ Weaver) \rightarrow_R (granted\text{-}role = SH.CoopPhysician)]_{true}SH$$

states that Kerry Weaver who is registered at the Cook County General hospital is granted the CoopPhysician (cooperating physician) role of the Sacred Heart domain. The assertion is an example for a static rule that is issued by the SH domain. In this example, the subject's affiliation is CCG. Thus, it varies from the domain of the granted role, which is SH. Therefore, the assignment represents a cross-domain role-assignment as mentioned above. Analogous considerations apply to cross-domain assignments of privileges. The privilege to delegate or revoke a role does not presuppose issuers to be members of the role themselves. Self-delegations can be prohibited by use of conditions.

Figure 5.2 illustrates the augmented policy model. In addition to the policy model shown in Figure 3.2 on page 21, it supports the described role delegation concepts: A role management permission (RMP) specifies the privilege to delegate or revoke roles. RMPs can be granted to subjects via role management assignments (RMAs). With regard to the assignment of roles to subjects, we differentiate between assignments that are issued by domains (RA) and third party delegations through authorized users (RAD).

The extended policy model supports a variety of administration schemes (see Section 2.2.4), e.g., including centralized administration schemes that were already supported by the basic access control model. Additionally, because of the delegation mechanism, the ownership paradigm and hierarchical schemes are enabled. Since inter-organizational assignments are supported, also a decentralized policy administration can be applied.

### 5.2.2 Multistep Delegations

Via role delegations and cross-domain assignments, privileges can be granted to users of foreign domains. A role can be delegated to individual users or roles of cooperating domains – whereby delegations are possibly restricted through conditions. In case the grantee of a role delegation is a role (cf. assertion (5.7)) and this recipient role can be further delegated, *multistep delegations* are realizable. That is, due to transitivity, roles defined at a domain  $D$  can be granted to roles and users of a domain  $D'$  through several intermediary delegation steps. Thus, through cross-domain role assignments, networks of trust relationships can be realized. As an example consider the three assertions

$$\begin{aligned} &[(uid = \text{CCG.Kerry Weaver}) \rightarrow_R (granted\text{-}role = \text{CCG.ChiefPhysician})]_{\text{true}} \text{CCG} \\ &[(role = \text{CCG.ChiefPhysician}) \rightarrow_R (granted\text{-}role = \text{SH.CoopPhysician})]_{\text{true}} \text{SH} \\ &[(role = \text{SH.CoopPhysician}) \rightarrow_R (granted\text{-}role = \text{CH.ProjectMember})]_{\text{true}} \text{CH} \end{aligned}$$

The first assertion states that Kerry Weaver is working as a chief physician at the Cook County General hospital, therefore being granted the corresponding role. Because of a collaboration with the Sacred Heart hospital, chief physicians of the Cook County General are granted the CoopPhysician role through the second assertion. The third assertion declares that members of the role CoopPhysician are granted the role ProjectMember of the Chicago Hope domain. Thus, through multistep cross-domain assignments, Kerry Weaver is granted access rights within the Chicago Hope domain. The authorization is justified through the *delegation chain*

$$\text{CCG.Kerry Weaver} \rightarrow_R \text{CCG.ChiefPhysician} \rightarrow_R \text{SH.CoopPhysician} \rightarrow_R \text{CH.ProjectMember}$$

In general, a delegation chain is a sequence of role assignments expressing role inheritances. Delegation chains represent authorization proofs, stating that the subjects specified by the beginning of the chain are granted the roles which are linked in the chain. A delegation chain can include roles of various domains. The *delegation depth* declares the number of domain hops. In the given example, the delegation depth is two. In general, the delegation depth of the chain

$$D_0.S \rightarrow_R D_1.r_1 \rightarrow_R D_2.r_2 \rightarrow_R \dots \rightarrow_R D_{n-1}.r_{n-1} \rightarrow_R D_n.r_n$$

is the cardinality of  $\{i \mid 0 \leq i < n, D_i \neq D_{i+1}\}$ .

### 5.2.3 Policy Representation and Implementation

In Sections 4.3 and 4.4, we described the representations of formal policy specifications in XACML. In order to also represent cross-domain assignments and role delegations, we made

the following adjustments: We introduced new subtypes of permission policies representing role delegation and role revocation rights. For instance, the delegation permission for the role CCG.AttendingPhysician is specified by a permission policy whose Resource references the role and whose Action is the keyword *delegate*. The XACML representation of this policy is outlined in Appendix C.4. Delegation permission policies are assigned to users or roles via base policies. Permission policies for role revocations are defined and applied analogously, using the keyword *revoke*. A role delegation is carried out through the generation of a corresponding role assignment policy. Let's assume that Kerry Weaver is granted the privilege to delegate the role CCG.AttendingPhysician, i.e.,

$$[(uid = \text{CCG.Kerry Weaver}) \rightarrow_R^{\text{delegate}} (d\text{-role} = \text{CCG.AttendingPhysician})]_{\text{true}} \text{CCG}$$

If she delegates this role to John Carter, denoted as

$$[(uid = \text{CCG.John Carter}) \rightarrow_R (granted\text{-role} = \text{CCG.AttendingPhysician})]_{\text{true}} \text{CCG.Kerry Weaver}$$

a role assignment policy as shown in Appendix C.4 is generated. As illustrated there, Issuer tags identify the issuer of a delegation and declare domain affiliations of objects and users. Following the ownership paradigm, the generated policies are added to the policy repository of the domains that administer the delegated roles. Considering two requests with the first one performing a delegation of any role  $r$ , while the second initiates the revocation of the respective delegation permission, policy repositories will become inconsistent. Therefore, policy repositories have to be modified exclusively, meaning that they have to be locked during updates. Otherwise, concurrency conflicts can arise.

We integrated the described inter-organizational delegation mechanism into our research Web service platform ServiceGlobe. So-called *delegation services* supervise the policy repositories of domains and provide role delegation and revocation functionality. Furthermore, they act as policy decision points within dynamic service coalitions, building the backbone for realizing distributed policy enforcement (see Section 5.3).

## 5.2.4 Revocation Schemes

In order to delegate a role, a respective role assignment policy is issued. The reverse operation, i.e., the revocation of a delegation, implies the invalidation of the corresponding role assignment policies. Thereby, attention has to be drawn to the revocation of role assignments that include the right to delegate the same or further roles. The question arises what to do with role delegations that have been issued based on the revoked role.

We present different revocation schemes by means of an example. Assume that the following



is an excerpt of the policy repository of the Cook County General hospital:

$$\begin{aligned} & [(role = CCG.ChiefPhysician) \rightarrow_R^{\text{delegate}} (d-role = CCG.ChiefPhysician)]_{\text{true}} CCG \\ & [(role = CCG.ChiefPhysician) \rightarrow_R^{\text{delegate}} (d-role = CCG.Surgeon)]_{\text{true}} CCG \\ & [(role = CCG.ChiefPhysician) \rightarrow_R^{\text{delegate}} (d-role = CCG.Internist)]_{\text{true}} CCG \\ & [(uid = CCG.Mark\ Greene) \rightarrow_R (granted-role = CCG.ChiefPhysician)]_{\text{true}} CCG \end{aligned}$$

These assertions state that chief physicians of the Cook County General can delegate the roles ChiefPhysician, Surgeon and Internist. That is, someone who is granted the ChiefPhysician role is allowed to perform further delegations. For instance, Mark Greene, who is granted this role, can initiate the following delegations:

- Grant ChiefPhysician to Douglas Ross.
- Grant Internist to Kerry Weaver.

Due to the first delegation, Douglas Ross is also authorized to delegate the three roles. He, for example, can issue the following assignments:

- Grant Internist to Kerry Weaver.
- Grant Surgeon to Kerry Weaver.

Therefore, the following assertions are added to the policy repository of the CCG domain:

$$\begin{aligned} & [(uid = CCG.Douglas\ Ross) \rightarrow_R (granted-role = CCG.ChiefPhysician)]_{\text{true}} CCG.Mark\ Greene \\ & [(uid = CCG.Kerry\ Weaver) \rightarrow_R (granted-role = CCG.Internist)]_{\text{true}} CCG.Mark\ Greene \\ & [(uid = CCG.Kerry\ Weaver) \rightarrow_R (granted-role = CCG.Internist)]_{\text{true}} CCG.Douglas\ Ross \\ & [(uid = CCG.Kerry\ Weaver) \rightarrow_R (granted-role = CCG.Surgeon)]_{\text{true}} CCG.Douglas\ Ross \end{aligned}$$

Figure 5.3(a) illustrates the delegation interdependencies that arise from these rules. As can be seen, Kerry Weaver is granted the role Internist according to two independent assignments – the first issued by Mark Greene and the second by Douglas Ross.

The concept of role delegation needs to be accompanied with possibilities of role revocations. For instance, if Douglas Ross resigns, his authorizations at the Cook County General hospital have to be deleted, meaning that he should be revoked from the delegated role ChiefPhysician. In this case, it is reasonable to keep delegations issued by Douglas Ross unchanged. If Douglas Ross is fired because he is suspected of being disloyal, the delegations he issued should be removed from the system as well. Thus, the authorization system has to control revocations, guaranteeing that only authorized entities can revoke assignments and that revocations are performed according to a predefined revocation scheme.

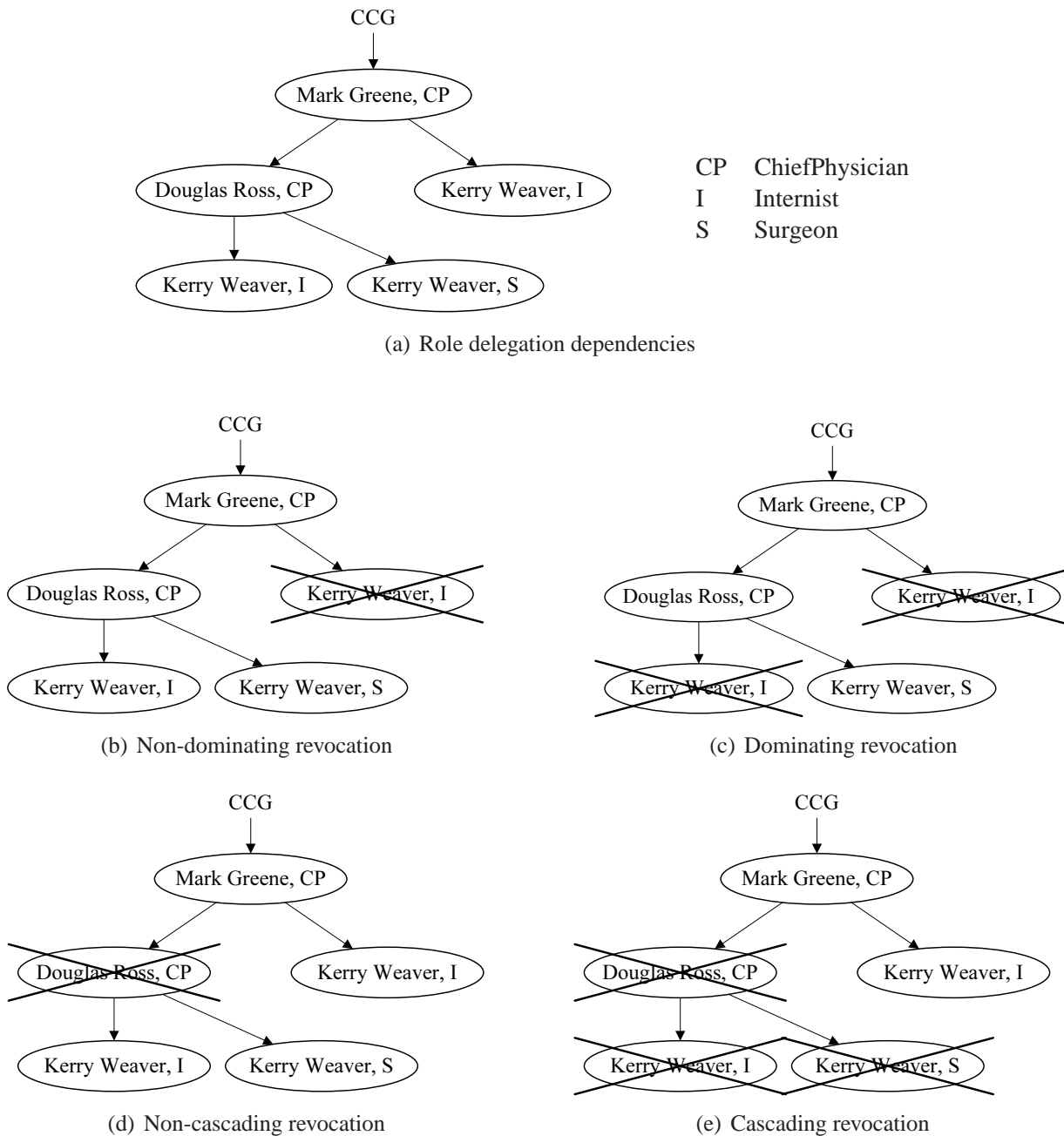


Figure 5.3: Effects of different revocation schemes

#### 5.2.4.1 Grant-dependency

According to our policy model, users are explicitly authorized to revoke assignments by being granted revocation permissions. For instance, chief physicians that are able to delegate the roles ChiefPhysician, Surgeon, and Internist are granted the privilege to revoke these roles according

to the following rules

$$\begin{aligned} & [(role = CCG.ChiefPhysician) \xrightarrow{R}^{revoke} (d-role = CCG.ChiefPhysician)]_{true} CCG \\ & [(role = CCG.ChiefPhysician) \xrightarrow{R}^{revoke} (d-role = CCG.Surgeon)]_{true} CCG \\ & [(role = CCG.ChiefPhysician) \xrightarrow{R}^{revoke} (d-role = CCG.Internist)]_{true} CCG \end{aligned}$$

As these assertions are not restricted by any condition, any user who is granted the ChiefPhysician role can revoke any of these assignments. This is also referred to as *grant-independent* revocation. For instance, the chief physician Mark Greene can revoke the assignment of the role Surgeon to Kerry Weaver, although this assignment was issued by Douglas Ross.

According to Barka and Sandhu (2000a), a revocation scheme is *grant-dependent* if delegations can only be repealed by the grantors. In our model, this is realized via conditions in the form of  $(issuer = requester)$  if *issuer* is the grantor (stored in the role assignment policy, see Section 5.2.3 and Appendix C.4) and *requester* is the subject that requests to revoke the delegation. Hence, grant-dependent as well as grant-independent revocation schemes are supported by our authorization framework.

#### 5.2.4.2 Dominance and Propagation

Hagström et al. (2001) provide a classification of revocation schemes, introducing the three dimensions *resilience*, *dominance*, and *propagation*. The first dimension, resilience, describes the technical realization of revocations. Possible approaches are the deletion of assignments and the publication of negative assertions, i.e., denials that overrule grants. As negative authorization is not considered by our approach, we focus on the dimensions dominance and propagation.

The dominance dimension describes the handling of conflicts which arise when subjects are delegated the same roles by several grantors. In our example, Kerry Weaver is granted the role Internist by Mark Greene and by Douglas Ross. Figure 5.3(b) shows the result of a *non-dominating* (also called *weak*) revocation of the role Internist induced by Mark Greene: As illustrated, only his delegation is deleted. Hence, Kerry Weaver is still granted this role due to a separate assignment by Douglas Ross. In contrast to this, the revocation is called *dominating* or *strong* if the delegation of Douglas Ross (who received the right to delegate the role from Mark Greene) is removed as well. The result is illustrated in Figure 5.3(c). Dominating revocation schemes have to take into account delegation dependencies, meaning that only assignments which are along a path from the root to a leaf in the tree can be deleted. Regarding our example scenario, Douglas Ross will not succeed in revoking all assignments of the role Internist from Kerry Weaver, as the delegation by Mark Green does not depend on a delegation of Douglas Ross – but arguably the other way round.

The last dimension, propagation, distinguishes between *non-cascading* and *cascading* revocations. Considering our example, Figure 5.3(d) illustrates a non-cascading revocation of the role ChiefPhysician from Douglas Ross by Mark Greene. As shown in the figure, delegations issued by Douglas Ross are not affected. A non-cascading revocation would, for instance, be applied if Douglas Ross retires. Privileges and roles he was granted in the context of the Cook County General hospital need to be revoked then. Nevertheless, delegations he issued during

the time of his employment will not lose validity. Applying a recursive revocation scheme to this scenario would lead to the undesired effect of deleting all assignments Douglas Ross issued and, recursively, all the delegations that were performed based upon them. Figure 5.3(e) shows the result of the cascading revocation of the role ChiefPhysician from Douglas Ross. Cascading revocations induce global updates of role assignments and offer possibilities to clear all effects of malicious users.

Consequently, eight revocation schemes can be realized through possible combinations in the dimensions grant-dependency, dominance, and propagation. Nevertheless, these configurations vary with regard to their usability in dynamic coalitions. As cooperating domains are autonomously administrating their policy repositories, dominating revocation schemes might not be acceptable and can imply high communication costs. Furthermore, the implementation has to handle exceptions like the temporal unavailability of nodes in the network. Similar considerations apply to cascading revocations. However, dominating and/or cascading revocation variants can be used in the context of tightly coupled federations, where authorization is performed centrally. The authorization system of ServiceGlobe provides the basis for loosely coupled federations, performing non-dominating and non-cascading revocations by default.

## 5.3 Policy Evaluation

In dynamic service coalitions, access control is performed through the interplay of local and distributed policy evaluation steps. If a user invokes a Web service which is provided by domain  $D$ , first of all the policy repository of  $D$  is browsed, evaluating whether a local authorization decision can be inferred. This is referred to as local policy evaluation. If the request cannot be authorized based on the  $D$ -local rules, distributed policy evaluation is initiated. Through distributed policy evaluation it is checked whether  $D$  participates in a collaboration network that grants the privileges needed to execute the service to the requester. In this regard, our authorization model does not depend on a central authority (also called *trusted third party*), which acts as central policy decision point within the network. Instead, in loosely coupled federations, the cooperating domains retain their authorization autonomy.

In the following,  $e$  denotes a request specifying the subject, the activity (i.e., resource and requested operation), and context information. In our scenario, the request applies to the execution of a Web service hosted by domain  $D$ .

### 5.3.1 Local Policy Evaluation

Local policy evaluation denotes that only policies of  $D$ 's policy repository are evaluated. The evaluation process based on the augmented policy model proceeds similar to the process described in Section 4.4:

1. First of all, the set  $\mathcal{R}_{\text{local}}$  which consists of roles administered by domain  $D$  that are assigned to the requester (specified by  $e$ ) are determined:

$$\mathcal{R}_{\text{local}} \stackrel{\text{def}}{=} \{D.r \mid D.r \text{ is a role} : \exists [S \rightarrow_R D.r']_c I \wedge \llbracket S \rrbracket_e = \text{true}, \llbracket c \rrbracket_e = \text{true}, D.r' \succeq_D^e D.r\}$$

That is,  $\mathcal{R}_{\text{local}}$  consists of those roles of the  $D$ -local role hierarchy which are granted to the requester.  $\mathcal{R}_{\text{local}}$  is determined by analyzing the role assignment policies of domain  $D$ .

In the above definition,  $\succeq_D^e$  represents the order defined by the  $D$ -local role hierarchy, whereby only role inheritances applying to  $e$  are taken into account. That is,  $D.r' \succeq_D^e D.r$  holds, in case  $r'$  is equal to  $r$  or  $r'$  is a senior role of the role  $r$  and any condition  $c$  restricting the applicability of any role inheritance is fulfilled by the request  $e$ . Formally,  $D.r' \succeq_D^e D.r$  if there exists a sequence of assignments  $[D.r_i \rightarrow D.r_{i-1}]_{c_i} I_i$ , with  $1 \leq i \leq n$ ,  $r_0 = r$ ,  $r_n = r'$  and  $\llbracket c_i \rrbracket_e = \text{true}$  for all  $i$ .

In general, the topmost roles in the  $D$ -local role hierarchy can be defined in foreign domains, i.e., there can be  $D'.r' \succeq_D^e D.r$  with  $D' \neq D$ . But there can be no further role  $D''.r''$  with  $D'' \neq D$  and  $D''.r'' \succeq_D^e D'.r' \succeq_D^e D.r$ . This is due to the ownership paradigm which requires the assignment  $D''.r'' \succeq D'.r'$  to be stored in the policy repository of domain  $D'$ .

2. Second, the set  $\mathcal{P}$  of granted privileges is determined, which is defined as

$$\mathcal{P} = \{(O, A) \mid (\exists [S \rightarrow_P (O, A)]_c I \wedge \llbracket S \rrbracket_e = \text{true}, \llbracket c \rrbracket_e = \text{true}) \vee (\exists [D.r \rightarrow_P (O, A)]_c I \wedge \llbracket c \rrbracket_e = \text{true}, r \in \mathcal{R}_{\text{local}})\}$$

$\mathcal{P}$  is calculated by scanning the base policies of  $D$ .

3. If any privilege  $(O, A) \in \mathcal{P}$  applies, i.e.,  $\llbracket (O, A) \rrbracket_e = \text{true}$ , the request  $e$  is locally authorized. This last evaluation step is performed by checking the permission policies of the  $D$ -local policy repository.

### 5.3.2 Distributed Policy Evaluation

If local policy evaluation fails, the request might still be authorized due to cross-domain assignments. For instance, if domain  $D$  collaborates with domain  $D'$ , roles of  $D$  are assigned to entities (users or roles) of domain  $D'$  and vice versa. Due to the ownership paradigm, the respective role assignment policies are administered by the domains the assigned roles belong to. Thus, the evaluation of authorizations within a federation implies a distributed policy evaluation as illustrated in Figure 5.4(a):

1. First, those domains are determined which are candidates for authorizing  $e$ . As local policy evaluation failed, we know that the requester cannot be granted the execution rights due to an assignment of the form  $[S \rightarrow_P (O, A)]_c I$  with  $\llbracket (S, O, A, c) \rrbracket_e = \text{true}$ . The only possibility left is that the requester is granted a role  $D.r$  that authorizes the request, whereby this authorization is based on a cross-domain role assignment. The set  $\mathcal{R}_{\text{foreign}}$  is defined as follows:

$$\mathcal{R}_{\text{foreign}} \stackrel{\text{def}}{=} \{D'.r' \mid D'.r' \text{ is a role, } D' \neq D : \exists D.r \text{ with } [D.r \rightarrow_P (O, A)]_c I \text{ such that } \llbracket (O, A) \rrbracket_e = \text{true}, \llbracket c \rrbracket_e = \text{true}, D'.r' \succeq_D^e D.r\}$$

$\mathcal{R}_{\text{foreign}}$  is calculated based on the evaluation of the  $D$ -local policy repository.

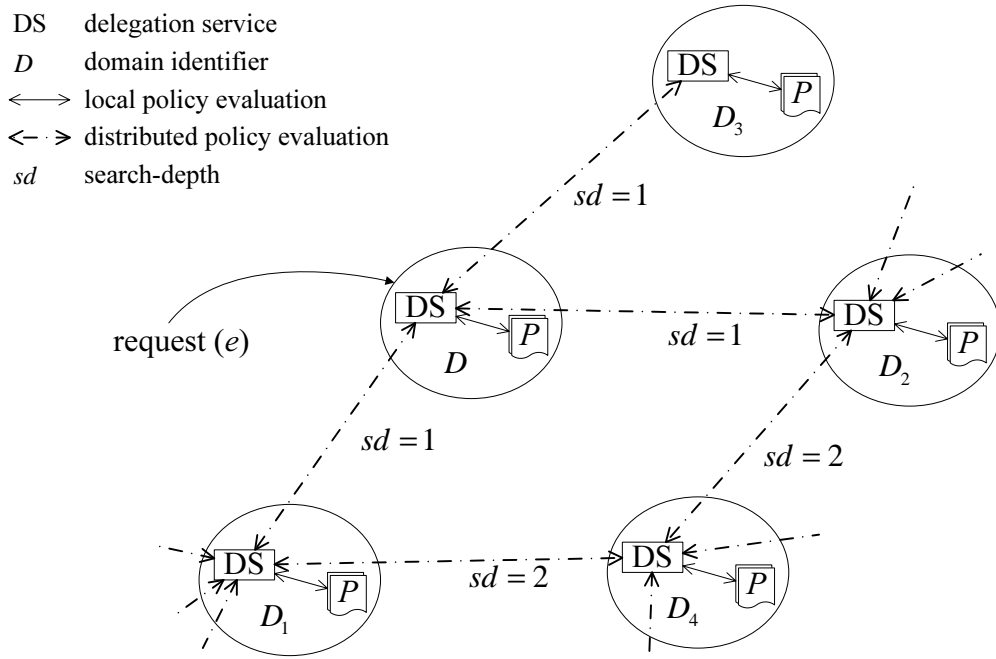
2. The request  $e$  is authorized if at least one role  $D'.r' \in \mathcal{R}_{\text{foreign}}$  is assigned to the requester. Distributed authorization is initiated by evaluating the policy repository of domain  $D'$ :
  - a) A  $D'$ -local policy evaluation is performed to find out whether the requester is granted the role  $D'.r'$  due to the  $D'$ -local role hierarchy. That is, in case there exists an assignment  $[S \rightarrow_R D'.\tilde{r}]_c I$  with  $\llbracket S \rrbracket_e = \text{true}$ ,  $\llbracket c \rrbracket_e = \text{true}$ , and  $D'.\tilde{r} \succeq_{D'}^e D'.r'$ , the request is granted because of transitivity. In this case, distributed policy evaluation terminates.
  - b) Otherwise, authorization can still be inferred based on a further delegation step. Therefore, we calculate the set  $\mathcal{R}'_{\text{foreign}}$  of senior roles of  $D'.r'$  which belong to cooperating domains.  $\mathcal{R}'_{\text{foreign}}$  is defined as:

$$\mathcal{R}'_{\text{foreign}} = \{D''.r'' \mid D''.r'' \text{ is a role, } D'' \neq D', D''.r'' \succeq_{D'}^e D'.r'\}$$

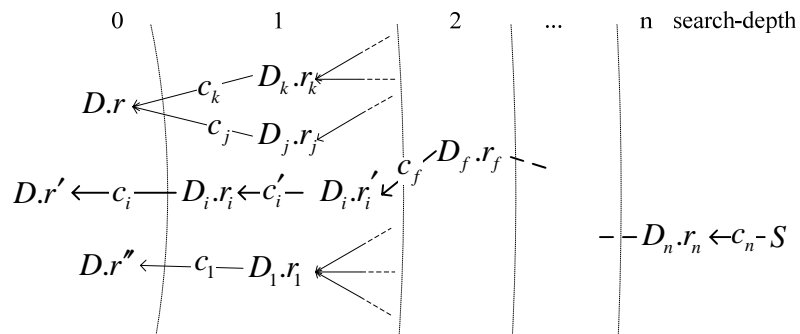
Distributed evaluation branches by evaluating the policy repositories of the domains  $D''$ , querying whether the requester is granted any role  $D''.r''$ . Thus, each branch can lead to further distributed policy evaluations, denoting that step 2.b) will be executed repeatedly. Distributed policy evaluation terminates unsuccessfully at domain  $D'$ , in case  $\mathcal{R}'_{\text{foreign}} = \emptyset$ .

Therefore, distributed policy evaluation corresponds to the evaluation of distributed role relationships which are defined through cross-domain role assignments. That is, trust relationships among cooperating domains are represented by a graph of distributed role assignments. In this graph, nodes stand for individual roles and edges represent (potentially distributed) role assignments that are labeled with conditions. There is a directed edge from  $D'.r'$  to  $D.r$  annotated with  $c$  if and only if there exists a role assignment policy  $[D'.r' \rightarrow_R D.r]_c I$ . As explained before, this assignment is stored as a role assignment policy at domain  $D$  which is the owner of role  $r$ .

An authorization is inferred if a delegation chain is derived which applies to the request  $e$ . Figure 5.4(b) illustrates that applicable delegation chains are determined by means of a backwards oriented depth first search (DFS) on the distributed role assignment graph. Whenever foreign policy repositories have to be evaluated, the search-depth is increased by one. In our prototype, delegation services provide the functionality to query distributed role assignments. It can be assumed that real-world collaborations tend to reveal short delegation depths, denoting that collaboration networks are likely to span only few trust domains and that organizations are usually cooperating with a reduced number of partners. Consequently, a breadth first search (BFS) algorithm is supposed to be particularly suitable by providing better response times than DFS. One possibility for realizing BFS is to compute a local copy of the role assignment graph which is then evaluated by a central authority. However, this contradicts security requirements of dynamic service coalitions. Alternatively, a breath first search can be realized on the distributed graph by iteratively increasing the search-depth. That is, first, delegations spanning two domains are evaluated. Then, in case authorization did not succeed, delegations involving three domains are checked, etc. However, due to the iterative enlargement of the search range, this strategy results in high network load. Hence, a BFS-variant can hardly be realized. Techniques for improving the performance of distributed policy enforcements are discussed in Section 5.4.



(a) Distributed policy evaluation in a network of collaborating domains

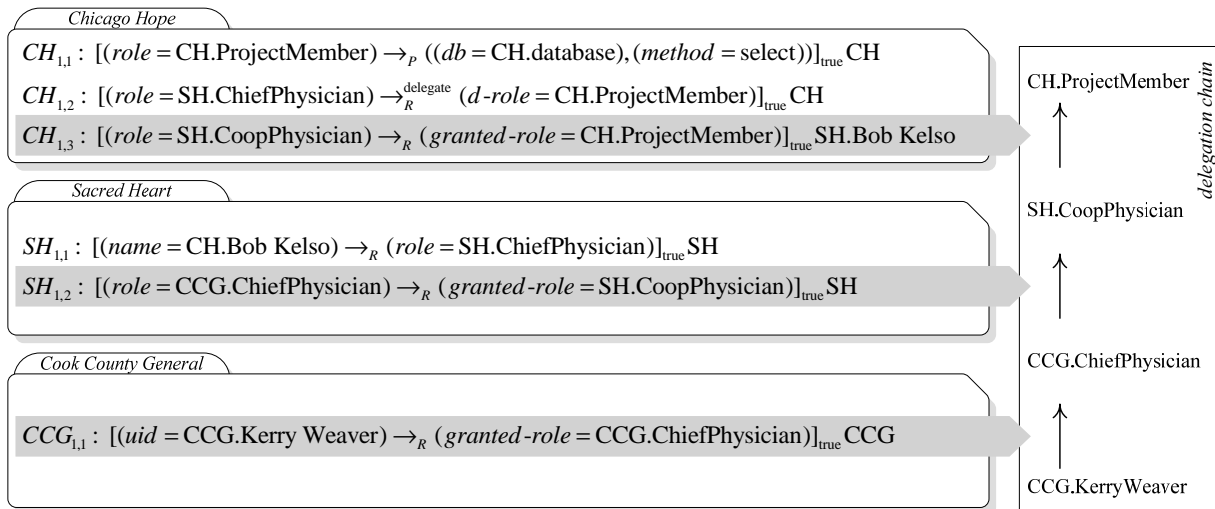


(b) Evaluation of the distributed role assignment graph

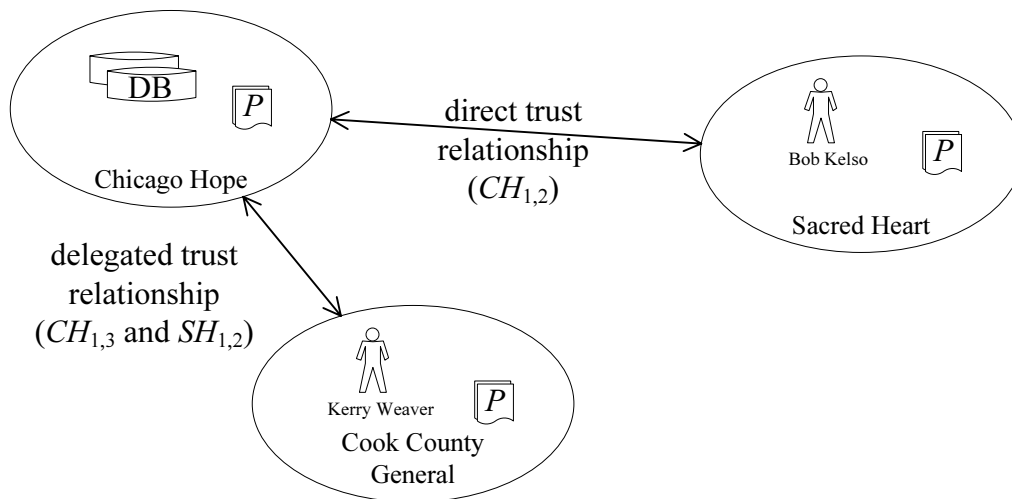
Figure 5.4: Distributed policy evaluation

### 5.3.3 Example

We demonstrate the distributed policy evaluation process by means of the scenario introduced in Section 5.1. We model a collaboration network consisting of the three hospitals Chicago Hope (CH), Sacred Heart (SH), and Cook County General (CCG). In the example, we focus on the execution of *PatientInfo-WS* by the physician Kerry Weaver who works at the Cook County General. Assuming that she queries the medical record of the patient Mr. Geiger who is housed at the



(a) Extracts of the policy repositories



(b) Trust relationships

Figure 5.5: Distributed policy evaluation in a dynamic coalition scenario

Chicago Hope hospital, the Web service *PatientInfo-WS* delegates her request to the *MedRecords-WS* service of the CH-domain. In order to retrieve Mr. Geiger’s data, Kerry Weaver needs to be granted access rights on the Chicago Hope medical database. In the following, we assume a loosely coupled federation, so that access control is performed by means of a distributed policy evaluation process.

Figure 5.5 illustrates extracts of the policy repositories of the three domains and the trust relationships which are defined through these access control rules. To keep the example simple, we disregard further conditions like the restriction that only medical records of patients who



agreed publishing their data within the federation would be accessible. The privilege to access the medical records of patients of the Chicago Hope hospital is given by the permission to execute a query statement on the hospital's database. Kerry Weaver's request that is forwarded to the *MedRecords-WS* instance of the Chicago Hope hospital cannot be granted based on a CH-local policy evaluation. However, the policies contain rules for the cross-domain assignments of roles: Rule  $CH_{1,2}$  states that chief physicians of SH are allowed to delegate the role CH.ProjectMember. This delegation permission is applied by the physician Bob Kelso who assigns the role to the role SH.CoopPhysician of the Sacred Heart hospital, which is represented by the generated assignment  $CH_{1,3}$ . Through  $SH_{1,2}$  this role in turn is assigned to chief physicians of the Cook County General. Because of rule  $CCG_{1,1}$ , the CCG.ChiefPhysician role is granted to Kerry Weaver. Thus, the rules specify the delegation chain

$$CCG.Kerry\ Weaver \rightarrow_R CCG.ChiefPhysician \rightarrow_R SH.CoopPhysician \rightarrow_R CH.ProjectMember$$

If Kerry Weaver tries to read the medical record of a patient housed at the Chicago Hope hospital, this authorization chain has to be traversed in reverse order. Therefore, as described above, the first step is to determine the roles of foreign domains which are senior roles of CH.ProjectMember. In our simplified scenario, this applies to SH.CoopPhysician. As Kerry Weaver is not granted the role based on the local policies, the policy repository of the Sacred Heart hospital is evaluated. For that purpose, the delegation service of SH is invoked, querying whether Kerry Weaver possesses the role SH.CoopPhysician. Again, this cannot be answered SH-locally, so that distributed policy enforcement is initiated at the CCG domain. Due to  $CCG_{1,1}$ , distributed authorization succeeds at the CCG domain and, thus, Kerry Weaver is allowed to execute the *MedRecords-WS* of the Chicago Hope domain and therewith also the service *PatientInfo-WS*.

## 5.4 Caching of Authorization Paths

The described policy evaluation strategy conforms to a depth first search (DFS) on the distributed role assignment graph. Thus, in the worst case, the complete collaboration network has to be analyzed sequentially. If  $m$  is the number of domains in the collaboration network and  $n$  the maximum number of roles administered by any domain, then  $O(m^2 \cdot n^2)$  is an upper bound for the number of distributed role delegations. The worst case complexity is attained if each role is assigned to each role of the federating domains.

While long execution times might be tolerable for unsuccessful policy enforcements, successful authorizations have to proceed as quickly as possible. Most collaborations can be assumed to span only few organizations. Therefore, delegation chains are likely to be rather short. However, as outlined before, breadth first search (BFS) strategies cannot be realized effectively. Response times of the DFS variant can be reduced substantially through parallelizing the search by asynchronously evaluating the distributed policy repositories. Unfortunately, lower response times then have to be traded for an increase of network traffic.

Both objectives, i.e., low response times and low network traffic can be obtained by caching authorization results of frequently and/or recently used requests. If a cache hit occurs, a depth

first search on the distributed role assignment graph can be avoided. Hence, the use of caches provides three main advantages, as there are reduced bandwidth consumption, reduced server load, and reduced latency. We supplied delegation services with authorization caches, storing entries of the form

$$\langle S \xrightarrow{c_{n+1}} D_n.r_n \xrightarrow{c_n} D_{n-1}.r_{n-1} \dots \xrightarrow{c_2} D_1.r_1 \xrightarrow{c_1} D.r \rangle$$

Such a cache entry represents the delegation chain which asserts that the role  $D.r$  is assigned to subjects that comply with the specification  $S$ . To put it another way, a cache entry represents a path in the distributed role assignment graph, whereby its applicability depends on the combined condition  $\bigwedge_{1 \leq i \leq n+1} c_i$ .

### Creation of Cache Entries

A new entry is added to the cache of the delegation service of domain  $D$  if a positive distributed policy evaluation result has been achieved. The cache entry is created when a requester who complies to the subject specification  $S$  invokes a Web service of  $D$  for which the privileges of role  $D.r$  are required and  $S$  is granted  $D.r$ . Authorization evidence is given in the form of an authorization path as discussed above. In order to store the complete delegation chain as a cache entry, cooperating delegation services have to return the history of distributed policy evaluations. This can contradict the security requirements of loosely coupled federations. In the next section, we present a modified design of caches which meets these concerns.

In the following, we refer to delegation services as *clients* in case they consume information by storing results of distributed authorization checks in their local cache. If delegation services return successful evaluation results which can be cached at collaborating organizations, they act as *servers* in the caching scenario.

### Evaluation of Caches

Caches are evaluated prior to the initiation of a distributed policy evaluation. In the first step of the distributed authorization process, those roles are determined that would authorize the request. Thus, for a request  $e$ , those cache entries are determined that apply to  $e$  and that include one of these roles. Thereby, not only exact matches can be handled. For example, if the role  $D.r'$  authorizes the request  $e$ , cache entries  $\langle S \xrightarrow{c_{n+1}} D_n.r_n \xrightarrow{c_n} D_{n-1}.r_{n-1} \dots \xrightarrow{c_2} D_1.r_1 \xrightarrow{c_1} D.r \rangle$  with the following characteristics are determined:

- $\llbracket S \rrbracket_e = \text{true}$ ,
- $\llbracket \bigwedge_{1 \leq i \leq n+1} c_i \rrbracket_e = \text{true}$ , and
- $D.r \succeq_D^e D.r'$

Cache entries are evaluated according to the employed caching strategy. If no applicable cache entry is found, distributed policy evaluation takes place as described in Section 5.3.2.

### 5.4.1 Caching Strategies

The caching of relevant access control data demands for caching strategies providing strong cache consistency. Authorization must not succeed based on outdated, i.e., invalidated access control information. Cao and Özsu (2002) give an overview of strong consistency Web caching techniques. In the following, we describe three Web caching strategies which can be applied in the authorization scenario, namely client validation, server invalidation, and lease based approaches.

#### 5.4.1.1 Client Validation

Employing client validation, client-delegation services have to ensure the validity of cached entries. In case a cache hit occurs, the cache entry is validated before authorization is granted. Let  $\varepsilon = \langle S \xrightarrow{c_{n+1}} D_n.r_n \xrightarrow{c_n} D_{n-1}.r_{n-1} \dots \xrightarrow{c_2} D_1.r_1 \xrightarrow{c_1} D.r \rangle$  be a cache entry that applies to the request  $e$ . The delegation service of  $D$  evaluates the validity of  $\varepsilon$  by checking the  $D$ -local policies, whether there is still a valid assertion of the form  $[D_1.r_1 \rightarrow_R D.r']_{c_1} I$  with  $D.r' \succeq_D^e D.r$ . Subsequently, the evaluation is continued at the delegation service of  $D_1$  for verifying the next extract of the authorization path, i.e.,  $D_2.r_2 \xrightarrow{c_2} D_1.r_1$ . Thus, the individual assertions are iteratively checked. Verification succeeds if each link of the delegation chain is still valid. Otherwise, the cache entry is removed from the local cache and further applicable entries of the cache are evaluated. Usual distributed policy evaluation proceeds if no applicable valid cache entry is found.

Client validation allows to reduce policy evaluation costs significantly because exhaustive evaluations of the distributed role assignment graph can be avoided. The caching of authorization paths differs from common Web caching scenarios: In general, the location of the requested Web content remains unchanged and performance is enhanced by reducing the transferred data volume. In our case, execution time is saved as the validation of authorization paths corresponds to a goal-oriented “walk” through the distributed role assignment graph, rather than a complete analysis of the graph in the worst case. Thus, lookup-time of distributed policy evaluation is reduced.

According to the above description, a cache entry represents a complete delegation chain. This design enables the efficient validation of cache entries, but demands for the cooperating organizations to agree on the exchange of access control information. If the cooperating organizations trust each other, this assumption might hold. In dynamic coalitions, the information flow is kept at a minimum by caching authorization path fragments of the form  $\langle S \rightarrow D_1.r_1 \xrightarrow{c_1} D.r \rangle$  at domain  $D$ . Because of the ownership-paradigm, the assignment  $[D_1.r_1 \rightarrow_R D.r]_{c_1} I$  is contained in the repository of  $D$  and the dissemination of security relevant data is confined. The complete authorization path is restored iteratively through a goal-oriented search starting at  $D_1$ . The delegation service of  $D_1$  either authorizes the request through a local policy evaluation, determines the subsequent fragment of the authorization path in its local cache (i.e.,  $\langle S \rightarrow D_2.r_2 \xrightarrow{c_2} D_1.r_1 \rangle$ ), or – in the worst case – triggers distributed policy enforcement.

### 5.4.1.2 Server Invalidation

Using server invalidation, server-delegation services have to inform adjacent clients in case of policy updates that would invalidate cache entries of clients. Therefore, server-delegation services have to log the request type and the URL of client-delegation services which received positive authorization responses. Modifications of policies must not be fixed before all affected cache entries have been invalidated and the invalidation has been acknowledged by the clients. Consequently, this approach is vulnerable regarding the unreachability of services, e.g., because of network failures. Malicious nodes can even misuse this shortcoming for attacks. These drawbacks clearly disqualify server invalidation for being applied in highly dynamic federations.

One further disadvantage of server invalidation is its limited scalability. Server-delegation services have to log requests in order to notify clients in case of policy updates. This is not considered to be a crucial drawback, however, as for many use cases the collaboration networks would be of manageable size. The advantage of server invalidation compared to client validation is its low bandwidth consumption and latency, as cache hits lead to the immediate authorization of requests without demanding for further validation checks.

### 5.4.1.3 Lease-based Approach

In 1989, Gray and Cheriton introduced the lease-based caching approach which constitutes a compromise between client validation and server invalidation. Leases are contracts between client and server-delegation services. Servers assert not to modify the administered access control policies as long as leases are valid. After the leases have expired, the duty to ensure the validity of cache entries is shifted to the clients. When updating policies, servers have to wait until each client has acknowledged the invalidation of cache entries or, in case of any of them being unreachable, until the respective leases have expired. Consequently, the lease-based approach is parameterizable by means of the validity periods of leases. Setting them close to zero results in a behavior similar to client validation. On the other hand, server invalidation is approximated when using long lease periods. Thus, depending on the parametrization, the pros and cons of client validation and server invalidation apply to lease-based caching, too.

## 5.4.2 Experimental Results

To the best of our knowledge there do not exist any standardized benchmarks for measuring and comparing the performance of access control information caching. Therefore, we developed test cases which illustrate the performance of distributed policy enforcement depending on the size of the collaboration network and the employed caching strategy. Performance is measured as system throughput, i.e., the number of authorization checks that can be performed per time unit. Another criteria is the network load, denoting communication costs. In a homogeneous network, the time consumption of sequential distributed authorization is approximately proportional to the number of messages that are exchanged among collaborating delegation services. Server invalidation provides the best performance, as no messages are exchanged in case of cache hits.

A collaboration network can be classified by means of the number of outgoing connections

per domain (*degree*  $d$ ) and the maximum delegation depth  $h$ . The outgoing degree specifies the number of domains direct collaborations are established with – through some of the partner members being granted local access rights (and potentially vice versa). The (maximum or mean) delegation depth denotes the number of trust domains a delegation chain spans (at maximum or mean).

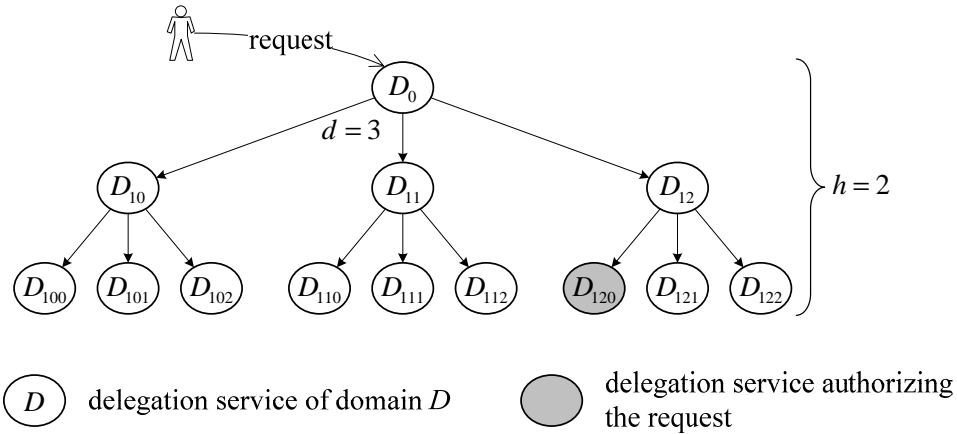
In general, collaboration networks are represented as directed graphs. In our experiments, we analyzed collaboration networks that constitute balanced  $d$ -ary trees of height  $h$  as illustrated in Figure 5.6(a). That is, we are looking at federations from the point of view of one domain that is represented by the root node. We posted requests to this node, measuring the time needed to perform access control. Thereby, we differentiate between requests being authorized locally and those requiring distributed policy evaluation. In the latter scenario, positive access rules are distributed among the policy repositories of the remaining delegation services. In the worst case, authorization requires the complete tree to be examined. Thus,  $O(d^h)$  constitutes an upper bound for the complexity.

The experiments were performed on a cluster of ServiceGlobe installations running on HP ProLiant BL20p Blade server systems with 2.8 GHz Intel Xeon processors. For each test scenario, we measured the performance of authorization when no caching, client validation, and server invalidation were used. Benchmark results are listed relative to the performance measures of server invalidation. The performance of the lease-based approach depends on the expiration period of leases and resides between the performance of client validation and server invalidation. Therefore, performance measures for this caching technique were not conducted separately. On average, the execution of a delegation service lasted 650 ms. About 30 % of this time are required for local policy evaluation and cache examination (with caches being realized as XML documents), while the predominant amount is needed for service loading and initialization.

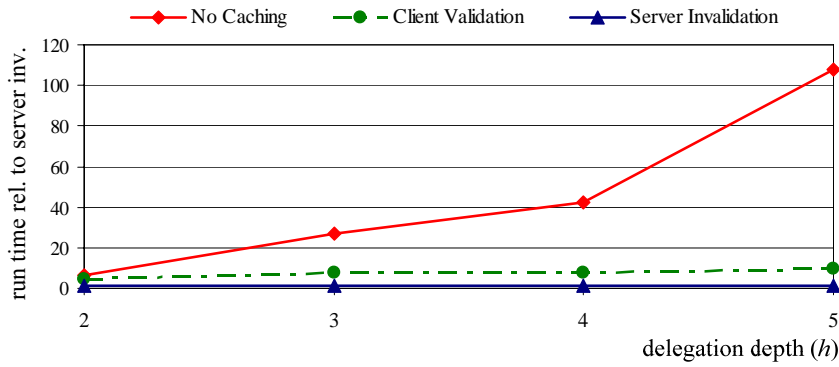
#### 5.4.2.1 Variation of the Network Dimensions

The parameter  $d$  and the maximum delegation depth  $h$  determine the complexity of the tree-like collaboration network. We performed experiments varying both dimension parameters. Figure 5.6(b) illustrates the results for the variation of  $h$ , when only one positive access rule exists in one of the undermost policy repositories. As expected, the execution time of the standard policy enforcement strategy approximates an exponential curve. In case of cache hits, client validation is linear with respect to  $h$  (and invariant with respect to  $d$ ), while server invalidation requires constant time. Already at a depth of  $h = 4$ , execution times without caching are more than five times higher compared to the results of client validation and more than 40 times higher compared to those of server invalidation.

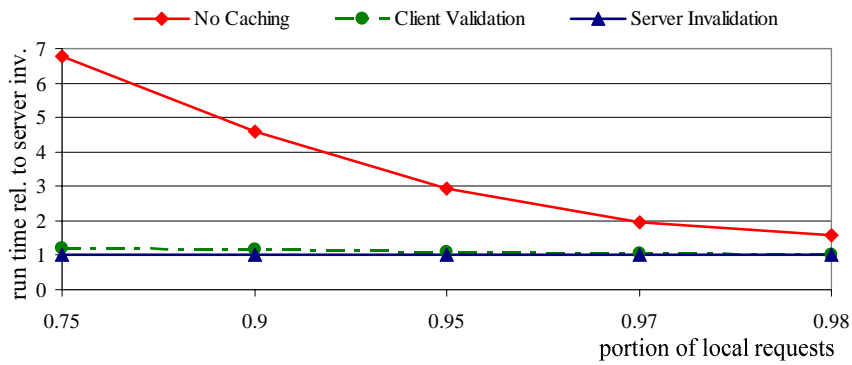
In order to measure cache reorganization costs, we also performed tests assuming a dynamic rights management. For that purpose, a varying number of positive rules were inserted into the graph and some of them were modified regularly. When increasing the update ratio, both curves converge to the non-optimized case. Nevertheless, using server invalidation, policy enforcement times are still lower compared to the usage of client validation.



(a) In the experiments, collaboration networks are modeled as  $d$ -ary trees



(b) Variation of the delegation depth



(c) Variation of the request characteristics

Figure 5.6: Experimental settings and results

### 5.4.2.2 Variation of the Request Characteristics

Figure 5.6(c) illustrates the results of a benchmark varying the relation of local and distributed policy enforcements. The proportion of requests that are evaluated locally is given by the parameter  $p$ . We simulated a static collaboration network with  $d = 2$  and a maximum delegation depth of  $h = 5$ . Thus, a network of 63 collaborating organizations was simulated.

In our experiments, 50 different request types were simulated with  $p \cdot 50$  positive access rules being inserted in the topmost policy repository, i.e., a portion of  $p$  requests can be authorized locally. The  $(1 - p) \cdot 50$  access rules that apply to the remaining part of the request types were inserted in the other repositories according to a Zipf distribution. When sorting the levels of the collaboration network according to the frequency with which access rights are assigned to them, Zipf's law states that the frequency of rights being assigned to a level  $l$  ( $\text{frq}(l)$ ) is inversely proportional to its ranking following a power law:  $\text{frq}(l) \sim 1/l^\alpha$ . Typically,  $\alpha \in [0, 1]$ . A uniform distribution is modeled through  $\alpha = 0$ , while a highly skewed distribution is achieved at  $\alpha = 1$ . By setting  $\alpha = 0.85$ , we simulated a scenario with the predominant part of requests being authorized after a few delegation steps, while only some require the enforcement of policies of the undermost policy repositories. For each value of  $p$ , 2000 requests were posted to the delegation service of the root domain and the average evaluation time was measured. The 2000 requests are separated into clusters of those requiring local policy evaluation and those requiring distributed policy evaluation with a ratio of  $p$  to  $(1 - p)$  – analogous to the distribution of access rights. Adamic and Huberman (2002) showed that Zipf-like distributions are useful for modeling the request characteristics of many Web applications. Therefore, we modeled the distribution of request types by a Zipf distribution with  $\alpha = 0.85$ .

In contrast to the caching of arbitrary Web content, the space requirements for the caching of authorization paths can be estimated in advance quite well. Thus, cache replacement strategies are of minor interest and were not considered in these experiments. In many real-world applications,  $p$  is assumed to be close to 1. Figure 5.6(c) illustrates the performance measurements relative to the results for server invalidation. As the presented experimental results show, response times for requests requiring distributed policy enforcement can be reduced significantly by the use of authorization caches.

## 5.5 Application Scenarios

### 5.5.1 Support of Loosely and Tightly Coupled Federations

The presented authorization framework supports both loosely and tightly coupled federations. A characteristic of loosely coupled networks is that cooperating domains retain their authorization autonomy. Access rules applying to roles or resources of domain  $D$  are solely administered by the security framework of  $D$  and not mirrored elsewhere within the network. Thus, domains preserve the decisive control over their resources and can autonomously terminate interconnections with other domains by revoking the applicable rules from their policy repository.

This paradigm also has to be taken into account when employing caches in dynamic coalitions. In order to support ad hoc networks with dynamically changing trust relationships, we

presented an alternative design of cache entries. Cache entries then only represent succeeding steps of the validation process but not complete paths. In case subsequent steps are stored in the caches of the cooperating domains, the modified design shows similar performance benefits as the original design of cache entries. Both approaches support the efficient goal-oriented revalidation of repeated requests.

In contrast to loosely coupled systems, tightly coupled networks are typically based on long-term and stable trust relationships. An example for a tightly coupled federation is a holding company and the relationships to its affiliates. Processes can be rationalized by passing the management functionality to the holding. Therefore, the affiliated companies give up the authorization autonomy over the shared operating resources. Access control is then performed by the delegation service of the holding company or a dedicated authorization authority. Following this design guideline, no distributed authorization is carried out within the collaboration network. Instead, access control for shared resources is performed by a local policy evaluation point at the central authority, so that access control for tightly coupled systems can be realized efficiently. Hence, the decision to either realize a loosely coupled or a tightly coupled network is also a choice between authorization autonomy and efficiency.<sup>2</sup> Nevertheless, as shown in the previous section, higher policy evaluation costs for loosely coupled federations can be compensated to some extent by adequate caching techniques.

### 5.5.2 Treating Revocations During (Long-lasting) Transactions

Another important issue is the interaction of the authorization system with (distributed) transaction management systems. Before fixing the effects of a transaction, all subtasks have to acknowledge their ability to terminate correctly. Usually, the reliable execution of a transaction is supervised using the two-phase-commit protocol. For Web service scenarios, the two-phase-commit protocol can be implemented by means of the WS-Coordination, WS-BusinessActivity, and in particular the WS-AtomicTransaction framework (described by Cabrera et al. (2005a,b,c)). One potential reason for the abort and rollback of a transaction can be that required authorizations have been revoked in the meantime. This can be realized by integrating authorization checks into the transaction workflow and by performing access control at the workflow layer as motivated in Chapter 3. Prior to the start of a transaction, authorization proceeds as described above. Before the services commit, access control is performed once again, e.g., by verifying the authorization path that was found during the initial authorization check. If authorization fails, the delegation service sends an abort message to the coordinator, initiating a rollback of the transaction.

## 5.6 Related Work

Several security systems supporting inter-organizational collaboration networks have been proposed in the past, e.g., PolicyMaker and KeyNote presented by Blaze et al. in 1996 and in 1999, respectively. These architectures are also called trust management systems as with them dynamic

---

<sup>2</sup>Confer Figure 1.2 on page 4.



coalitions – i.e., networks of trust – can be established. In a trust management system, security policies are specified by means of security credentials like digital certificates. That is, credentials represent delegations of trust by binding public keys to authorizations. In contrast to this, traditional certificates bind keys to names and identities. In PolicyMaker, authorizations are inferred by the evaluation of delegation chains, with the algorithms being presented and analyzed by Blaze et al. (1998b). While in PolicyMaker applications had to realize significant parts of the policy evaluation process, usability, in particular, the integration into existing software systems has been improved by KeyNote. Blaze et al. (1999) give an overview over trust management systems which had a significant impact on the design of upcoming distributed security infrastructures. Further well known trust management system are REFEREE presented by Chu et al. (1997), DL presented by Li et al. (2003a, 2000), the Simple Distributed Security Infrastructure (SDSI) by Rivest and Lampson (1996), and the Simple Public Key Infrastructure (SPKI) proposed by Ellison et al. (1999). The two last mentioned projects were later on combined and led to the joint research architecture SPKI/SDSI presented by Clarke et al. (2001). We borrow from the early trust management systems the basic idea of delegating trust (i.e., privileges) to entities of collaborating domains. In the same way as trust is assigned to credentials, we delegate authorizations by means of cross-domain role assignments. By employing scalable RBAC schemes for distributed applications, our approach is also usable for large scale systems. Though our approach is generic, it is particularly feasible for Web service applications. It can be realized by state of the art Web services technology and languages like XACML for authorization and WS-Security and SAML for secure messaging, in particular authentication.

Cohen et al. (2002) and Tolone et al. (2005) provide an overview over access control schemes that are suitable for realizing inter-organizational federations. As these surveys show, RBAC models like the one we employ in our authorization system are well applicable for such scenarios. Models and architectures of distributed role based access control profiles have, for example, been proposed by Barka and Sandhu (2000b), Demurjian et al. (2001), Phillips et al. (2002a), and Zhang et al. (2003). Flexible trust negotiation models have been presented by Li et al. (2002, 2003b). Our access control architecture is closely related to these research projects and illustrates how basic concepts addressed by them can be realized in modern service-oriented software architectures. Freudenthal et al. (2002) also employ a distributed RBAC scheme. They present a publish-and-subscribe algorithm for countering the costs and the complexity of the distributed policy evaluation process. Their approach is closely related to server invalidation caching techniques. As our comparison of caching strategies shows, server invalidation is not reliable for highly dynamic and/or large scale coalitions, since it is vulnerable with regard to the unreachability of domains. Instead, we recommend the use of client validation. For the caching of access control results we used and adapted caching techniques that are established in the Web caching domain. Cao and Liu (1998) and Cao and Özsu (2002) present surveys of caching strategies providing strong cache consistency which is essential for their applicability in the authorization context. As discussed in Section 5.4, the caching of authorization requests provides optimization capabilities for distributed policy evaluations. Typically, distributed authorizations constitute the smaller portion of requests that have to be handled by the security system. Therefore, the majority of requests can be evaluated locally. Breslau et al. (1998, 1999) and Adamic and Huberman (2002) analyzed the request characteristics of Web applications, coming to the conclusion that

these can be modeled reliably via Zipf distributions. Therefore, we also employed Zipf-like distributions when modeling the request characteristics of our experiments which we presented in Section 5.4.2.

IT infrastructures supporting dynamic and distributed services require new security mechanisms as discussed by Posegga (1999) and Karjoth and Posegga (2000). By means of our research Web service platform ServiceGlobe we showed how the proposed authorization mechanisms can be integrated into upcoming service-oriented architectures. Birrell et al. (1986) present a central authorization service which can be employed in dynamic coalitions. Entities like users can be described by means of their characterizing attributes. If subjects have to be uniquely identifiable – which, for instance, is required to denote the issuers of role delegations –, a federated identity management (FIM) as proposed by Ahn et al. (2004), Ahn and Lam (2005), and Gaedke et al. (2005) can be employed. Hommel and Reiser (2005) compare different FIM initiatives like Liberty Alliance<sup>3</sup> and Microsoft Passport<sup>4</sup>. As stated above, our generic authorization framework is particularly suitable for service-oriented architectures where a federated identity management can, for example, be realized by means of SAML (see [Cantor et al. (2005)]) and WS-Federation (see [Bajaj et al. (2003)]).

Biskup and Leineweber (2001) and Biskup et al. (2003) present the SDSD system (abbreviation for State-Dependent Security Decisions) which is a security infrastructure that allows to regulate access on distributed objects. They present protocols for defining the sequential control of allowed activities on shared resources. Our proposed authorization framework in combination with a federated identity management provides the technological basis for sharing data within Web service federations. The framework is very flexible and allows the realization of tightly as well as loosely coupled systems. Examples for tightly coupled systems are the distributed coalition service registry (DCSR) presented by Mukkamala et al. (2006) and Warner et al. (2005) and the community authorization service (CAS) of the Globus Toolkit presented by Pearlman et al. (2002) and Welch et al. (2003). Tightly coupled federations building upon central authorization services can seamlessly be realized in our architecture by shifting policy enforcement to a trusted authority, thus breaking authorization down to local policy enforcement. The CAS uses a push model for inferring authorizations while our approach relies on a pull model to determine the roles and privileges that are granted to users. A pull model is also used in Akenti that is related to the X.509 certification technique as presented by Thompson et al. (2003, 1999). Policies, conditions, and attribute statements can be encapsulated in certificates. Akenti allows access control for one resource to be administered by multiple authorities. Yu et al. (2003) propose an automated mechanism for establishing trust using digital credentials. In our access control model, trust propagation is supported by means of role delegations. Hence, these approaches can be combined well, e.g., by using public key credentials to represent roles.

Throughout this chapter, we described the functionality of our authorization framework by use of a simplified example from the healthcare domain. Bhatti et al. (2005) elaborate on the security requirements of e-health applications – e.g., with regard to the reliable exchange of clinical

---

<sup>3</sup>Project page: <http://www.projectliberty.org/>

<sup>4</sup>Project page: <http://www.passport.net>

data records between physicians of cooperating hospital departments. Networks of cooperating autonomous partners can be set up in many fields of applications, including e-commerce where enterprises tend to realize inter-organizational value creation chains. Moreover, the ubiquitous computing paradigm is more and more adopted by e-science applications to interconnect research communities. Phillips et al. (2002b) present example scenarios of international coalitions for managing disasters, humanitarian relief and terrorist incidents. Such scenarios demand for dynamic coalitions as the cooperating organizations are partners during such crisis but can be adversaries in another, thus requiring flexible ways to define and revoke trust relationships in an ad hoc manner.

## 5.7 Conclusion

Because of standardized interface descriptions and communication protocols, Web services provide the technological basis for realizing inter-organizational workflows and for sharing data in collaboration networks. Depending on the trust relationships of cooperating organizations, tightly and loosely coupled federations can be differentiated. Therefore, in order to enable the sharing of data in dynamic coalitions, a capable and flexible security system is required.

In this chapter, we presented the model and architecture of our authorization framework supporting distributed service compositions. The underlying policy model is based on a distributed RBAC scheme which makes this approach feasible for large scale applications. Due to the interplay of local and distributed policy enforcements, authorization autonomy of cooperating domains is preserved. Hence, the proposed techniques are suitable both for tightly and loosely coupled federations. Tightly coupled federations demand for stable trust relationships among the participating organizations, allowing access control to be performed efficiently by means of local policy evaluations at a central authorization service.

A characteristic of dynamic service coalitions is that trust relationships among the cooperating partners can change. Therefore, the distributed policy evaluation strategy has to preserve the authorization autonomy of the individual organizations. Distributed authorization is less efficient than centralized access control in tightly coupled systems. The performance of the proposed policy enforcement mechanism is improved through the use of adequate caching techniques. Cache entries represent positive authorization results of recently or frequently posted requests. We presented three caching approaches providing the required strong cache consistency. Our recommendation for highly dynamic federations with frequent policy updates and unreliable trust relationships is to use client validation. Employing this strategy, cache hits are validated by means of goal-oriented walks through the distributed role assignment graph. The benefits of caching compared to the traditional distributed policy evaluation mechanism comprise reduced network traffic and reduced latency. An interesting future research direction is the caching of negative authorization results. This would help to reduce network load caused by illegitimate requests, in particular with regard to potential denial of service attacks.



# Conclusion

---

Security for service-oriented architectures is a relatively new but nonetheless fundamental topic. In this thesis, we proposed a package of authorization techniques providing flexible access control for service-oriented IT infrastructures, whereby efficiency of the presented approaches constitutes a cross-cutting concern. In this regard, we presented a security engineering approach which supports the reliable integration of applications – in particular database systems – into service-oriented middleware systems. Thereby, the integration was addressed both, for intra- and inter-organizational business processes. In particular, distributed service compositions are supported by our flexible authorization scheme for dynamic coalitions.

In Chapter 3, we presented our approach for optimizing the access control of Web service workflows and generic composite applications. The basic idea is to shift access control to the layer of the composite application, thus, avoiding repeated and potentially redundant authorization checks by the autonomous sub-activities. The optimization relies on the combination of the sub-activities' security policies and the preparation of a consolidated view onto the access control configuration of composite applications. Based on an expressive access control model, we presented a generic formal policy consolidation strategy and respective algorithmic solutions. In our model, entities are specified by means of attributes. For instance, subjects can be characterized by their age, profession skills and so on. This is particularly necessary to ensure scalability of service-oriented infrastructures, where identity-based access control is often inadequate. Hence, the proposed approach supports the optimization of business processes which, for example, are realized as service compositions. Furthermore, considering the execution of Web service transactions, the early evaluation of requests helps avoiding transaction rollbacks and costly compensating transactions. Without our optimization, these effects can be caused by ultimately unauthorized requests which lead to aborts at later phases of the execution.

Policy consolidation is also an integral part of the implementation of secure service interfaces. Security vulnerabilities and risks for integrated applications are avoided by realizing the principle of least privilege. Many enterprise services rely on database backends for providing

their functionality. In Chapter 4, we presented our security engineering approach for the specification and implementation of secure service interfaces for database systems. Using our approach, the principle of least privilege can be ensured for database services. Thus, service developers are relieved from the burden of implementing and verifying the security functionality by hand. This allows them to concentrate their efforts on the realization of the application logic.

As proof of concept, the presented authorization concepts have been integrated into our research Web service platform ServiceGlobe. ServiceGlobe is based on common Web service standards like SOAP, WSDL, and UDDI. These specifications constitute the technological basis of service interoperability, enabling inter-organizational Web service compositions. From the security viewpoint, mechanisms for the delegation of trust across domains and the enforcement of access control in virtual organizations are needed. As presented in Chapter 5, these issues are addressed by the authorization framework of ServiceGlobe that realizes a distributed role based access control scheme. It allows assigning privileges and roles to subjects of organizations that participate in the ServiceGlobe federation. Cross-domain assignments and role delegations enable dynamic service coalitions in which access control is performed as an interplay of local and distributed policy evaluation steps. Using this approach, cooperating organizations retain their authorization autonomy and no central policy enforcement point needs to be realized. In order to optimize distributed policy enforcements and, thus, to make this approach suitable for large-scale dynamic coalitions, we devised access control caching techniques. The proposed caching strategy realizes a goal-oriented validation of authorization proofs in the distributed role assignment graph. By avoiding the dissemination of sensitive access control information, data can be shared reliably within dynamic virtual organizations.

In ongoing work, the presented security principles and techniques are enhanced. Optimizing the access control of composite applications was done from the single-user / single-role perspective. Therefore, this approach is of particular relevance for job-specific business processes. Currently, we are developing efficient enforcement strategies for partial authorizations that rely on compact labels for the representation of access paths and the evaluation of authorizations. Furthermore, future research deals with the efficient and reliable execution of multi-user workflows. In such workflows, sub-processes can be executed by varying subjects, whereby the independent tasks should be performed by experts in order to obtain good work results. The optimization objective can be described by the term *quality of execution*, relating to quality of service. Concerning quality of service, appropriate service instances are selected to achieve good results. Quality of execution on the other hand stands for the process of assigning users to tasks, aiming at the reliable execution of the overall business process and good load balancing characteristics. Obviously, besides scheduling, security is a key factor for this optimization issue, for which the spadework has been presented in this thesis.

---

## Graphical Workflow Notation

---

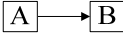
### Workflow States

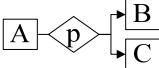
 Start state       End state


### Sub-activities, Resources, and Policy Files


 (Composite) application  
 Database object       Policy file

### Control flow Components

 SEQUENCE: A before B

 SWITCH: Depending on condition p, either B or C is executed

 Merge / synchronization

 Database interaction

## APPENDIX B

---

# Probabilistic Performance Estimation of Policy Comparisons

---

In the following, we present a probabilistic complexity evaluation<sup>1</sup> for the algorithm implies which is shown in Figure 3.10 on page 35. In each iteration of the recursive algorithm implies, a number of remainder subterms is generated. These are evaluated against the remaining conjunctive subterms of  $T'$  in subsequent iterations. Let  $t$  be a conjunctive subterm of  $T$ . We make the following estimation:

The likelihood of  $t$  being subsumed by any  $t'_1$  of  $T'$  or not matching with  $T'$  at all is  $(1 - p) \in [0, 1]$  (cases 1 and 2 in Section 3.4.2). Then,  $p$  is the probability of  $t$  overlapping partially with any  $t'_1$ , i.e.,  $(t \wedge \neg t'_1) \neq \text{false}$  (case 3).

If  $k$  is the number of conjunctive subterms of  $T$ , then  $(1 - p) \cdot k$  terms are either subsumed by any conjunctive subterm of  $T'$  or do not match with  $T'$  at all. Consequently,  $p \cdot k$  terms are subdivided into remainder terms. We estimate the number of remainder terms through the upper bound  $2l$ . Again, the relation of those terms that are not subdivided into subterms to those being split is  $(1 - p)/p$ . The recursion terminates at a maximum depth of  $(k' - 1)$ . We estimate the number of predicate conjunctions that are evaluated in the best case ( $p = 0$ ) and in the worst case ( $p = 1$ ). The following variables are used:

- $k$  The number of conjunctive terms of  $T$ .
- $k'$  The number of conjunctive terms of  $T'$ .
- $l$  The number of distinguished attributes.
- $p$  The likelihood for  $t$  and  $t'_1$  overlapping partially.

---

<sup>1</sup>See, for example, [Ausiello et al. (2003)], chapter 9.



## Reasoning

At each recursion level  $r$ , where  $\#Terms$  terms are compared with  $T'$ ,  $((1-p) \cdot \#Terms)$  terms are subsumed by one of the  $k' - r$  terms of  $T'$  or do not match with  $T'$ . The first  $r$  subterms of  $T'$  need not be considered due to comparisons at earlier stages. That leads to

$$(1-p) \cdot \#Terms \cdot \underbrace{(k' - r)}_{\substack{\text{for comparisons} \\ \text{with up to } (k' - r) \text{ terms}}}$$

comparisons of predicate conjunctions. The remaining  $(p \cdot \#Terms)$  terms are evaluated in subsequent iterations. Thus,

$$p \cdot \#Terms \cdot \left( \underbrace{1}_{\substack{\text{each term is compared with } t'_r}} + [\text{costs of recursive evaluations}] \right)$$

comparisons of predicate conjunctions will be performed.

Initially, i.e., at recursion level 0,  $\#Terms = k$ . The number of branches generated per recursion are  $2l$  at most. Hence, the number of comparisons can be estimated by

$$\begin{aligned} & \underbrace{(1-p)k \cdot k'}_{\substack{\text{first part matches with} \\ \text{one of the } k' \text{ terms of } T'}} + \underbrace{pk}_{\substack{\text{second part is} \\ \text{split through subtract}}} \cdot \left( 1 + \underbrace{(1-p)2l \cdot (k' - 1)}_{\substack{\text{first part matches with one} \\ \text{of the rem. } (k' - 1) \text{ terms of } T'}} + \underbrace{p2l}_{\substack{\text{second part is} \\ \text{split again}}} \right) \\ & \left( 1 + \underbrace{(1-p)2l \cdot (k' - 2)}_{\substack{\text{match } \dots}} + \underbrace{p2l}_{\substack{\text{split } \dots}} \cdot \left( \dots \left( 1 + \underbrace{(1-p)2l(k' - (k' - 1))}_{-2l} + p2l \right) \right) \right) \end{aligned} \quad (\text{B.1})$$

$$= (1-p)kk' + pk(1 + (1-p)2l(k' - 1) + p2l(1 + (1-p)2l(k' - 2) + p2l(\dots(1 + (1-p)2l(k' - (k' - 2)) + p2l(1 + 2l)) \dots))) \quad (\text{B.2})$$

$$\begin{aligned} & = (1-p)kk' + pk(1 + p2l + (p2l)^2 + \dots + (p2l)^{k'-2} + (1-p)2l(k' - 1) \\ & \quad + (1-p)p(2l)^2(k' - 2) + \dots + (1-p)p^{k'-3}(2l)^{k'-2}(k' - (k' - 2)) \\ & \quad + p^{k'-2}(2l)^{k'-1}) \end{aligned} \quad (\text{B.3})$$

Because of  $p^{k'-2}(2l)^{k'-1} = (p2l)^{k'-2} \cdot ((1-p)(k' - (k' - 1))2l + p2l)$

$$= (1-p)(k' - (k' - 1))p^{k'-2}(2l)^{k'-1} + (p2l)^{k'-1} \text{ we receive:}$$

$$(\text{B.3}) = (1-p)kk' + pk \left( \sum_{i=1}^{k'-1} (1-p)p^{i-1}(2l)^i(k' - i) + \sum_{i=0}^{k'-1} (p2l)^i \right) \quad (\text{B.4})$$

$$= (1-p)kk' + (1-p)k \left( \sum_{i=1}^{k'-1} (p2l)^i(k' - i) \right) + pk \left( \sum_{i=0}^{k'-1} (p2l)^i \right) \quad (\text{B.5})$$

We substitute  $p2l$  with  $x$  and receive:

$$(B.5) = (1-p)kk' + (1-p)k \left( \sum_{i=0}^{k'-1} x^i(k'-i) - k' \right) + pk \left( \sum_{i=0}^{k'-1} x^i \right) \quad (B.6)$$

We then use the series formulae

$$\sum_{i=0}^{k'} x^i = \frac{x^{(k'+1)} - 1}{x - 1}$$

and

$$\sum_{i=0}^{k'} (k' - i)x^i = \frac{x^{(k'+1)} - (k' + 1)x + k'}{(x - 1)^2}$$

and rewrite equation (B.6)

$$(B.6) = (1-p)kk' + (1-p)k \left( \frac{x^{k'} - k'x + (k' - 1)}{(x - 1)^2} - k' \right) + pk \frac{x^{k'} - 1}{x - 1} \quad (B.7)$$

Resubstituting  $x$  with  $p2l$  results in:

$$(B.7) = (1-p)kk' + (1-p)k \left( \frac{(p2l)^{k'} - k'p2l + (k' - 1)}{(p2l - 1)^2} - k' \right) + pk \frac{(p2l)^{k'} - 1}{p2l - 1} \quad (B.8)$$

## Lower Bound and Upper Bound Estimates

We can estimate lower and upper bounds for the number of comparisons of predicate conjunctions through setting the likelihood  $p$  to 0 and 1, respectively. Thus, from estimation (B.8), we can infer the lower bound

$$\Omega(kk' - k)$$

An estimate for the upper bound is

$$O\left(\frac{k \cdot (2l)^{k'} - 1}{2l - 1}\right)$$

---

## Policy Representation

---

In the following, we sketch how access control rules specified in our formal policy model are represented in XACML syntax. The XACML fragments shown in this chapter are simplified in order to ease their readability. For instance, function and type identifiers are abbreviated and attributes that are of minor relevance for the conceptual presentation are omitted. Further details about the syntax of XACML are provided by Moses (2005).

**Representation of Attribute Comparisons** As shown in Section 3.2, subjects, objects, and actions are represented by means of attribute comparisons given in disjunctive normal form. Attribute comparisons are predicates of the form (*attribute-identifier*  $\circ$  *constant*), where *attribute-identifier* is an element in  $Attr = S-Attr \cup O-Attr \cup A-Attr \cup E-Attr$ . In XACML, predicates are represented via *attribute matches*. In particular, predicates defined over elements in *S-Attr* are represented by `SubjectMatch` elements. Analogously, object predicates are represented by `ResourceMatch` elements and action predicates by `ActionMatch` elements. For example, the predicate

$$(role = \text{Physician})$$

with  $role \in S-Attr$  is represented by the following `SubjectMatch` element:

```
<SubjectMatch MatchId="function#role-equal">
  <AttributeValue DataType="Role">
    <Role>
      <RoleName>Physician</RoleName>
    </Role>
  </AttributeValue>
  <SubjectAttributeDesignator DataType="Role"
    AttributeId="role"/>
</SubjectMatch>
```

As another example consider the predicate

$$(column = \text{Hospital-DB.MedicalRecords.Medication})$$

It represents the column Medication of the MedicalRecords table that is part of the Hospital-DB database. The attribute identifier *column* is defined in *O-Attr*. In XACML, the predicate is represented as follows:

```
<ResourceMatch MatchId="function#dbObject-equal">
  <AttributeValue DataType="DBObject">
    <DBObject>
      <Database>Hospital-DB</Database>
      <Table>MedicalRecords</Table>
      <Column>Medication</Column>
    </DBObject>
  </AttributeValue>
  <ResourceAttributeDesignator DataType="DBObject"
    AttributeId="column"/>
</ResourceMatch>
```

In XACML, the type of attributes must be declared explicitly, like Role and DBObject in the examples. Attributes that are not restricted by any predicate (i.e., equal to  $\perp$ ) are not transformed into XACML. In the following, let  $|S-Attr| = l$ ,  $|O-Attr| = l'$ , and  $|A-Attr| = l''$  and constants  $k, k', k'' > 0$ . As introduced in Section 4.3, we differentiate between permission policies, base policies, and role assignment policies. Administrative policies are provided by role delegation and role revocation policies.

## C.1 Permission Policies

Let  $(O, A)$  be a privilege, with

$$O = (o_{1,1} \wedge \dots \wedge o_{1,l'}) \vee \dots \vee (o_{k',1} \wedge \dots \wedge o_{k',l'}) \text{ and}$$

$$A = (a_{1,1} \wedge \dots \wedge a_{1,l''}) \vee \dots \vee (a_{k'',1} \wedge \dots \wedge a_{k'',l''})$$

We represent the privilege  $(O, A)$  by a permission policy of the form:

```
<Policy RuleCombiningAlgId="permit-overrides">
  <Target>
    <Subjects><AnySubject/></Subjects>
    <Resources><AnyResource/></Resources>
    <Actions><AnyAction/></Actions>
  </Target>
  <Rule Effect="Permit">
    <Target>
      <Subjects>
        <AnySubject/>
      </Subjects>
      <Resources>
```

```

    <Resource>
      <ResourceMatch>o1,1</ResourceMatch>
      ...
      <ResourceMatch>o1,l'</ResourceMatch>
    </Resource>
    ...
    <Resource>
      <ResourceMatch>ok',1</ResourceMatch>
      ...
      <ResourceMatch>ok',l'</ResourceMatch>
    </Resource>
  </Resources>
  <Actions>
    <Action>
      <ActionMatch>a1,1</ActionMatch>
      ...
      <ActionMatch>a1,l'</ActionMatch>
    </Action>
    ...
    <Action>
      <ActionMatch>ak'',1</ActionMatch>
      ...
      <ActionMatch>ak'',l''</ActionMatch>
    </Action>
  </Actions>
</Target>
</Rule>
</Policy>

```

## C.2 Base Policies

Via base policies, privileges are assigned to subjects like users or roles. Hence, they correspond to rules  $(S, O, A)$  with  $S = (s_{1,1} \wedge \dots \wedge s_{1,l}) \vee \dots \vee (s_{k,1} \wedge \dots \wedge s_{k,l})$  representing subjects and  $(O, A)$  representing the privilege. As illustrated before, a privilege  $(O, A)$  is expressed by a permission policy. This permission policy is included in a base policy by means of a policy reference as illustrated below:

```

<PolicySet PolicyCombiningAlgId="permit-overrides">
  <Target>
    <Subjects>
      <Subject>
        <SubjectMatch>s1,1</SubjectMatch>
        ...
        <SubjectMatch>s1,l</SubjectMatch>
      </Subject>
      ...
      <Subject>
        <SubjectMatch>sk,1</SubjectMatch>

```

```

    ...
    <SubjectMatch> $s_{k,l}$ </SubjectMatch>
  </Subject>
</Subjects>
<Resources>
  <AnyResource/>
</Resources>
<Actions>
  <AnyAction/>
</Actions>
</Target>
<PolicyIdReference>
  Reference to the XACML permission policy of (O, A), confer [Moses (2005)].
</PolicyIdReference>
</PolicySet>

```

### C.3 Role Assignment Policies

Role assignment policies represent rules of the form

$$(S, (\textit{granted-role} = \text{Granted Role}), (\textit{method} = \text{enable}), c)$$

with  $S = (s_{1,1} \wedge \dots \wedge s_{1,l}) \vee \dots \vee (s_{k,1} \wedge \dots \wedge s_{k,l})$ . Such a rule expresses the assignment of the role `Granted Role` to all subjects that fulfill the specification of  $S$ . The following extract sketches the representation of this rule in XACML-format:

```

<Policy RuleCombiningAlgId="permit-overrides">
  <Target>
    <Subjects><AnySubject/></Subjects>
    <Resources><AnyResource/></Resources>
    <Actions><AnyAction/></Actions>
  </Target>
  <Rule Effect="Permit">
    <Target>
      <Subjects>
        <Subject>
          <SubjectMatch> $s_{1,1}$ </SubjectMatch>
          ...
          <SubjectMatch> $s_{1,l}$ </SubjectMatch>
        </Subject>
        ...
        <Subject>
          <SubjectMatch> $s_{k,1}$ </SubjectMatch>
          ...
          <SubjectMatch> $s_{k,l}$ </SubjectMatch>
        </Subject>
      </Subjects>
    </Target>
  </Rule>
</Policy>

```

```

<Resource>
  <ResourceMatch MatchId="function#role-equal">
    <AttributeValue DataType="Role">
      <Role><RoleName>Granted Role</RoleName></Role>
    </AttributeValue>
    <ResourceAttributeDesignator
      AttributeId="granted-role" DataType="Role"/>
    </ResourceMatch>
  </Resource>
</Resources>
<Actions>
  <Action>
    <ActionMatch MatchId="function#roleAction-equal">
      <AttributeValue DataType="RoleAction">
        <RoleAction>enable</RoleAction>
      </AttributeValue>
      <ActionAttributeDesignator AttributeId="method"
        DataType="RoleAction"/>
    </ActionMatch>
  </Action>
</Actions>
</Target>
<Condition FunctionId="depends on c">
  XACML representation of c.
  The representation of conditions is shown by Moses (2005).
</Condition>
</Rule>
</Policy>

```

## C.4 Role Delegation and Revocation Policies

Role delegation policies define the privilege to delegate roles to other subjects. For instance,

$$((d\text{-role} = \text{CCG.AttendingPhysician}), (method = \text{delegate}))$$

represents the privilege to delegate the role `AttendingPhysician` (of the domain `CCG`). In the same way, role revocation permissions are defined. For example, the privilege to revoke the role `AttendingPhysician` is given by

$$((d\text{-role} = \text{CCG.AttendingPhysician}), (method = \text{revoke})).$$

Role delegation permissions are represented in XACML similar to usual permission policies:

```

<Policy RuleCombiningAlgId="permit-overrides">
  <Target>
    <Subjects><AnySubject/></Subjects>
    <Resources><AnyResource/></Resources>
    <Actions><AnyAction/></Actions>
  </Target>
  <Rule Effect="Permit">
    <Target>
      <Subjects><AnySubject/></Subjects>
      <Resources>
        <Resource>
          <ResourceMatch MatchId="function#role-equal">
            <AttributeValue DataType="Role">
              <Role><RoleName>AttendingPhysician</RoleName></Role>
            </AttributeValue>
            <ResourceAttributeDesignator Issuer="CCG"
              AttributeId="d-role" DataType="Role" />
          </ResourceMatch>
        </Resource>
      </Resources>
      <Actions>
        <Action>
          <ActionMatch MatchId="function#roleAction-equal">
            <AttributeValue DataType="RoleAction">
              <ActionValue><RoleAction>delegate</RoleAction>
            </ActionValue>
            <ActionAttributeDesignator DataType="RoleAction"
              AttributeId="method" />
          </ActionMatch>
        </Action>
      </Actions>
    </Target>
  </Rule>
</Policy>

```

If a legitimate subject delegates a role, a role assignment policy is generated. Correspondingly, in case of a revocation, the respective role assignment policy is deleted, depending on the used revocation scheme (cf. Section 5.2.4). For example, if Kerry Weaver delegates the role AttendingPhysician to John Carter, the following role assignment policy is generated:

$$[(uid = \text{CCG.John Carter}) \rightarrow_R (\text{granted-role} = \text{CCG.AttendingPhysician})]_{\text{true}} \text{CCG.Kerry Weaver.}$$

We represent this assignment by:



```
<Policy RuleCombiningAlgId="permit-overrides">
  <Target>
    <Subjects><AnySubject/></Subjects>
    <Resources><AnyResource/></Resources>
    <Actions><AnyAction/></Actions>
  </Target>
  <Rule Effect="Permit">
    <Target>
      <Subjects>
        <Subject>
          <SubjectMatch MatchId="function:string-equal">
            <AttributeValue DataType="string">John Carter
            </AttributeValue>
            <SubjectAttributeDesignator DataType="string"
              AttributeId="uid" Issuer="CCG"/>
          </SubjectMatch>
        </Subject>
      </Subjects>
      <Resources>
        <Resource>
          <ResourceMatch MatchId="function#role-equal">
            <AttributeValue DataType="Role">
              <Role><RoleName>AttendingPhysician</RoleName></Role>
            </AttributeValue>
            <ResourceAttributeDesignator DataType="Role"
              AttributeId="granted-role" Issuer="CCG" />
          </ResourceMatch>
        </Resource>
      </Resources>
      <Actions>
        <Action>
          <ActionMatch MatchId="function#roleAction-equal">
            <AttributeValue DataType="RoleAction">
              <ActionValue>
                <RoleAction>enable</RoleAction>
                <Issuer>CCG.Kerry Weaver</Issuer>
              </ActionValue>
            </AttributeValue>
            <ActionAttributeDesignator AttributeId="method"
              DataType="RoleAction" />
          </ActionMatch>
        </Action>
      </Actions>
    </Target>
  </Rule>
</Policy>
```



---

## Bibliography

---

- Abadi, M. and L. Lamport (1993). *Composing Specifications*. In ACM Transactions on Programming Languages and Systems (TOPLAS), volume 15, no. 1 pages 73–132.
- Adam, N. R., V. Atluri, and W.-K. Huang (1998). *Modeling and Analysis of Workflows Using Petri Nets*. In Journal of Intelligent Information Systems, volume 10, no. 2 pages 131–158.
- Adamic, L. A. and B. A. Huberman (2002). *Zipf's Law and the Internet*. In Glottometrics, volume 3 pages 143–150.
- Ahad, R., J. Davis, S. Gower, P. Lyngbaek, A. Marynowski, and E. Onuegbe (1992). *Supporting Access Control in an Object-oriented Database Language*. In *Proceedings of the 3rd International Conference on Extending Database Technology (EDBT)*. Springer-Verlag, London, UK, pages 184–200.
- Ahn, G.-J. and J. Lam (2005). *Managing Privacy Preferences for Federated Identity Management*. In *Proceedings of the 2005 Workshop on Digital Identity Management (DIM)*. Fairfax, VA, USA, pages 28–36.
- Ahn, G.-J., D. Shin, and S.-P. Hong (2004). *Information Assurance in Federated Identity Management: Experimentations and Issues*. In *Proceedings of the 5th International Conference on Web Information Systems Engineering (WISE)*, volume 3306 of *Lecture Notes in Computer Science (LNCS)*. Brisbane, Australia, pages 78–89.
- Aho, A. V., R. Sethi, and J. D. Ullman (1986). *Compilers – Principles, Techniques, and Tools*. Addison-Wesley, Reading, MA, USA, 1st edition.
- Aiouche, O. (2005). *Auxiliary BPEL Support for a Java-based Workflow Engine*. Technical report, SAP Research, Sophia Antipolis, France.

- Albutiu, M. C. (2006). *Optimization of Workflow Access Control – Integration of Single User Workflow Execution in Gabriel/Nehemiah*. Technical report, SAP Research, Sophia Antipolis, France.
- Altunay, M., D. Brown, G. Byrd, and R. Dean (2005). *Trust-based Secure Workflow Path Construction*. In *Proceedings of the 3rd International Conference on Service Oriented Computing (ICSOC)*, volume 3826 of *Lecture Notes in Computer Science (LNCS)*. Amsterdam, The Netherlands, pages 382–395.
- Anderson, A. (editor) (2005). *Core and Hierarchical Role Based Access Control (RBAC) Profile of XACML, Version 2.0*, last visited December '06.  
[http://docs.oasis-open.org/xacml/2.0/access\\_control-xacml-2.0-rbac-profile1-spec-os.pdf](http://docs.oasis-open.org/xacml/2.0/access_control-xacml-2.0-rbac-profile1-spec-os.pdf)
- Anderson, A. and H. Lockhart (editors) (2005). *SAML 2.0 Profile of XACML, Version 2.0*, last visited December '06.  
[http://docs.oasis-open.org/xacml/2.0/access\\_control-xacml-2.0-saml-profile-spec-os.pdf](http://docs.oasis-open.org/xacml/2.0/access_control-xacml-2.0-saml-profile-spec-os.pdf)
- Andrews, T., F. Curbera, H. Dholakia, Y. Golland, J. Klein, F. Leymann, K. Liu, D. Roller, D. Smith, S. Thatte, I. Trickovic, and S. Weerawarana (2003). *Business Process Execution Language for Web Services (BPEL4WS), Version 1.1*, last visited December '06.  
<http://www6.software.ibm.com/software/developer/library/ws-bpel.pdf>
- ANSI INCITS 359-2004 (2004). *Role Based Access Control*. American National Standards Institute, Inc. (ANSI), Washington, DC, USA.
- Antonioletti, M., M. Atkinson, A. Krause, S. Laws, S. Malaika, N. W. Paton, D. Pearson, and G. Riccardi (2005a). *Web Services Data Access and Integration – The Core (WS-DAI) Specification, Version 1.0*.
- Antonioletti, M., B. Collins, A. Krause, S. Laws, J. Magowan, S. Malaika, and N. W. Paton (2005b). *Web Services Data Access and Integration – The Relational Realisation (WS-DAIR) Specification Version 1.0*.
- Antonioletti, M., S. Hastings, A. Krause, S. Langella, S. Laws, S. Malaika, and N. W. Paton (2005c). *Web Services Data Access and Integration – The XML Realization (WS-DAIX) Specification Version 1.0*.
- Antonioletti, M., A. Krause, N. W. Paton, A. Eisenberg, S. Laws, S. Malaika, J. Melton, and D. Pearson (2006). *The WS-DAI Family of Specifications for Web Service Data Access and Integration*. In *ACM SIGMOD Record*, volume 35, no. 1 pages 48–55.
- Atluri, V. (2001). *Security for Workflow Systems*. In *Information Security Technical Report*, volume 6, no. 2 pages 59–68.
- Atluri, V. and W.-K. Huang (1996). *An Authorization Model for Workflows*. In *Proceedings of the 4th European Symposium on Research in Computer Security (ESORICS)*, volume 1146 of *Lecture Notes in Computer Science (LNCS)*. Rome, Italy, pages 44–64.

- Atluri, V., W.-K. Huang, and E. Bertino (1997). *An Execution Model for Multilevel Secure Workflows*. In *Proceedings of the IFIP TC11 WG11.3 11th International Conference on Database Security (DBSec)*, volume 113 of *IFIP Conference Proceedings*. Lake Tahoe, California, USA, pages 151–165.
- Atluri, V., W.-K. Huang, and E. Bertino (2000). *A Semantic-based Execution Model for Multi-level Secure Workflows*. In *Journal of Computer Security*, volume 8, no. 1.
- Ausiello, G., P. Crescenzi, G. Gambosi, V. Kann, A. Marchetti-Spaccamela, and M. Protasi (2003). *Complexity and Approximation – Combinatorial Optimization Problems and their Approximability Properties*. Springer, 2nd edition.
- Backes, M., M. Dürmuth, and R. Steinwandt (2004). *An Algebra for Composing Enterprise Privacy Policies*. In *Proceedings of the 9th European Symposium on Research Computer Security (ESORICS)*, volume 3193 of *Lecture Notes in Computer Science (LNCS)*. Sophia Antipolis, France, pages 33–52.
- Backes, M., B. Pfitzmann, and M. Schunter (2003). *A Toolkit for Managing Enterprise Privacy Policies*. In *Proceedings of the 8th European Symposium on Research Computer Security (ESORICS)*, volume 2808 of *Lecture Notes in Computer Science (LNCS)*. Gjøvik, Norway, pages 162–180.
- Bajaj, S., G. Della-Libera, B. Dixon, M. Dusche, M. Hondo, M. Hur, C. Kaler, H. Lockhart, H. Maruyama, A. Nadalin, N. Nagaratnam, A. Nash, H. Prafullchandra, and J. Shewchuk (2003). *Web Services Federation Language (WS-Federation)*.  
<http://www-128.ibm.com/developerworks/library/specification/ws-fed/>
- Barka, E. and R. Sandhu (2000a). *Framework for Role-based Delegation Models*. In *Proceedings of the 16th Annual Computer Security Applications Conference (ACSAC)*. New Orleans, Louisiana, USA, pages 168–176.
- Barka, E. and R. Sandhu (2000b). *A Role-based Delegation Model and some Extensions*. In *Proceedings of the 23rd National Information Systems Security Conference*. Baltimore, MD, pages 101–110.
- Bauder, I. (2006). *Microsoft SQL Server 2005 für Administratoren*. Carl Hanser Verlag, Munich, Germany, 1st edition.
- BEA (2006). *BEA WebLogic Server and WebLogic Express – Programming WebLogic JDBC*, last visited December '06. BEA Systems, Inc.  
<http://e-docs.bea.com/wls/docs81/jdbc/index.html>
- Bell, D. E. and L. J. LaPadula (1973). *Secure Computer Systems: Mathematical Foundations*. Technical Report 2547, volume 1, MITRE, Bedford, MA, USA.
- Bell, D. E. and L. J. LaPadula (1976). *Secure Computer Systems: Unified Exposition and Multics Interpretation*. Technical Report ESD-TR-75-306, MITRE, Bedford, MA, USA.

- Bertino, E., B. Catania, E. Ferrari, and P. Perlasca (2001). *A Logical Framework for Reasoning about Access Control Models*. In *Proceedings of the 6th ACM Symposium on Access Control Models and Technologies (SACMAT)*. Chantilly, Virginia, United States, pages 41–52.
- Bertino, E., E. Ferrari, and V. Atluri (1999a). *The Specification and Enforcement of Authorization Constraints in Workflow Management Systems*. In *ACM Transactions on Information and System Security (TISSEC)*, volume 2, no. 1 pages 65–104.
- Bertino, E., E. Ferrari, F. Buccafurri, and P. Rullo (1999b). *A Logical Framework for Reasoning on Data Access Control Policies*. In *Proceedings of the 1999 IEEE Computer Security Foundations Workshop (CSFW)*. Mordano, Italy, pages 175–189.
- Bettini, C., X. S. Wang, and S. Jajodia (2002). *Temporal Reasoning in Workflow Systems*. In *Distributed and Parallel Databases*, volume 11, no. 3 pages 269–306.
- Bhatti, R., E. Bertino, A. Ghafoor, and J. B. D. Joshi (2004). *XML-based Specification for Web Services Document Security*. In *IEEE Computer*, volume 37, no. 4 pages 41–49.
- Bhatti, R., J. B. D. Joshi, E. Bertino, and A. Ghafoor (2003). *Access Control in Dynamic XML-based Web Services with X-RBAC*. Technical Report 2003-36, CERIAS.
- Bhatti, R., B. Shafiq, M. Shehab, and A. Ghafoor (2005). *Distributed Access Management in Multimedia IDCs*. In *IEEE Computer*, volume 38, no. 9 pages 60–69.
- Biba, K. J. (1977). *Integrity Considerations for Secure Computer Systems*. Technical Report TR-3153, MITRE, Bedford, MA, USA.
- Birrell, A., B. W. Lampson, R. M. Needham, and M. D. Schroeder (1986). *A Global Authentication Service without Global Trust*. In *Proceedings of the IEEE Symposium on Security and Privacy*. Oakland, CA, USA, pages 223–230.
- Bishop, M. (2002). *Computer Security: Art and Science*. Addison-Wesley, Reading, MA, USA, 1st edition.
- Biskup, J. and T. Leineweber (2001). *State-Dependent Security Decisions for Distributed Object-Systems*. In *Proceedings of the 15th Working Conference on Data and Applications Security (DBSec)*, volume 215 of *IFIP Conference Proceedings*. Niagara on the Lake, Ontario, Canada, pages 105–118.
- Biskup, J., T. Leineweber, and J. Parthe (2003). *Administration Rights in the SDSD-System*. In *Proceedings of the 17th Working Conference on Data and Applications Security (DBSec)*. Estes Park, Colorado, USA, pages 149–162.
- Blaze, M., J. Feigenbaum, J. Ioannidis, and A. D. Keromytis (1999). *The Role of Trust Management in Distributed Systems Security*. In *Proceedings of Secure Internet Programming, Security Issues for Mobile and Distributed Objects*, volume 1603 of *Lecture Notes in Computer Science (LNCS)*. pages 185–210.

- Blaze, M., J. Feigenbaum, and A. D. Keromytis (1998a). *KeyNote: Trust Management for Public-Key Infrastructures*. In *Proceedings of the 1998 Security Protocols International Workshop*, volume 1550 of *Lecture Notes in Computer Science (LNCS)*. Cambridge, UK, pages 59–63.
- Blaze, M., J. Feigenbaum, and J. Lacy (1996). *Decentralized Trust Management*. In *Proceedings of the IEEE Conference on Security and Privacy (SP)*. Oakland, CA, USA, pages 164–173.
- Blaze, M., J. Feigenbaum, and M. Strauss (1998b). *Compliance Checking in the PolicyMaker Trust Management System*. In *Proceedings of the 2nd International Conference on Financial Cryptography (FC)*, volume 1465 of *Lecture Notes in Computer Science (LNCS)*. Anguilla, British West Indies, pages 254–274.
- Blobel, B. and K. Pommerening (1997). *Datenschutz und Datensicherheit in Informationssystemen des Gesundheitswesens*. In *IT-Sicherheit*, volume 97, no. 3 pages 2–8.
- Bonatti, P., S. de Capitani di Vimercati, and P. Samarati (2000). *A Modular Approach to Composing Access Control Policies*. In *Proceedings of the 7th ACM Conference on Computer and Communications Security (CCS)*. Athens, Greece, pages 164–173.
- Bonatti, P., S. de Capitani di Vimercati, and P. Samarati (2002). *An Algebra for Composing Access Control Policies*. In *ACM Transactions on Information and System Security (TISSEC)*, volume 5, no. 1 pages 1–35.
- Breslau, L., P. Cao, L. Fan, G. Phillips, and S. Shenker (1998). *On the Implications of Zipf's Law for Web Caching*. In *Proceedings of the 3rd International WWW Caching Workshop*. Manchester, England, pages 1–11.
- Breslau, L., P. Cao, L. Fan, G. Phillips, and S. Shenker (1999). *Web Caching and Zipf-like Distributions: Evidence and Implications*. In *Proceedings of the 18th Annual Joint Conference of the IEEE Computer and Communications Societies (INFOCOM)*. New York, NY, USA, pages 126–134.
- Cabrera, L. F., G. Copeland, M. Feingold, R. W. Freund, T. Freund, J. Johnson, S. Joyce, C. Kaler, J. Klein, D. Langworthy, M. Little, A. Nadalin, E. Newcomer, D. Orchard, I. Robinson, J. Shewchuka, and T. Storey (2005a). *Web Services Coordination (WS-Coordination), Version 1.0*, last visited December '06.  
<ftp://www6.software.ibm.com/software/developer/library/WS-Coordination.pdf>
- Cabrera, L. F., G. Copeland, M. Feingold, R. W. Freund, T. Freund, J. Johnson, S. Joyce, C. Kaler, J. Klein, D. Langworthy, M. Little, A. Nadalin, E. Newcomer, D. Orchard, I. Robinson, T. Storey, and S. Thatte (2005b). *Web Services Atomic Transaction (WS-AtomicTransaction), Version 1.0*, last visited December '06.  
<ftp://www6.software.ibm.com/software/developer/library/WS-AtomicTransaction.pdf>
- Cabrera, L. F., G. Copeland, M. Feingold, R. W. Freund, T. Freund, S. Joyce, J. Klein, D. Langworthy, M. Little, F. Leymann, E. Newcomer, D. Orchard, I. Robinson, T. Storey, and

- S. Thatte (2005c). *Web Services Business Activity Framework (WS-BusinessActivity), Version 1.0*, last visited December '06.  
<ftp://www6.software.ibm.com/software/developer/library/WS-BusinessActivity.pdf>
- Cannan, S. and G. Otten (1993). *SQL – the Standard Handbook*. McGraw-Hill, Inc., New York, San Francisco, Washington, DC, 1st edition.
- Cantor, S., J. Kemp, R. Philpott, and E. Maler (editors) (2005). *Assertions and Protocols for the OASIS Security Assertion Markup Language (SAML), Version 2.0*, last visited December '06.  
<http://docs.oasis-open.org/security/saml/v2.0/saml-core-2.0-os.pdf>
- Cao, L. Y. and M. T. Özsu (2002). *Evaluation of Strong Consistency Web Caching Techniques*. In *World Wide Web*, volume 5, no. 2 pages 95–123.
- Cao, P. and C. Liu (1998). *Maintaining Strong Cache Consistency in the World Wide Web*. In *IEEE Transactions on Computers*, volume 47, no. 4 pages 445–457.
- Castano, S., M. G. Fugini, G. Martella, and P. Samarati (1994). *Database Security*. Addison-Wesley, Reading, MA, USA, 1st edition.
- Caumanns, J. (2006). *Der Patient bleibt Herr seiner Daten*. In *Informatik-Spektrum* 29 (5) pages 323–331.
- Caumanns, J., H. Weber, A. Fellien, H. Kurrek, O. Boehm, J. Neuhaus, J. Kunsmann, and B. Struif (2006). *Die eGK-Lösungsarchitektur Architektur zur Unterstützung der Anwendungen der elektronischen Gesundheitskarte*. In *Informatik-Spektrum* 29 (5) pages 341–348.
- Chu, Y.-H., J. Feigenbaum, B. LaMacchia, P. Resnick, and M. Strauss (1997). *REFEREE: Trust Management for Web Applications*. In *Computer Networks*, volume 29, no. 8–13 pages 953–964.
- Clarke, D., J.-E. Elien, C. Ellison, M. Fredette, A. Morcos, and R. L. Rivest (2001). *Certificate Chain Discovery in SPKI/SDSI*. In *Journal of Computer Security*, volume 9, no. 4 pages 285–322.
- Cohen, E., R. K. Thomas, W. Winsborough, and D. Shands (2002). *Models for Coalition-based Access Control (CBAC)*. In *Proceedings of the 7th ACM Symposium on Access Control Models and Technologies (SACMAT)*. Monterey, CA, USA, pages 97–106.
- Cranor, L. F. and S. Garfinkel (editors) (2005). *Security and Usability*. O'Reilly & Associates, Sebastopol, CA, USA, 1st edition.
- Curtin, M. (2001). *Developing Trust: Online Privacy and Security*. Apress, Berkeley, CA, USA, 1st edition.
- Date, C. J. and H. Darwen (1997). *A Guide to the SQL Standard*. Addison-Wesley, Reading, MA, USA, 4th edition.



- DeBetta, P. (2004). *Introducing Microsoft SQL Server 2005 for Developers*. Microsoft Press, Buffalo, NY, USA, 1st edition.
- Demurjian, S., T. C. Ting, J. Balthazar, C. E. Phillips, and P. Barr (2001). *A User Role-based Security Model for a Distributed Environment*. In *Research Advances in Database and Information Systems Security*.
- Dröge, R. and M. Raatz (2005). *Microsoft SQL Server 2005 – Konfigurierung, Administration, Programmierung*. Microsoft Press Deutschland, Unterschleißheim, Germany, 1st edition.
- Ellison, C., B. Frantz, B. Lampson, R. Rivest, B. Thomas, and T. Ylonen (1999). *Simple Public Key Certificate*. Internet Draft, last visited December '06.  
<http://theory.lcs.mit.edu/~rivest/simple-public-key-certificate.txt>
- Evered, M. and S. Bögeholz (2004). *A Case Study in Access Control Requirements for a Health Information System*. In *Proceedings of the 2nd Workshop on Australasian Information Security, Data Mining and Web Intelligence, and Software Internationalisation (ACSW Frontiers)*. Dunedin, New Zealand, pages 53–61.
- Fernandez, E. B., E. Gudes, and H. Song (1994). *A Model for Evaluation and Administration of Security in Object-oriented Databases*. In *IEEE Transactions on Knowledge and Data Engineering*, volume 6, no. 2 pages 275–292.
- Ferraiolo, D. F., R. Sandhu, S. Gavrila, D. R. Kuhn, and R. Chandramouli (2001). *Proposed NIST Standard for Role-based Access Control*. In *ACM Transactions on Information and System Security (TISSEC)*, volume 4, no. 3 pages 224–274.
- Freudenthal, E., T. Pesin, L. Port, E. Keenan, and V. Karamcheti (2002). *dRBAC: Distributed Role-based Access Control for Dynamic Coalition Environments*. In *Proceedings of the 22nd IEEE International Conference on Distributed Computing Systems (ICDCS)*. Vienna, Austria, pages 411–420.
- Fröhlich, L., C. Czarski, and K. Maier (2005). *Oracle 10g – Kompendium*. Markt+Technik Verlag, Munich, Germany, 1st edition.
- Gaedke, M., J. Meinecke, and M. Nussbaumer (2005). *A Modeling Approach to Federated Identity and Access Management*. In *Special Interest Tracks and Posters of the 14th International Conference on World Wide Web (WWW)*. Chiba, Japan, pages 1156–1157.
- Galbraith, B., W. Hankison, A. Hiotis, M. Janakiraman, D. V. Prasad, R. Trivedi, and D. Whitney (2002). *Professional Web Services Security*. Wrox Press Ltd, Birmingham, UK, 1st edition.
- Graham, C. (2006). *Market Share: Relational Database Management Systems by Operating System, Worldwide, 2005*. Survey, Gartner, Stamford, USA.

- Gray, C. and D. Cheriton (1989). *Leases: An Efficient Fault-tolerant Mechanism for Distributed File Cache Consistency*. In *Proceedings of the 12th ACM Symposium on Operating Systems Principles (SOSP)*. Litchfield Park, AZ, USA, pages 202–210.
- Gudes, E., M. S. Olivier, and R. P. van de Riet (1999). *Modelling, Specifying and Implementing Workflow Security in Cyberspace*. In *Journal of Computer Security*, volume 7, no. 4 pages 287–315.
- Gudgin, M. and A. Nadalin (editors) (2005). *Web Services Secure Conversation Language (WS-SecureConversation)*, last visited December '06.  
<ftp://www6.software.ibm.com/software/developer/library/ws-secureconversation.pdf>
- Guo, S., W. Sun, and M. A. Weiss (1996). *Solving Satisfiability and Implication Problems in Database Systems*. In *ACM Transactions on Database Systems (TODS)*, volume 21, no. 2 pages 270–293.
- Gutiérrez, C., E. Fernández-Medina, and M. Piattini (2005). *Web Services Enterprise Security Architecture: A Case Study*. In *Proceedings of the 2005 Workshop on Secure Web Services (SWS)*. Fairfax, VA, USA, pages 10–19.
- Haas, F. (2006). *Oracle Security in der Praxis*. Carl Hanser Verlag, Munich, Germany, 1st edition.
- Hagström, A., S. Jajodia, F. Parisi-Presicce, and D. Wijesekera (2001). *Revocations – A Classification*. In *Proceedings of the 14th IEEE Computer Security Foundations Workshop*. Nova Scotia, Canada, pages 44–58.
- Harrison, M., W. Ruzzo, and J. Ullman (1976). *Protection in Operating Systems*. In *Communications of the ACM*, volume 19, no. 8 pages 461–471.
- Hommel, W. and H. Reiser (2005). *Federated Identity Management: Die Notwendigkeit zentraler Koordinationsdienste*. In *Kommunikation in Verteilten Systemen KiVS Kurzbeiträge und Workshop*, volume 61 of *Lecture Notes in Informatics (LNI)*. GI, Kaiserslautern, Germany, pages 65–72.
- Huang, W.-K. and V. Atluri (1999). *SecureFlow: a Secure Web-enabled Workflow Management System*. In *Proceedings of the 4th ACM Workshop on Role-based Access Control (RBAC)*. Fairfax, VA, USA, pages 83–94.
- IBM (1993 – 2004). *IBM DB2 Universal Database – SQL Reference 2 Version 8.2*. IBM Corporation.
- IBM and Microsoft (2002). *Security in a Web Services World: A Proposed Architecture and Roadmap*. A joint security whitepaper from IBM Corporation and Microsoft Corporation, last visited December '06.  
<http://www-106.ibm.com/developerworks/webservices/library/ws-secmap/>

- Informatik-Spektrum 29(5) (2006). *Informatik-Spektrum*, volume 29. Springer Berlin / Heidelberg.
- ISO/IEC (1997). *10181-3: Information technology – Open Systems Interconnection – Security frameworks for open systems: Access control framework*.
- Jaeger, T. (1999). *Access Control in Configurable Systems*. In *Secure Internet Programming, Security Issues for Mobile and Distributed Objects*, volume 1603 of *Lecture Notes in Computer Science (LNCS)*. pages 289–316.
- Jajodia, S., P. Samarati, M. L. Sapino, and V. S. Subrahmanian (2001). *Flexible Support for Multiple Access Control Policies*. In *ACM Transactions on Information and System Security (TISSEC)*, volume 26, no. 2 pages 214–260.
- Jajodia, S., P. Samarati, and V. S. Subrahmanian (1997a). *A Logical Language for Expressing Authorizations*. In *Proceedings of the 1997 IEEE Symposium on Security and Privacy (SP)*. Oakland, CA, USA, pages 31–42.
- Jajodia, S., P. Samarati, V. S. Subrahmanian, and E. Bertino (1997b). *A Unified Framework for Enforcing Multiple Access Control Policies*. In *Proceedings of the 1997 ACM SIGMOD International Conference on Management of Data*. Tucson, Arizona, United States, pages 474–485.
- Jajodia, S. and R. Sandhu (1990). *Polyinstantiation Integrity in Multilevel Relations*. In *Proceedings of the 1990 IEEE Symposium on Security and Privacy (SP)*. Oakland, CA, USA, pages 104–115.
- Jajodia, S. and R. Sandhu (1991). *Toward a Multilevel Secure Relational Data Model*. In *Proceedings of the 1991 ACM SIGMOD International Conference on Management of Data*. pages 50–59.
- Kang, M. H., J. S. Park, and J. N. Froscher (2001). *Access Control Mechanisms for Inter-organizational Workflow*. In *Proceedings of the 6th ACM Symposium on Access Control Models and Technologies (SACMAT)*. Chantilly, VA, United States, pages 66–74.
- Karjoth, G. and J. Posegga (2000). *Mobile Agents and Telcos' Nightmares*. In *Annales des télécommunications*, volume 55, no. 7-8 pages 388–400.
- Keahey, K. and V. Welch (2002). *Fine-Grain Authorization for Resource Management in the Grid Environment*. In *Proceedings of the 3rd International Workshop on Grid Computing*, volume 2536 of *Lecture Notes in Computer Science (LNCS)*. Baltimore, MD, USA, pages 199–206.
- Keahey, K., V. Welch, S. Lang, B. Liu, and S. Meder (2003). *Fine-Grain Authorization Policies in the GRID: Design and Implementation*. In *Workshop Proceedings of the International Middleware Conference*. Rio de Janeiro, Brazil, pages 170–177.

- Kehr, R., J. Posegga, R. Schmitz, and P. Windirsch (2001). *Mobile Security for Internet Applications*. In *Arbeitskonferenz Kommunikationssicherheit 2001*, Lecture Notes in Computer Science (LNCS).
- Keidl, M. (2004). *Metadata Management and Context-based Personalization in Distributed Information Systems*. Ph.D. thesis, Technische Universität München.
- Keidl, M., S. Seltzsam, K. Stocker, and A. Kemper (2002). *ServiceGlobe: Distributing E-Services across the Internet (Demonstration)*. In *Proceedings of the International Conference on Very Large Data Bases (VLDB)*, pages 1047–1050.
- Kemper, A. and A. Eickler (2006). *Datenbanksysteme – Eine Einführung*. R. Oldenbourg Verlag, Munich, Germany, 6th edition.
- Khalaf, R. and F. Leymann (2003). *On Web Services Aggregation*. In *Proceedings of the 4th International Workshop on Technologies for E-Services (TES)*, volume 2819 of *Lecture Notes in Computer Science (LNCS)*. Berlin, Germany, pages 1–13.
- Kühnemann, A. and H. Vogler (1997). *Attributgrammatiken – Eine grundlegende Einführung*. Vieweg, 1st edition.
- Lampson, B. (1974). *Protection*. In *ACM SIGOPS Operating Systems Review*, volume 8, no. 1 pages 18–24.
- Li, N., B. Grosz, and J. Feigenbaum (2003a). *Delegation Logic: A Logic-based Approach to Distributed Authorization*. In *TISSEC 2003 v.6* pages 128–171.
- Li, N., B. N. Grosz, and J. Feigenbaum (2000). *A Practically Implementable and Tractable Delegation Logic*. In *Proceedings of the 2000 IEEE Symposium on Security and Privacy (SP)*. Berkeley, CA, USA, pages 27–42.
- Li, N., J. C. Mitchell, and W. H. Winsborough (2002). *Design of a Role-based Trust Management Framework*. In *Proceedings of the 2002 IEEE Symposium on Security and Privacy (SP)*. Oakland, CA, USA, pages 114–130.
- Li, N., W. H. Winsborough, and J. C. Mitchell (2003b). *Distributed Credential Chain Discovery in Trust Management*. In *Journal of Computer Security*, volume 11, no. 1 pages 35–86.
- Linthicum, D. S. (2001). *B2B Application Integration: E-Business – Enable Your Enterprise*. Addison-Wesley, Reading, MA, USA, 1st edition.
- Linthicum, D. S. (2003). *Next Generation Application Integration: From Simple Information to Web Services*. Addison-Wesley, Reading, MA, USA, 1st edition.
- Lippe, S. (2004). *How to Create Cool Models with Maestro... and how to Execute them in Nehemiah*. Technical report, SAP Research, Brisbane, Australia.

- Loney, K. (2005). *Oracle Database 10g – Die umfassende Referenz*. Carl Hanser Verlag, Munich, Germany, 1st edition.
- Lopez, J., R. Oppliger, and G. Pernul (2004). *Authentication and Authorization Infrastructures (AAIs): a Comparative Survey*. In *Computers & Security*, volume 23, no. 7 pages 578–590.
- Lord, P. (2002). *Managing E-Business Security Challenges*. Technical report, Oracle Corporation, Redwood Shores, CA, USA.
- Moses, T. (editor) (2003). *XACML Profile for Web-services*, last visited December '06.  
<http://www.oasis-open.org/committees/download.php/3661/draft-xacml-wspl-04.pdf>
- Moses, T. (editor) (2005). *eXtensible Access Control Markup Language (XACML), Version 2.0*, last visited December '06.  
[http://docs.oasis-open.org/xacml/2.0/access\\_control-xacml-2.0-core-spec-os.pdf](http://docs.oasis-open.org/xacml/2.0/access_control-xacml-2.0-core-spec-os.pdf)
- Mukkamala, R., V. Atluri, J. Warner, and R. Abbasasari (2006). *A Distributed Coalition Service Registry for Ad-Hoc Dynamic Coalitions: A Service-oriented Approach*. In *Proceedings of the 20th Working Conference on Data and Applications Security (DBSec)*, volume 4127 of *Lecture Notes in Computer Science (LNCS)*. Sophia Antipolis, France, pages 209–223.
- Nadalin, A., C. Kaler, R. Monzillo, and P. Hallam-Baker (editors) (2006). *Web Services Security: SOAP Message Security 1.1 (WS-Security 2004)*, last visited December '06.  
[http://www.oasis-open.org/committees/tc\\_home.php?wg\\_abbrev=wss](http://www.oasis-open.org/committees/tc_home.php?wg_abbrev=wss)
- Neuhaus, J., W. Deiters, and M. Wiedeler (2006). *Mehrwertdienste im Umfeld der elektronischen Gesundheitskarte*. In *Informatik-Spektrum* 29 (5) pages 332–340.
- O'Neill, M. (2003). *Web Services Security*. McGraw-Hill, Inc., New York, San Francisco, Washington, DC, 1st edition.
- Orhanovic, J., I. Grodtke, and M. Tiefenbacher (2004). *DB2 Administration – Einführung, Handbuch und Referenz*. Addison-Wesley, Munich, Germany, 1st edition.
- Osborn, S., R. Sandhu, and Q. Munawer (2000). *Configuring Role-based Access Control to Enforce Mandatory and Discretionary Access Control Policies*. In *ACM Transactions on Information and System Security (TISSEC)*, volume 3, no. 2 pages 85–106.
- Paulus, S. (2006). *Collaborative Workflow Security*. Key Note at the International Conference on Emerging Trends in Information and Communication Security (ETRICS 2006), Freiburg, Germany.
- Pearlman, L., I. F. V. Welch, C. Kesselman, and S. Tuecke (2002). *A Community Authorization Service for Group Collaboration*. In *Proceedings of the 3rd International Workshop on Policies for Distributed Systems and Networks (POLICY)*. Monterey, CA, USA, pages 50–59.

- Pettey, C. (2006). *Gartner Says Worldwide Relational Database Market Increased 8 Percent in 2005*, last visited December '06.  
[http://www.gartner.com/press\\_releases/asset\\_152619\\_11.html](http://www.gartner.com/press_releases/asset_152619_11.html)
- Phillips, C. E., S. Demurjian, and T. C. Ting (2002a). *Security Engineering for Roles and Resources in a Distributed Environment*. In *Proceedings of the 3rd Annual Information Systems Security Engineering Conference*. page 12.
- Phillips, C. E., T. C. Ting, and S. Demurjian (2002b). *Information Sharing and Security in Dynamic Coalitions*. In *Proceedings of the 7th ACM Symposium on Access Control Models and Technologies (SACMAT)*. Monterey, CA, USA, pages 87–96.
- Posegga, J. (1998). *Die Sicherheitsaspekte von Java*. In *Informatik Spektrum*, volume 21, no. 1 pages 16–22.
- Posegga, J. (1999). *Jini: Infrastruktur für dynamische Dienste in verteilten Systemen – Aktuelles Schlagwort*. In *Informatik-Spektrum*, volume 22, no. 1 pages 43–44.
- Priebe, T., W. Dobmeier, B. Muschall, and G. Pernul (2005). *ABAC – Ein Referenzmodell für attributbasierte Zugriffskontrolle*. In *Proceedings of Sicherheit 2005: Sicherheit - Schutz und Zuverlässigkeit*, volume 62 of *Lecture Notes in Informatics (LNI)*. GI, Regensburg, Germany, pages 285–296.
- Rabitti, F., E. Bertino, W. Kim, and D. Woelk (1991). *A Model of Authorization for Next-Generation Database Systems*. In *ACM Transactions on Database Systems (TODS)*, volume 16, no. 1 pages 88–131.
- Rabitti, F., D. Woelk, and W. Kim (1988). *A Model of Authorization for Object-oriented and Semantic Databases*. In *Proceedings of the 1st International Conference on Extending Database Technology (EDBT)*, volume 303 of *Lecture Notes in Computer Science (LNCS)*. Venice, Italy, pages 231–250.
- Richardson, J., P. Schwarz, and L.-F. Cabrera (1992). *CACL: Efficient Fine-grained Protection for Objects*. In *Proceedings of the Conference on Object-Oriented Programming Systems, Languages, and Applications (OOPSLA)*. Vancouver, Canada, pages 263–275.
- Rits, M., B. De Boe, and A. Schaad (2005). *XacT: a Bridge between Resource Management and Access Control in Multi-layered Applications*. In *Proceedings of the 2005 Workshop on Software Engineering for Secure Systems (SESS)*. St. Louis, MO, USA, pages 1–7.
- Rivest, R. L. and B. Lampson (1996). *SDSI – A Simple Distributed Security Infrastructure*, last visited December '06.  
<http://research.microsoft.com/Lampson/59-SDSI/WebPage.html>

- Robinson, P., F. Kerschbaum, and A. Schaad (2006). *From Business Process Choreography to Authorization Policies*. In *Proceedings of the 20th Working Conference on Data and Applications Security (DBSec)*, volume 4127 of *Lecture Notes in Computer Science (LNCS)*. Sophia Antipolis, France, pages 297–309.
- Rosenberg, J. and D. Remy (2004). *Securing Web Services with WS-Security: Demystifying WS-Security, WS-Policy, SAML, XML Signature, and XML Encryption*. Pearson Education, 1st edition.
- Rosenkrantz, D. J. and H. B. Hunt (1980). *Processing Conjunctive Predicates and Queries*. In *Proceedings of the International Conference on Very Large Data Bases (VLDB)*. Montreal, Canada, pages 64–72.
- Ruh, W. A., F. X. Maginnis, and W. J. Brown (2000). *Enterprise Application Integration*. John Wiley & Sons, New York, NY, USA, 1st edition.
- Samarati, P. and S. de Capitani di Vimercati (2001). *Access Control: Policies, Models, and Mechanisms*. In *Foundations of Security Analysis and Design*, volume 2171 of *Lecture Notes in Computer Science (LNCS)*. Italy.
- Sandhu, R. (1996). *Roles Versus Groups*. In *Proceedings of the 1st ACM Workshop on Role-based Access Control (RBAC)*. page 7.
- Sandhu, R. S., E. J. Coyne, H. L. Feinstein, and C. E. Youman (1996). *Role-based Access Control Models*. In *IEEE Computer*, volume 29, no. 2 pages 38–47.
- Sandhu, R. S. and P. Samarati (1994). *Access Control: Principles and Practice*. In *IEEE Communications Magazine*, volume 32, no. 9 pages 40–48.
- SAP Info (2006). *Protecting Against Negligence, Fending off Intrusions: IT SECURITY*, volume 136 of *SAP INFO*. SAP AG, Walldorf, Germany.
- Schaad, A. and P. Spadone (2005). *Gabriel Conceptual Model*. Technical report, SAP Research, Sophia Antipolis, France.
- Schaad, A., P. Spadone, and H. Weichsel (2005). *A Case Study of Separation of Duty Properties in the Context of the Austrian "eLaw" Process*. In *Proceedings of the 2005 ACM Symposium on Applied Computing (SAC)*. Santa Fe, New Mexico, pages 1328–1332.
- Seltzsam, S. (2005). *Security, Caching, and Self-Management in Distributed Information Systems*. Ph.D. thesis, Technische Universität München.
- Seltzsam, S., S. Börzsönyi, and A. Kemper (2001). *Security for Distributed E-Service Composition*. In *Proceedings of the 2nd International Workshop on Technologies for E-Services (TES)*, volume 2193 of *Lecture Notes in Computer Science (LNCS)*. Rome, Italy, pages 147–162.

- Shen, H. and P. Dewan (1992). *Access Control for Collaborative Environments*. In *Proceedings of the 1992 ACM Conference on Computer-supported Cooperative Work (CSCW)*. ACM Press, Toronto, Ontario, Canada, pages 51–58.
- Stiegler, M., A. Karp, K.-P. Yee, T. Close, and M. Miller (2006). *Polaris: Virus-safe Computing for Windows XP*. In *Communications of the ACM*, volume 49, no. 9 pages 83–88.
- Sun, X.-H., N. Kamel, and L. M. Ni (1989). *Solving Implication Problems in Database Applications*. In *Proceedings of the 1989 ACM SIGMOD International Conference on Management of Data*. Portland, OR, USA, pages 185–192.
- Thompson, M., A. Essiari, and S. Mudumbai (2003). *Certificate-based Authorization Policy in a PKI Environment*. In *TISSEC 2003 v.6* pages 566–588.
- Thompson, M., W. Johnston, S. Mudumbai, G. Hoo, K. Jackson, and A. Essiari (1999). *Certificate-based Access Control for Widely Distributed Resources*. In *Proceedings of the 8th Usenix Security Symposium*. Washington, DC, USA, pages 215–228.
- TISSEC 2003 v.6 (2003). *ACM Transactions on Information and System Security (TISSEC)*, volume 6. ACM Press, New York, NY, USA.
- Tolone, W., G.-J. Ahn, T. Pai, and S.-P. Hong (2005). *Access Control in Collaborative Systems*. In *ACM Computing Surveys*, volume 37, no. 1 pages 29–41.
- Walter, T., L. Bussard, J. Haller, R. Kilian-Kehr, J. Posegga, and P. Robinson (2004). *Secure Mobile Business Applications: Framework, Architecture and Implementation*. In *Information Security Technical Report Journal*, volume 9, no. 4 pages 6–22.
- Warner, J., V. Atluri, and R. Mukkamala (2005). *A Credential-based Approach for Facilitating Automatic Resource Sharing Among Ad-Hoc Dynamic Coalitions*. In *Proceedings of the 19th Working Conference on Data and Applications Security (DBSec)*, volume 3654 of *Lecture Notes in Computer Science (LNCS)*. Storrs, CT, USA, pages 252–266.
- Weerawarana, S., F. Curbera, F. Leymann, T. Storey, and D. F. Ferguson (2005). *Web Services Platform Architecture: SOAP, WSDL, WS-Policy, WS-Addressing, WS-BPEL, WS-Reliable Messaging, and More*. Prentice Hall, Upper Saddle River, NJ, USA, 1st edition.
- Weilbach, J. and M. Herger (2005). *SAP xApps and the Composite Application Framework*. Galileo Press, Bonn, Germany, 1st edition.
- Welch, V., F. Siebenlist, I. Foster, J. Bresnahan, K. Czajkowski, J. Gawor, C. Kesselman, S. Meder, L. Pearlman, and S. Tuecke (2003). *Security for Grid Services*. In *Proceedings of the 12th IEEE International Symposium on High Performance Distributed Computing (HPDC)*. Seattle, WA, USA, pages 48–61.



- Wiehler, G. (2004). *Mobility, Security und Web Services – Neue Technologien und Serviceorientierte Architekturen für zukunftsweisende IT-Lösungen*. Siemens AG, Publicis Corporate Publishing, Erlangen, 1st edition.
- Wijesekera, D. and S. Jajodia (2001). *Policy Algebras for Access Control: the Propositional Case*. In *Proceedings of the 8th ACM Conference on Computer and Communications Security (CCS)*. Philadelphia, PA, USA, pages 38–47.
- Wijesekera, D. and S. Jajodia (2002). *Policy Algebras for Access Control: the Predicate Case*. In *Proceedings of the 9th ACM Conference on Computer and Communications Security (CCS)*. Washington, DC, USA, pages 171–180.
- Wijesekera, D. and S. Jajodia (2003). *A Propositional Policy Algebra for Access Control*. In *TISSEC 2003 v.6* pages 286–325.
- Wimmer, M., M.-C. Albutiu, and A. Kemper (2006a). *Optimized Workflow Authorization in Service Oriented Architectures*. In *Proceedings of the International Conference on Emerging Trends in Information and Communication Security (ETRICS)*, volume 3995 of *Lecture Notes in Computer Science (LNCS)*. Freiburg, Germany, pages 30–44.
- Wimmer, M., D. Eberhardt, P. Ehrnlechner, and A. Kemper (2004). *Reliable and Adaptable Security Engineering for Database-Web Services*. In *Proceedings of the 4th International Conference on Web Engineering (ICWE)*, volume 3140 of *Lecture Notes in Computer Science (LNCS)*. Munich, Germany, pages 502–515.
- Wimmer, M., P. Ehrnlechner, A. Fischer, and A. Kemper (2005a). *Flexible Autorisierung in Datenbank-basierten Web Service-Föderationen*. In *Informatik – Forschung und Entwicklung*, volume 20, no. 3 pages 167–181.
- Wimmer, M., P. Ehrnlechner, and A. Kemper (2005b). *Flexible Autorisierung in Web Service-Föderationen*. In *Proceedings of the GI Conference on Database Systems for Business, Technology, and Web (BTW)*. Karlsruhe, Germany, pages 185–204.
- Wimmer, M. and A. Kemper (2005). *An Authorization Framework for Sharing Data in Web Service Federations*. In *Proceedings of the 2nd VLDB Workshop on Secure Data Management (SDM)*, volume 3674 of *Lecture Notes in Computer Science (LNCS)*. Trondheim, Norway, pages 47–62.
- Wimmer, M., A. Kemper, M. Rits, and V. Lotz (2006b). *Consolidating the Access Control of Composite Applications and Workflows*. In *Proceedings of the 20th Working Conference on Data and Applications Security (DBSec)*, volume 4127 of *Lecture Notes in Computer Science (LNCS)*. Sophia Antipolis, France, pages 44–59.
- Wimmer, M., A. Kemper, and S. Seltzsam (2006c). *Security for Web Applications*. In *Web Engineering: The Discipline of Systematic Development*, John Wiley & Sons, New York, NY, USA pages 265–292.

- Woods, D. (2004). *Enterprise Services Architecture*. Galileo Press, Bonn, Germany, 1st edition.
- WSBPEL (2006). *OASIS Web Services Business Process Execution Language (WSBPEL)*, last visited December '06.  
[http://www.oasis-open.org/committees/tc\\_home.php?wg\\_abbrev=wsbpel](http://www.oasis-open.org/committees/tc_home.php?wg_abbrev=wsbpel)
- Yu, T., M. Winslett, and K. E. Seamons (2003). *Supporting Structured Credentials and Sensitive Policies through Interoperable Strategies for Automated Trust Negotiation*. In TISSEC 2003 v.6 pages 1–42.
- Zhang, L., G.-J. Ahn, and B.-T. Chu (2003). *A Rule-based Framework for Role-based Delegation and Revocation*. In TISSEC 2003 v.6 pages 404–441.