Tilburg University

Access control and service-oriented architectures

Leune, C.J.

Publication date:
2007

Link to publication in Tilburg University Research Portal

Citation for published version (APA):
Leune, C. J. (2007). *Access control and service-oriented architectures*. CentER, Center for Economic Research.

General rights
Copyright and moral rights for the publications made accessible in the public portal are retained by the authors and/or other copyright owners and it is a condition of accessing publications that users recognise and abide by the legal requirements associated with these rights.

- Users may download and print one copy of any publication from the public portal for the purpose of private study or research.
- You may not further distribute the material or use it for any profit-making activity or commercial gain
- You may freely distribute the URL identifying the publication in the public portal

Take down policy
If you believe that this document breaches copyright please contact us providing details, and we will remove access to the work immediately and investigate your claim.

Download date: 12. May. 2021

ACCESS CONTROL
AND
SERVICE-ORIENTED ARCHITECTURES

Kees Leune

ACCESS CONTROL
AND
SERVICE-ORIENTED ARCHITECTURES

PROEFSCHRIFT

ter verkrijging van de graad van doctor aan de Universiteit van Tilburg,
op gezag van de rector magnificus, prof.dr. F.A. van der Duyn Schouten,
in het openbaar te verdedigen
ten overstaan van een door het college voor promoties aangewezen commissie
in de aula van de Universiteit op woensdag 28 februari 2007 om 14:15 uur

door

Cornelis Jan Leune

geboren op 29 augustus 1973 te Breda

Promotores: prof.dr.ir. M.P. Papazoglou
prof.dr. H.A. Proper
Copromotor: dr. W.J.A.M. van den Heuvel

SIKS Dissertation Series No. 2007-01 CentER Dissertation Series No. 188

The research reported in this thesis has been carried out under the auspices of SIKS, the Dutch Research School for Information and Knowledge Systems, and CentER, the Graduate School of the Faculty of Economics and Business Administration of Tilburg University.

Netherlands Organisation for Scientific Research

This research has been partially funded by the Netherlands Organisation for Scientific Research (NWO) via project PRONIR.

Copyright © Kees Leune, 2004–2007

All rights reserved. No part of this publication may be reproduced, stored in a retrieval system, or transmitted, in any form or by any means, electronic, mechanical, photocopying, recording, or otherwise, without prior written permission from the author.

ISBN 978-90-5668-188-5

For my wife, and for my parents.

Contents

Contents **vii**

1 Introduction **1**
1.1 EFSOC 2
1.2 Research Motivation 3
1.3 Research goal and scope of the research 4
1.4 Research questions 6
1.5 Research methodology 6
1.6 Contributions 9
1.7 Structure of this thesis 10

I Background and Theory 13

2 Background in SOA, Security and Event-Driven Processing **15**
2.1 Introduction 15
2.2 Service Security 15
  2.2.1 Authentication 16
  2.2.2 Integrity and Confidentiality 17
  2.2.3 Public Key Infrastructures 18
  2.2.4 Web of Trust 19
  2.2.5 Access Control 21
  2.2.6 Discretionary Access Control 22
  2.2.7 Mandatory Access Control 23
  2.2.8 Role-Based Access Control 24
  2.2.9 Auditing 26
  2.2.10 Security and Grid Services 27
  2.2.11 Discussion 27
2.3 Service-Oriented Computing 28
  2.3.1 Service-Oriented Architecture 28
  2.3.2 Loosely Coupled Message-Oriented Systems 30
  2.3.3 Service Composition 30
  2.3.4 Web Services 31
  2.3.5 The IBM and Microsoft Road-map for Web Services Security 32
  2.3.6 SAML: Security Assertion Markup Language 35

2.3.7 XACML 36
2.3.8 Discussion 37
2.4 Event-driven processing 38
2.4.1 Properties of event-driven processing 39
2.4.2 Enterprise Service Bus 41
2.4.3 Event-driven interaction patterns 42
2.4.4 Event message filtering 43
2.5 Discussion 45
2.5.1 Trust 45
2.5.2 Delegation 46
2.6 Conclusions 48
2.6.1 Objectives for a secure SOA 48
2.6.2 Requirements for Access Control and Service-Oriented Architectures 50
2.6.3 State of the Art in Research 51
2.6.4 Summary 52

3 The EFSOC Service-Oriented Architecture 55
3.1 Introduction 55
3.2 Case study 56
3.2.1 Elicitation Process 56
3.2.2 The HIPAA Privacy Rule 57
3.2.3 Northside Hospital 57
3.2.4 Hospital Policies 58
3.2.5 Running example 62
3.3 Concepts 64
3.4 Event operations 65
3.4.1 Publishing events and subscribing to events 66
3.4.2 Sending and receiving events 67
3.4.3 Events in context 67
3.5 EFSOC and the Enterprise Service Bus 69
3.6 Role operations 69
3.6.1 Subjects and roles 69
3.6.2 Role assignments and role sessions 71
3.7 Access control 72
3.7.1 Taking access control decisions 73
3.7.2 Evaluating access control policies 77
3.7.3 Evaluating access control rules 78
3.8 Architecture 78
3.9 Example: Applying Access Control Policies 82
3.10 Discussion 85
3.10.1 EFSOC and Web Services 85
3.10.2 EFSOC and its design objectives 86
3.10.3 Delegation and role hierarchies 88
3.11 Summary 89

4 EFSOC Definition and Execution Language **91**
4.1 Extensible Markup Language 91
4.2 The EDL language 91
4.3 EDL in relation to WSDL 92
4.4 Notational conventions 93
4.5 Definition Language 93
  4.5.1 Representing events 93
4.6 Vocabulary Definitions 93
  4.6.1 Subject 93
  4.6.2 Role 95
  4.6.3 Role Attribute Type 95
  4.6.4 Role Attribute Value 95
  4.6.5 Event 95
  4.6.6 Event Header 96
  4.6.7 Event Body 96
  4.6.8 Event Body Type 97
  4.6.9 Access Control Policies 97
4.7 Execution language definitions 97
  4.7.1 Publish 97
  4.7.2 Unpublish 98
  4.7.3 Send 98
  4.7.4 Subscribe 99
  4.7.5 Unsubscribe 99
  4.7.6 Assign 100
  4.7.7 Unassign 100
  4.7.8 Activate 101
  4.7.9 Deactivate 101
  4.7.10 Set 101
4.8 Access Control Rules 101
  4.8.1 Principal 102
  4.8.2 Permission 102
  4.8.3 Operation 102
  4.8.4 Condition 102
  4.8.5 Transport level expressions 103
  4.8.6 Message level expressions 103
  4.8.7 Message-context level expressions 104
  4.8.8 Misc expressions 105
  4.8.9 Combining conditions 105
4.9 Summary 105

5 EFSOC Query Language **107**
5.1 Querying XML 107
  5.1.1 XPath 107
  5.1.2 XQuery 109
  5.1.3 XSL 111

5.2 EFSOC Query Language Overview 112
    5.2.1 Basic queries 113
    5.2.2 Second-order queries 115
5.3 Summary 115

II Validation 117

6 Formal Foundations 119
6.1 Reasons for formalizing 119
6.2 Approach 120
    6.2.1 Predicate Logic 121
    6.2.2 Datalog 122
    6.2.3 Telos 122
6.3 Definition and Constraint Language 124
6.4 Query Language 126
6.5 Expressing Security Policies and Security Rules 128
    6.5.1 Security Rules: Queries or Constraints 129
6.6 Implementing Separation of Duty 129
6.7 Relationship with EDL 130
6.8 Discussion 131

7 Prototype Implementation 133
7.1 Introduction 133
7.2 Laboratory Experiment 133
7.3 Architecture and Technology 134
    7.3.1 Main prototype 134
    7.3.2 Proofs-of-concept 135
7.4 Element definitions 140
7.5 Defining access control rules 142
7.6 Observations and Conclusions 144

8 Conclusions, Discussion and Future Research 145
8.1 Summary 145
8.2 Research Results 146
8.3 Case study results 148
8.4 Contributions 148
8.5 Benefits and Limitations 149
8.6 Future Research 150

III Appendices 155

EFSOC XML Definitions 157

EFSOC Conceptbase definitions 169

Samenvatting 179

IV Reference 181

SIKS Dissertation Series 183

List of Figures 191

Bibliography 193

Author Index 199

Index 205

Preface

When I was still a student in high school, I read a book titled *The Cuckoo's Egg*, written by Clifford Stoll (1990). In it, Stoll describes the (often extremely low-tech) chase of a computer hacker who had gained access to a plethora of computers, all of which were connected to an Internet. In that time, I was happy when I could occasionally use a dial-up modem to connect to bulletin-board systems. The world of computer systems–and the connections that could exist between them–has been on my mind since then.

It was not until 1992 that I enrolled as a full-time student of Information Systems and Technology at Tilburg University. At university, I was lucky enough to be selected to become one of about one hundred students to participate in the pilot *Student Email*. My interest in computer networks, which had been mostly dormant until that point, got a new impulse. At that time, Tilburg University claimed to have the most advanced *electronic library* of Europe, containing several hundred personal computers with Internet access for students.

Unfortunately, while these machines did indeed possess an Internet connection, no client software was made available to us, and we were limited to navigating the Internet through text-based menus which ran predefined telnet-commands to connect to on-line public access catalogs of remote libraries. Most of the public Internet-enabled applications were Gopher, WAIS, Veronica, Archie, etc.

Every now and then—much less frequently nowadays—we would find outdated connections, and we would be given the privilege of falling back to a telnet-prompt. It was in that time that my fellow students and I also discovered Internet games (MUDs, BBSes, MOOs, IRC, etc.), and my real education began.

Over the years, I managed to convince faculty that I really needed to have access to central machines (at first VAX/VMS, then DEC Ultrix, followed by SunOS and Solaris, HP/UX and finally on to just about all different Linuxes). After I joined the board of the study association of the Department of Information Systems, I co-founded a student-run server-infrastructure, which later evolved into a service which (still) provides web space to *all* student associations of Tilburg University.

Over the course of those years, my interest in computer networks, and more specifically, in security of information technology, became more and more pronounced. After my graduation, and after my first job at a research center at University, I eventually switched back to a position in academia, and began the work that is reported in this thesis.

Tilburg University's Infolab provided a open environment that highly encouraged experimentation. While such an environment provides a researcher with all the freedom that he needs to explore his interests, it also requires him to stay focused on his goal. There are simply too many interesting questions to find an answer to, or technologies to learn. However, despite the many distractions that are part of life in a university research lab, my research started showing results from 2003 onwards with academic publications in workshops, conference, and journals.

In addition to the open research environment, the Infolab also prides itself on being a research lab that addresses real-world problems. As a result of this viewpoint, I have been involved in many applied-research projects, which are often contract based. The most important projects that stand out to me are the MeMo project, in which I designed and built the search facilities for an e-commerce broker for vertical markets; the AIRT project, in which I was the lead architect and developer of a support system for computer-security incident-response teams, and the UvT-CERT project, in which I participated in the daily IT security operations of the University.

Several of these projects were instrumental to where I am now in my professional career. In 2004, Tilburg University established its own computer-security incident response team (UvT-CERT) and a task force was created to improve the overall security of the University's computer systems and networks. My involvement with UvT-CERT and with the security task force was a major influence on my level of knowledge and expertise, and provided the opportunity for me to get to know the security world. I was introduced to the global Forum for Incident Response and Security Teams (FIRST), and I started getting invited to speak in forums organized by SURFnet (the Dutch National Research and Education Network), the operational incident response team meeting cycle, organized by GOVCERT, and I spoke at the 2006 FIRST conference. All of these appearances and contacts eventually convinced me that in the short-term, my future was not in academia, but that I wanted to tackle practical problems in real-world environments. For this insight, I would like to thank Teun Nijssen for having enough confidence in my abilities and for giving me these chances.

None of the work that I have done at CentER Applied Research and at Tilburg University would have been of the same level of quality if I did not feel the support of my colleagues. I wish to express gratitude to prof.dr.ir. Mike Papazoglou for strategically keeping my research on focus, for providing me the opportunity to enrich my own knowledge by teaching several classes, and for showing me how to write good scientific articles. I would also like to thank dr. Willem-Jan van den Heuvel for helping me in my day-to-day research and providing a sounding board for several of my ideas.

Many thanks also to the Infolab crew, dr.ir. Jeroen Hoppenbrouwers (hoppie) for his critical voice and technical expertise, ir. Frans Laurijssen for his view of life and for his in-depth programming knowledge, and to drs. Benedikt Kratz and to drs. Bart Orriëns as academic sparring partners. I would also like to mention the many students that graduated at the Infolab. Each of you have has had an impact on my research. Special mention goes to Herman Suijs, Martin Schapendonk, and to Karin van den Berg.

Finally, I would like to express gratitude to my Ph.D. committee: Prof.dr.ir. Mike Papazoglou (promotor), Prof.dr. Erik Proper (promotor), Dr. Willem-Jan van den Heuvel (co-promotor), Dr. Manfred Jeusfeld, Dr. Hans Weigand, Prof.dr. Nikolaou and Dr. Weiss. The members of the committee tested the scientific value of my research and provided valuable feedback to clarify certain ideas that I had.

Writing a Ph.D.-thesis is hard work. However, in addition to work, it also has a profound influence on the personal life of the candidate. Especially during the writing phase, minds are never really off and attention spans can be short, which can be hard on family life. I would like very much to thank my wife, Lou-Anne, for her patience and understanding, and for willingly giving up many things that we would have been able to do together. She is also an excellent mother to our daughter, Paulette, who is probably too young to realize what was going on, but experienced the consequences of having parents with families on different sides of the ocean. Last, but not least, a special thank-you goes to Kristin Sheerin for providing me with many editorial hints, comments and corrections!

Finally, I would like to acknowledge the NWO and thank them for funding the PRONIR project. Without that project, this thesis would not have been written.

A web site to accompany this PhD thesis has been set up, and will be available at `http://www.leune.org/thesis`. The site will contain errata, downloads and an online version of the thesis.

Last, but not least, a note to the reader: while much care has been given to maintaining a high level of consistency between the body of the thesis and its appendices, it is possible that inconsistencies do occur. Mostly, these inconsistencies are the result of repeated revisions of the text. If such inconsistencies are detected, the appendices are always authorative. The models in the appendices have all been implemented in software and as such, should be—at least logically—correct.

Tilburg, The Netherlands
August, 2006.

Chapter 1

Introduction

> The main skill is to keep from getting lost. Since the roads are used only by local people who know them by sight nobody complains if the junctions aren't posted. And often they aren't. When they are it's usually a small sign hiding unobtrusively in the weeds and that's all. County-road-sign makers seldom tell you twice. If you miss that sign in the weeds that's your problem, not theirs. Moreover, you discover that the highway maps are often inaccurate about county roads. And from time to time you find your ”county road” takes you onto a two-rutter and then a single rutter and then into a pasture and stops, or else it takes you into some farmer's backyard.
>
> So we navigate mostly by dead reckoning, and deduction from what clues we find. I keep a compass in one pocket for overcast days when the sun doesn't show directions and have the map mounted in a special carrier on top of the gas tank where I can keep track of miles from the last junction and know what to look for. With those tools and a lack of pressure to ”get somewhere” it works out fine[...]
>
> *Robert M. Pirsig, Zen and the Art of Motorcycle Maintenance*

The topic of this thesis is access control in service-oriented architectures.

Access control is about making sure that only those who are entitled to something can get to it. The lock on your door is an example of access control; if it functions correctly, it makes sure that only people who have the correct key can enter your house. However, if you accidentally leave a window open, or if you forget to lock your house, you may still have undesired guests. A similar concept exists for computers; not everyone should have access to the data that you store on your computer, have access to your Internet connection, or use your computer's processing power for things that you do not approve of.

Securing a computer from unauthorized access is much more difficult than securing a house. The outer perimeter of a house is fairly easy to understand. Once you have secured your doors and your windows, you are reasonably safe from an intruder. Of course, there is always the possibility that someone takes a large shovel and tears a hole in your wall, but that is highly unlikely, and if it does happen, easy to detect.

In the case of a computer, there are many layers of protection, and they all need to work flawlessly. The working domain for computer security specialists ranges from the network level, the computer's operating system(s), to system services, applications, and messages that are exchanged by applications.

The focus of this research is access control for service-oriented architectures. Service-oriented computing is a field in computing that views *distributed information systems* as (autonomous) services that provide operations to other computer systems.

While service-oriented computing is in the spotlight of both industry and academia, surprisingly little progress is made in describing a conceptual framework on which services are implemented. This is surprising, since large scale web service deployments across multiple applications and services requires appropriate security, and particularly access control mechanisms are necessary to ensure that while a complex business process flows from one activity to the next, only authorized actors can invoke the supporting web services (Leune et al., 2004b).

We propose an framework that ingrains access control in service-oriented architectures (SOAs). The benefits of having a conceptual framework for SOA are:

1. A conceptual framework provides a set of named concepts, and describes the relationship between them, for further service-oriented developments.

2. A conceptual framework will provide a means to correlate many of the technical specifications that are currently under development, and it will provide a solid base for further technical developments.

3. A conceptual framework will allow for the development of a holistic approach to transcend from a technical approach to service-oriented computing to a more business-oriented approach.

The availability of a common framework will benefit service security in particular. While some efforts are made to present a coherent road-map for security in service-oriented computing, those efforts lack a common conceptual foundation and progress only slowly.

We address these problems by developing an access control framework for service-oriented architectures called EFSOC.

The majority of the research that is presented in this thesis takes place in a combination of fields. We explore the fields of service-oriented computing, event-driven architectures and security. Especially service-oriented computing (SOC) and security are fields which are currently very much in the spotlight of the industry and attract a lot of interest from all layers of our diverse community.

1.1 EFSOC

To address the under-developed nature of access control in service-oriented architectures, we designed an event-driven framework in which access control is a pri-

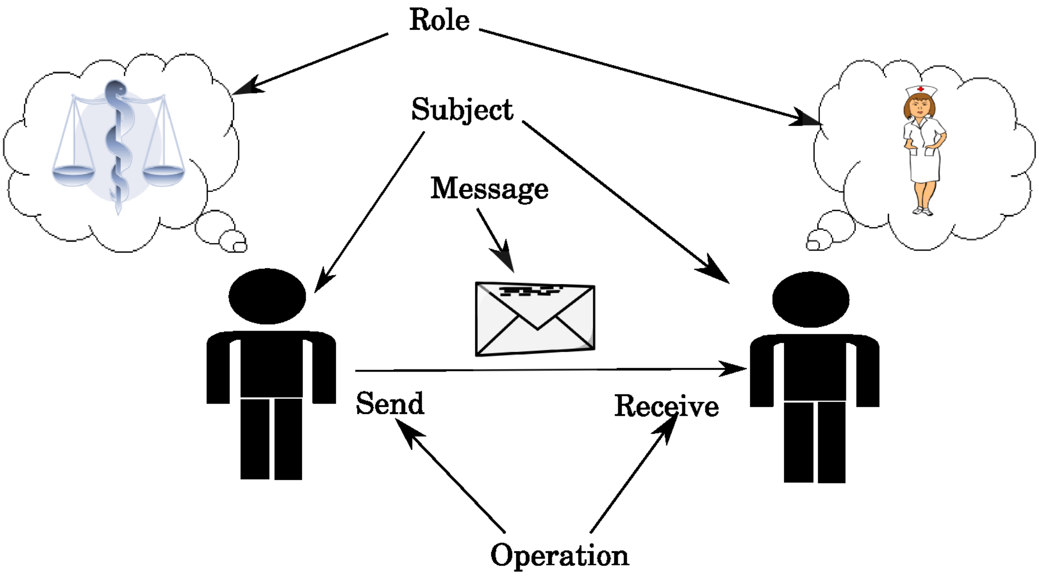


Figure 1.1: EFSOC overview

mary design objective. The framework—called EFSOC, or Event-Driven Framework for Service-Oriented Computing—combines the strengths of role-based access control and event-driven systems and applies them to service-oriented computing. The basic framework is shown in Figure 1.1.

Subjects may play certain roles and communicate by sending messages, which are generated as a result of some kind of event. Subjects can be real people, agents, information systems, services, etc. However, in the EFSOC framework, we consider only one type of subject: the service. Each service consists of one or more operations, or of other services.

1.2 Research Motivation

Web services technology started gaining attention in the early 2000's and was heralded as a new paradigm to design and develop distributed software components. Fairly quickly after the initial introduction of web services, visionaries started to see a brighter future for the same technology, and they adopted the concept of *services* that consists of *operations* as a way to view organizations as dynamic entities that are constantly creating, invoking, and discontinuing services.

When this research project started, it was not clearly understood what web services were, and indeed, if they really had a future. What was understood is that to become successful, web services—more precisely, service-oriented architectures—should fulfill a large number of business requirements.

In the research presented in this thesis, services are studied with the objective to investigate in which way access control considerations can be addressed by organizations that adopt service-orientation as a way forward. When the project started, security—and especially access control—was something that people had heard about, but it compared in no way to the massive attention that it is getting now. The field of security is a broad one; it ranges from very specific mathematical applications in the field of cryptography, to management approaches for large corporations.

Engineering secure systems relies on a large number of enabling technologies (e.g., cryptography, auditing, messaging, etc), which in themselves are mostly well-understood. However, the application of these technologies, especially in emerging areas, such as service-oriented computing, requires additional research to increase the level of understanding and knowledge that is required.

Security for service-oriented computing must ensure the availability of services, the correctness of messages that are exchanged between services and the confidentiality of those messages. In part, this is achieved by specifying and enforcing appropriate access control systems.

While the entire spectrum of security in service-oriented computing needs attention, this research focuses on *access control in SOA*. While this is the focus of my research, authorization and access control have close relations with other aspects of service-oriented computing.

The current state-of-the-art in service-oriented computing is mostly technology-driven and narrowly focussed. Many of the initiatives lack firm grounding on a commonly accepted conceptual model, yet address only very narrow aspects of the overall domain. This results in a plethora of standards and products that need to be tuned finely to interoperate effectively.

In addition, much of the technological developments have not matured, or even reached the stage of adolescence. Many of the current standards are developed by large consortia of commercial organizations. This is reflected in many of the WS-* standards, which often appear as compromises that provide no clear direction in which the technology should develop.

Finally, access control considerations are currently only minimally presented in service-oriented computing. IBM and Microsoft developed a security road-map for SOC, however the protocol that addresses access control has yet to be addressed (Web Services Security Roadmap, 2002). While the Globus alliance[1] has developed initiatives towards access control, no explicit conceptual model underlays its access control model. Rather, it is based on the use of access control lists, augmented by the ability to implemented custom authorization modules.

1.3 Research goal and scope of the research

This research aims to further understand the problems outlined in the previous section, their causes, and ways to mitigate them. More specifically, this research can be positioned on the intersection of the fields *Security & Access Control*, *Service-Oriented Architectures*, and *Event-Driven Processing*.

The goal of this research is formulated as follows:

---

[1] `http://www.globus.org`

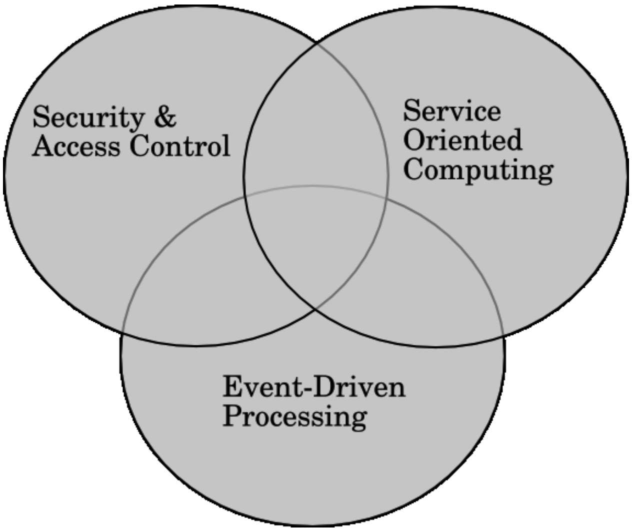


Figure 1.2: Positioning of research

> *Design, develop and validate a framework for inter-organizational and intra-organizational access control in service-oriented architectures.*

Security is a broad research field, ranging from highly mathematical theories describing cryptographic algorithms, to business theories about risk management and business continuity planning. This research will take place somewhere in the middle of that playing field.

The research presented in this thesis will take place at two levels:

1. *Conceptual level*
   At the conceptual level, we introduce a number of concepts that unify theory from event-driven systems, security (especially access control) and service-oriented computing into the EFSOC Service-Oriented Architecture. The EFSOC Architecture is described in-depth in Chapters 3, 4, and 6. Throughout those chapters, we illustrate our proposal with a running example that is derived from the medical domain.

2. *Operational level*
   The framework that is created at the conceptual level is partially implemented by a configurable prototype. The prototype implementation utilitizes state-of-the-art programming techniques and builds on current technologies.

At the conceptual level, we use Unified Modeling Language (UML 2.0, 2005) class diagrams to represent the (static) vocabulary elements of our model, as well as the dynamic behavior of the model.

The validation of this research will take place in three stages. First, we will conduct a case study in the medical domain. Based on interviews conducted with domain specialists at several hospitals, we will draw a realistic patient-care scenario. The scenario will subsequently be expressed in the EFSOC vocabulary and grammar.

Second, we will implement the configurable software prototype. This operational implementation will illustrate the feasibility of taking the theory and turning it into practice.

Third, we will configure the prototype with the data from the case study to create a laboratory experiment.

1.4 Research questions

To achieve our research goal, we formulate the following questions:

1. What is the state-of-the-art in access control? Here we perform an in-depth study of access control models and analyze them to find their strengths and weaknesses.
2. Is the current state-of-the-art of access control adequate for use in service-oriented computing? This question can be further subdivided in the following questions:
    (a) What are the security requirements for SOC?
    (b) How are requirements of security in general, and access control in particular, addressed in service-oriented computing?

We will design a framework to capture access control requirements for service-oriented architectures. The creation of the framework induces the following questions:

3. In what way should a definition language that supports the framework be formed?
4. In what way should a query language that supports the framework be formed?
5. What does an architecture for implementing the framework, the definition language and the query language, look like?

1.5 Research methodology

This section outlines the methodology that was taken to conduct this research. A methodology is defined as a set of techniques to meet a predefined goal (Welke, 1981). The set of techniques is also referred to as a method, which can be supported by a tool. A methodology should prescribe the sequence in which the techniques need to be performed across (part of) a software development cycle, and offer some rules for checking their consistency (van den Heuvel, 2002).

Figure 1.3 graphically represents the phases of the methodology that has been adopted for this research.

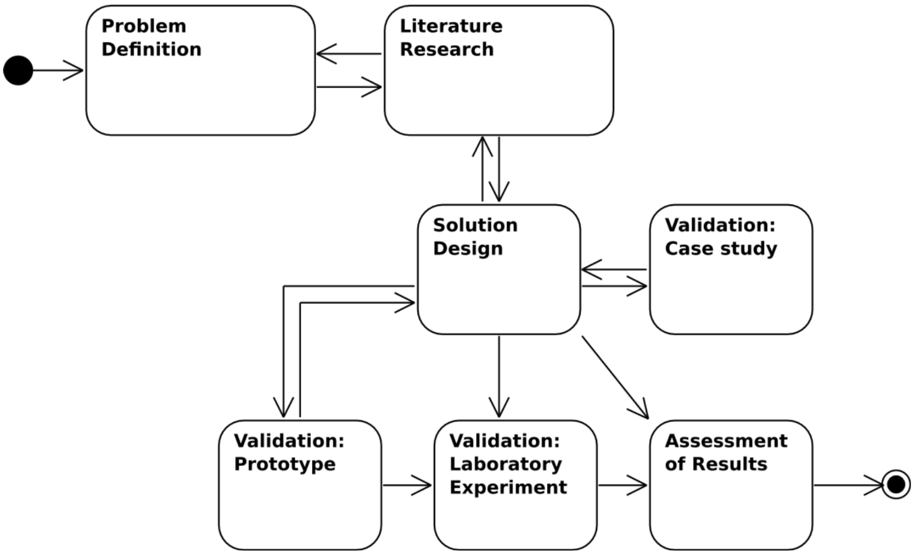


Figure 1.3: Research methodology

1. *Problem Definition*
   The first step in any research project is to understand the problem that is being investigated to an extent that a start can be made with formulating preliminary hypothesis.

   When the research progresses, and the understanding of the material deepens, the problem definition phase usually goes through several iterations in which the formulation of the problem and the research goals becomes more concrete and the scope of the research is delineated more clearly.

2. *Literature Research*
   While listed as a separate step, literature research is an ongoing process. Literature research serves a twofold purpose. First, literature research provides a good tool to get familiar with the subject matter at hand, and—provided it is executed correctly—will quickly bring the reader up-to-date on the current state-of-the-art. Second, literature research can provide a starting point to the relevance of the research; in addition to gaining insight in what is perceived as achievements, it also offers a way to ground the research in other efforts.

   The literature research will consist of a study of previously published material, accompanied by a projection of the published material on the research domain.

3. *Solution Design*
   Based on the results obtained in the problem definition phase and the literature research phase, the solution design phase attempts to formulate a theory that addresses the limitations or shortcomings that were identified. The resulting design must serve as a proposal to further the state-of-the-art.

In the solution design phase, we will adopt a top-down approach. We will begin by identifying a number of general principles to which the service-oriented architecture that will be designed must adhere. Next, the conceptual framework will be specified in a number of iterations. The conceptual framework design is followed by the proposal for a language that can be used to express the vocabulary and grammar that are part of the conceptual design. Finally, a formal representation of the language is proposed to allow comparison to other approaches, and to prove formal soundness of the language.

4. *Validation*

   The validation phase takes the results of the solution design phase, and attempts to make it plausible that the chosen design is adequate and correct.

   The validity of our approach is captured by establishing the accuracy, meaningfulness and the credibility of our proposal. The process of establishing the validity of an approach can be divided in establishing the internal validity and establishing the external validity.

   The external validity of the approach is established when the proposed solution can be successfully applied to other contexts. The external validity of EFSOC was shown by conducting a case study, which is presented in Section 3.2.

   An approach is internally valid if the conclusions that are drawn are valid from the problem solution that is suggested.

   We will validate this research empirically by executing the following activities:

   (a) *Case Study*
   We will describe a case study in a domain that has enough complexity that there will be many organization-spanning message exchanges. The case study will yield a description in natural language of one or more processes. It will then be attempted to express the case study in terms of the proposed solution.

   Using a real-world scenario, rather than limiting ourselves to a controlled laboratory experiment, exposed our approach to a wider range of problems. Performing the case study, and projecting the case study's results on our approach will establish the credibility and the meaningfulness of our approach.

   (b) *Formal Foundations*

   In this research, we establish the internal validity of our approach in Chapter 6, which presents a formalized definition of EFSOC. The formalization will present a model that is internally consistent, and axiomatically complete.

(c) *Prototype*
In parallel with the case study, we will develop a software prototype that will show that our solution is achievable. The solution will take the form of proof-of-concept using software that is freely available. The development of the prototype and the proofs-of-concept does not play a role in establishing the validity of our approach. Rather, it is a necessary input for the laboratory experiment that is discussed in the next bullet.

(d) *Laboratory Experiment*
The final phase of the validation process is to take the results from the case study and input them into the prototype. This will enable us to run a laboratory experiment in which the practical applicability of our solution can be tested.

We will prepare two sets of interpretations of the model that we propose. The first set reflects desired states of the model, and should fully be accepted by the prototype. The second set reflects a set of undesired states of the model, and as such, it should be fully rejected by the prototype.

Executing the laboratory experiment will establish the accuracy of our approach.

5. *Assessment of the Results*
The final stage of the research project is to look back on the theory that was developed and the results of the validation phases. In this phase, we will identify in which respects our solution will provide added benefits, as well as the conditions under which these benefits can be leveraged the best. In addition, we will identify opportunities for further research.

1.6 Contributions

This section briefly summarizes the contributions of this research. A more detailed description is available in Section 8.4. Current approaches to access control in distributed environments tend to focus on centrally administered access control policies, which are deployed to decentralized points of enforcement. In service-oriented architectures, this is not valid.

Contribution 1 *We introduced an access control model for service-oriented architectures that is decentrally administered, yet centrally enforced.*

Of all the access control models, Role-Based Access Control is often heralded as the most modern approach, and as an approach that closely aligns with the way that enterprises organize their processes. EFSOC adopts a role-based approach, but acknowledges the fact that some permissions must not be assigned to roles, but should be assigned to individuals within a role.

Contribution 2 *EFSOC adopts a discretionary role-based access control model.*

Service-oriented architectures are often not based on a shared conceptual model and do not use consistent vocabularies to express access control requirements.

Contribution 3 *EFSOC provides a common methodological framework and provides a reference architecture for decentrally managed, yet centrally enforced, discretionary role-based access control.*

1.7 Structure of this thesis

The remainder of this chapter is structured as follows. Since this research takes place on the intersection of several research fields, Chapter 2 covers background information on event-driven systems (Section 2.4), security and access control (Section 2.2) and service-oriented computing (Section 2.3). Chapter 3 introduces the EFSOC Service-Oriented Architecture, which we believe provides a value-added layer on top of web services technology. A language to specify elements of EFSOC, and execute it in a dynamic environment is discussed in Chapter 4. EFSOC Query Language (EQL) is described in detail in Chapter 5. Chapter 6 provides a formal grounding for that language.

The structure of the thesis is graphically represented in Figure 1.4. Throughout the thesis, we will refer to a case study that was performed in the medical domain, in which a surgery patient is followed from its initial point of contact to his discharge from hospital care. The case study is discussed in Section 3.2.

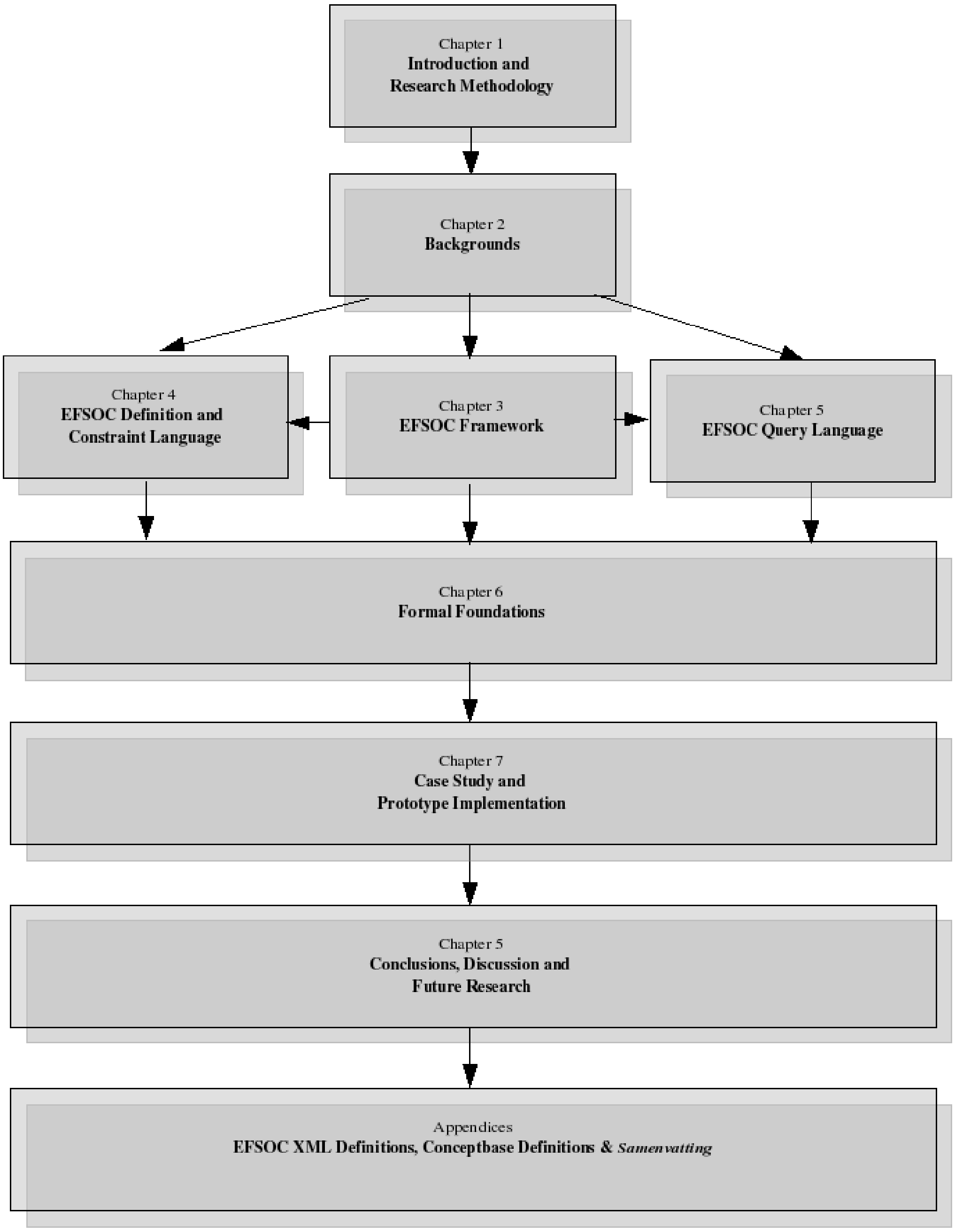

Chapter 1
Introduction and Research Methodology
Chapter 2
Backgrounds
Chapter 4
EFSOC Definition and Constraint Language
Chapter 3
EFSOC Framework
Chapter 5
EFSOC Query Language
Chapter 6
Formal Foundations
Chapter 7
Case Study and Prototype Implementation
Chapter 5
Conclusions, Discussion and Future Research
Appendices
EFSOC XML Definitions, Conceptbase Definitions & *Samenvatting*

Figure 1.4: Thesis outline

Part I

Background and Theory

Chapter 2

Background in SOA, Security and Event-Driven Processing

> “What terrifies you most in purity,” I asked?
> “Haste,” William answered.
>
> *Umberto Eco, The Name of the Rose, Fifth Day, Nones*

2.1 Introduction

The research that is presented in this thesis is positioned on the intersection of a multitude of research fields. This chapter provides the reader with background information on each of these fields, and includes a discussion on observations that were made in each of them.

Section 2.2 explores the (very wide) research area of information security, computer security and network security. The focal point of Section 2.2 is a discussion of popular access control models.

Section 2.3 introduces service-oriented computing, with a focus on loose couplings, service composition and web services technology.

Next, we will explore event-driven messaging in Section 2.4.

The chapter will be concluded in Section 2.5, with a discussion of the previous sections and an outlook to security requirements for a service-oriented architecture in Section 2.6.4.

2.2 Service Security

Information security addresses the need to protect resources from unauthorized use, manipulation, or inspection so that the availability of services can be guaranteed so that business processes can continue to operate. The security field is traditionally divided in a number of sub-fields:

1. *Authentication* is the process of determining the identity of a person or an object. To be able to rely on a proper authentication scheme is a fundamental requirement for any security infrastructure.

2. *Authorization* is the process of determining what permissions a person has. Typically, access control results in some subjects receiving a set of authorizations to perform certain actions on certain objects.

3. *Confidentiality* is the process of ensuring that message contents are only available to duly authorized subjects.

4. *Integrity* is the process of ensuring that messages are not manipulated while they are in transit. That is, integrity ensures that messages are received in exactly the same way as they were sent.

5. *Auditing* is the process of keeping a trail of things that happened for later reference. Audit trails can be used to establish *non-repudiation*, i.e. the process unambiguously establishes the fact that something happened, or for conflict resolution. Auditing is also used to detect security breaches and to get an impression of the consequences of such actions.

In the following sections, each of the above aspects will be discussed in more detail.

2.2.1 Authentication

In computer security, authentication is often defined as the process by which someone or something attempts to confirm its identity to someone else (Sandhu and Samarati, 1996).

The goal of an authentication scheme is to unambiguously determine the identity of a subject, and to make it impossible to assume an false identity.

After authentication, two principals (people, computers, services) should be entitled to believe that they are communicating with each other and not with intruders (Burrows et al., 1990).

Authentication is an important process, since it provides the basis for determining what rights a person will be granted in a later stage.

Authentication approaches are typically based on *shared credentials*. What is meant by this is that the subject who desires to become authenticated has the ability to produce something that can be validated by the authenticator. A typical example of using shared credentials is using a password to log in to a computer, or using a PIN code on an ATM card to withdraw funds from a bank machine. In both situations, the credentials must be known by the subject and can be validated by the authenticator.

Authentication methods are generally divided in three categories:

1. *Authentication, by knowledge*
   The most common authentication category that is currently in use in computer security is password based authentication. Password based authentication assumes that a subject knows a password, which is also known to the authenticator. Only when the user provides the correct password, he will be authenticated. Any authentication that revolves around a subject having some kind of knowledge that can be tested by the authenticator belongs to this category.

2. *Authentication, by possession*
   Anyone who regularly travels internationally is familiar with passport authentication. The subject (i.e. the traveler) is issued a document which entitles him to cross borders and receive protection while residing in a foreign country. The agency that issued the document may have added certain features which ensure the validity of the document. In additional to showing the authentication document, no additional actions to establish an identity are required.

3. *Authentication, by being*
   Biometrics is the science and technology of determining the identity of a subject physiological features. The most common forms of biometrics are based on a subject's fingerprints, retinal patterns, or facial features. Less common, but still in use, are voice recognition and handwriting analysis. All of these approaches assume that a subject can be identified, with a degree of statistical certainty, based on one or more of such features.

A good authentication method typically shares at least two of the three categories. For example, simple password authentication is generally not considered to be strong enough, as it only belongs to the second category. However, when password authentication is combined with, for example, a certificate on a chip card, or with some form of biometric analysis, the authentication scheme becomes stronger and therefore it becomes harder to deceive an authenticator.

2.2.2 Integrity and Confidentiality

Integrity and confidentiality are message-level protections to prevent a message from being altered by anyone else than its original sender, resp. to prevent anybody else but the intended recipients of the message to learn its contents.

Integrity and confidentiality are closely related concepts, since they are often achieved using very similar technology.

Message integrity is commonly achieved by digitally signing messages. Not only allows digital signing the sender of a message to be established unambiguously, it also provides a means to detect if any changes were made after the message was signed. When a message's content is altered after a signature has been placed, that signature will no longer be correct and a breach in the message's integrity can be detected.

Notice that digitally signing messages does not prevent messages from getting altered. It does, however, provide the final recipient with a way to *detect* that the message that he received is not the same message that was signed.

Message encryption is the process of changing the contents of a message in such a way that its contents do not make sense to anyone, including the intended recipient. However, that recipient will have the means to undo the changes that were made in order to re-create the original message. Most often, messages are encrypted using cipher algorithms that replace characters, or sequences of characters, according to certain mathematical rules.

2.2.3 Public Key Infrastructures

A public key infrastructure is a system designed to authenticate users using digitally signed certificates that are issued by trusted third parties. A public key infrastructure usually consists of one or more *certificate authorities* (CA's), which may issue *certificates* to *end-entities* (users) or to other CA's of who they have verified the identity.

A typical application of public key infrastructures is found in SSL-enabled web servers. A service operator will obtain a digital certificate from a CA, which is expected to establish the service's identity. When the CA decides that the service is who it claims to be, a certificate will be issued which carries the CA's digital signatures.

When other services wish to bind to the service, they will be able to check its certificate and establish the fact that it was signed by the CA. When this CA is trusted by both parties, a level of trust between them can be assumed.

It is important to realize that a PKI can only be used to establish *identities*, and to provide a trusted way to exchange public keys. Even with a valid certificate, a service may still not be a reliable trading parter.

Using a PKI approach offers a number of benefits:

1. *Hierarchies of CA's*
   To prevent the need for having a large number of root certificate authorities, CA's may be organized in a hierarchy, as shown in Figure 2.1. The added benefit of this is that CA's can certify each other to issue valid certificates. Clients will only have to possess the root CA's certificate to validate a server's certificate to establish its identity.

2. *Centralized administration of certificates and revocations*
   Because CA's are organized in a hierarchy, it is always possible to establish a *certificate path* that can be used to find out if the certificate has been issued by a trusted authority. Even more interestingly is the ability for any CA in the hierarchy to *revoke* a certificate.

   Certificate revocation can be used when a CA that is located lower in the hierarchy has not functioned adequately and can no longer be trusted, or when an end-entity does not live up to the requirements that were agreed on when the certificate was issued.

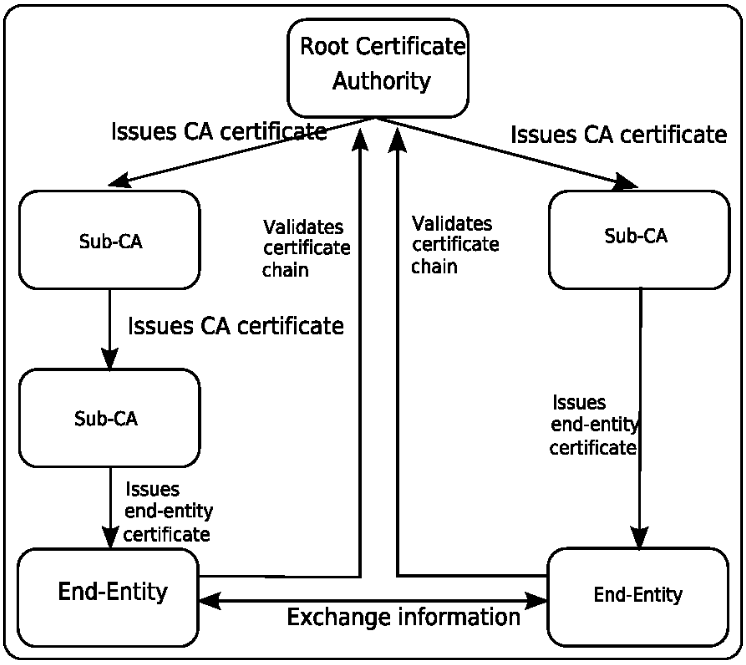

Root Certificate Authority
Issues CA certificate
Issues CA certificate
Sub-CA
Validates certificate chain
Validates certificate chain
Sub-CA
Issues CA certificate
Sub-CA
Issues end-entity certificate
Issues end-entity certificate
End-Entity
Exchange information
End-Entity

Figure 2.1: Public Key Infrastructure CA Hierarchy

The central administration of certificate chains makes a PKI infrastructure a powerful tool for use in centrally administered trust systems.

3. *Trusted third-parties*
Even if end-entities do not know each other well enough to be sure that they are indeed talking to the service who it claims to be, using trusted third parties allows them to establish a basic level of trust. Assume that a hospital and an ambulance company need to exchange confidential information, but have not yet formally verified their identities. Yet, the hospital and the ambulance service's identity have both been verified by the same CA, and the CA's identity has been established by both organizations. When the hospital and the ambulance service obtain certificates that are signed by the CA that they have in common, it becomes possible for them to exchange information by relying on the certificates that were issued.

2.2.4 Web of Trust

Unlike public key infrastructures, the web of trust does not adopt any central administration of certificates and/or keys. Instead, each user in the web of trust is expected to generate his own key pair (consisting of a public key and a private key).

Asymmetric encryption requires that each user possesses two mathematically linked keys. One of the keys must be made publicly available and one must be kept secret. Using the key pair, messages can be encrypted, digitally signed or both.

Assume that John wants to send a message to Mary that needs to be kept confidential. In other words, nobody except Mary must be able to understand the contents of the message. To achieve this objective, John needs to obtain Mary's public key. Since that key is not secret, Mary can send it to John any way she wants, or even publish it on a web site. John will use that key to encrypt the message, and send the result to Mary.

In order for Mary to be able to decrypt the message, she needs to use her private key.

Next, Mary wants to send a message that is not confidential back to John. However, Mary wants to ensure that John can establish with a high degree of likeliness that Mary is indeed the sender of the message. Mary can achieve this by digitally signing her message using her private key.

When John receives the message, he can use Mary's public key to verify that she indeed sent the message.

In the web of trust approach, trust is established by allowing people to place a digital signature (using their own private key) on somebody else's public key, and than share it with the community.

The underlying assumption is that somebody with many signatures on his key is somebody who's identity has been established by many different people.

While signatures on a public key may provide a certain level of confidence in the identity of the owner of the key, the web of trust decouples identity from trust. In the web of trust, trust is considered something that is local to a user.

Assume a situation in which a user *Jack* has a public key that has the signature of *John* on it, as outlined in Figure 2.2. *Mary*, needs to send Jack an encrypted message, but she does not know him. Yet, Mary wants to ensure Jack's identity before sending the message because she wants to prevent disclosure of information to the wrong person. Fortunately, Mary does know (and trust) John. In turn, John does know Jack and has signed his public key to attest to this. When Mary retrieves Jack's public key, and sees that it carries John's signature, she is confident enough to use it to send Jack an encrypted message.

The Web of Trust gains it strength from the fact that it has a large number of users. Since credibility of keys depends on obtaining signatures from other users, a large user-base is required for the web of trust to function well.

The most common software on which the web of trust relies consists of the GNU Privacy Guard[1], and software published by the PGP company[2]. Both of these applications have the disadvantage that they are fairly technical in nature. However, with the advent of user-friendly GUI front-ends, the web of trust is gaining popularity fairly quickly.

The most common application domain of the web of trust is in digital signing and/or encryption of email. While that is the reason for the majority of users to participate in the web of trust, there is no reason why it cannot not be used for the protection of other messages.

A disadvantage of the web of trust approach in a corporate setting is that nobody is able to control who issues digital signatures, and under which conditions. In the

---

[1] see http://www.gnupg.org

[2] see http://www.pgp.com

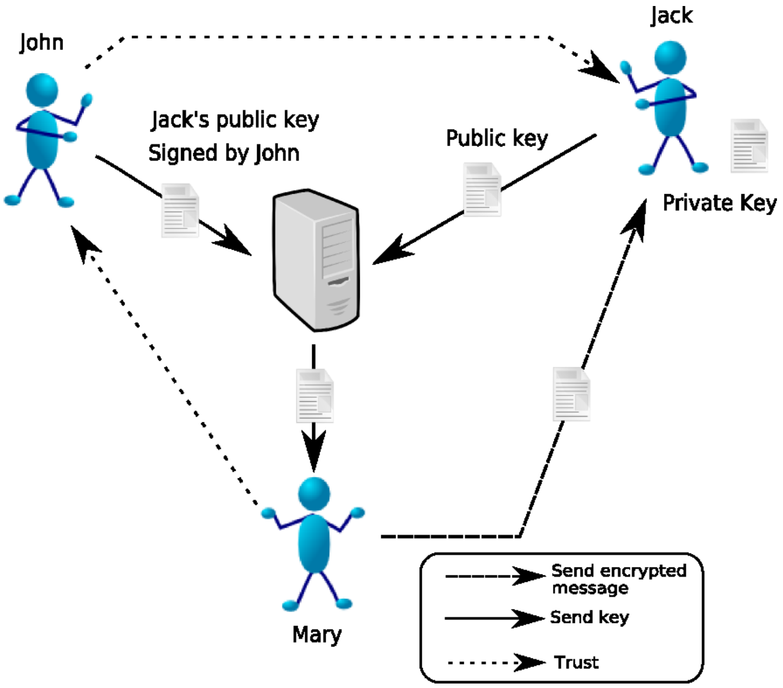


Figure 2.2: Web of Trust

web of trust approach, nobody can present a user from signing everybody's public key, regardless of whether he has verified the owner's identity. It is assumed that such situations self-regulate themselves. First, when it becomes known that the user adopts these practices, it will become virtually impossible for him to accumulate digital signatures which will give him credibility. Second, the web of trust assumes that a user with multiple signatures is more likely to be who he claims to be than someone with just one or two signatures.

2.2.5 Access Control

To develop secure systems, security should be considered at all stages of design, so that the design not only satisfies its functional specifications but also satisfies security requirements. To do this we need to start with high-level models that represent the security policies of the organization(Schumacher et al., 2006).

Access control refers to the process of ensuring that no resources are used in an unauthorized way. Two basic principles underlay access control:

1. *The Least Privilege Principle*
   The least privilege principle states that a user should only have the absolute minimum of privileges to perform his job at that point in time (Saltzer and Schroeder, 1975). Any additional privileges may be abused and can lead to circumvention of security mechanisms that are in place. In the context of SOA, the least privilege principle thus postulates that the only interaction with services may take place in the context of an executing business process.

2. *Separation of Duty*
   This principle aims at preventing potential fraud by disseminating responsibilities for the execution of fragments of a business process among several participants. The classic example in administrative theory is that the manager of the inventory should never be the same person as the one who is responsible for purchase requisition. Separation of duty can be enforced statically and dynamically. Static separation of duties is a very restrictive technique that aims at avoiding conflicting roles, conflicting permissions, conflicting users and conflicting tasks. Dynamic separation of roles on the other hand, enforces the activation of roles according to a policy specification of the required separation of duty in the context of a business process. Similar to its static counterpart, four dimensions of conflicts need to be circumvented, viz. dynamically conflicting roles, dynamically conflicting duties, dynamically conflicting users and dynamically conflicting tasks. The interested reader is referred to (Botha and Eloff, 2001) for an in-depth examination of structural and dynamic separation of duties in the context of workflow systems.

Over the past few years, a number of access control models were developed including discretionary access control, mandatory access control, and role-based access control. We will review these techniques briefly and discuss their relevance in the domain of SOAs in the following.

2.2.6 Discretionary Access Control

The prevalent model for access control is the discretionary access control model (DAC). Discretionary access control is a means of restricting access to objects based on the identity of subjects and/or groups to which they belong (Neon Orange Book, 1987).

In DAC, access control requirements are specified in the form of an *access matrix*. The access matrix is conceptually simple and the majority of access control models are based on it (Hayton et al., 1998).

An access control matrix is a data structure with a row for every subject and a column for every object (Harrison et al., 1976). Each cell of the table contains the access rights of the subject on the object. As a side-effect, the rows of the table will provide a capability list per subject, while the columns describe in detail what kind of access is allowed for each object.

Despite the fact that DAC is widely adopted, it is a well known fact that there are several fundamental shortcomings which make it less suitable for large-scale use in dynamic, heterogeneous environments (M. Krause, 1999). Firstly, discretionary access control adopts the assumption that the owner of an object controls access to it. When translated to the context of service-oriented architecture, this means that the owner of a service, hence the service provider, controls the access to the services it advertises in its interface. A restriction of DAC that makes it less suitable for SOA's is that it cannot deal with additional security requirements of clients.

Secondly, the DAC approach regulates access by using access control matrix. The access control matrix denotes a matrix in which all subjects are cross-referenced and permissions are defined for each possible combination (object-operation)-subject pairs. While this may be a viable solution for small scale business processes that need a limited number of web-services, it will quickly become obsolete and too costly for more complex and distributed business processes that may involve many more services.

2.2.7 Mandatory Access Control

Mandatory Access Control (MAC), also referred to as Multilevel Access Control, originates from military applications and was designed to regulate the flow of information. In (Orange Book, 1985), it is defined as a means of restricting access to objects based on a mapping between the sensitivity of the information that is contained in the objects and the formal authorization of subjects to access this information. This security model forbids creators of information to manage access to them; instead, access is controlled by a central security administrator who designs a hierarchy of security levels and administers the assignment of security levels between objects and subjects. If a subject is operating at a security clearance which is at least as high as the security classification of the object, access is granted.

The most well-known mandatory access control models are the Bell-LaPadula Model of 1973, the Biba Access Control Model of 1977 and the Clark-Wilson Access Control Model of 1987.

The Bell-LaPadula Model

The Bell-LaPadula model (BLP) (Bell and LaPadula, 1973), was the first successful attempt at mathematically describing an access control paradigm based on two simple rules: *no read up* and *no write down*. These rules are known as the *simple security property*, resp. the **-property*. In BLP, all objects and all subject have a fixed security class. The simple security rule states that subjects can only read an object when its security class is equal to, or lower than the security class of the subject. The star property states that a subject may only write to an object if it has a security class which is equal to, or higher than the security class of the subject.

The Biba Model

Whereas the Bell-LaPadula model is a model to enforce *confidentiality*, the Biba model (Biba, 1977) was designed to ensure *integrity*. The Biba model is very similar to BLP. It contains two rules: *no write up* and *no read down*. These rules are known as the *simple integrity property*, resp. the *integrity *-property*. The simple integrity property requires that subjects can only modify objects which are below the integrity class of the subject. The integrity star property requires that subjects can read objects with a security class that is higher than the subject's security class.

The Clark-Wilson Model

The Bell-LaPadula Model and the Biba model both originate from the military domain. In 1987, Clark and Wilson published *A Comparison of Military Computer Security Policies* (Clark and Wilson, 1987). In that article, they proposed a policy for *well-formed transactions*. The Clark-Wilson model assumes that some data items and constrained so that they can be acted on only by certain *transformation procedures*. These procedures takes an *unconstrained data item* and turn it into a *constrained data item*. Clark Wilson suggest that there is a set of *integrity verification procedures* that check the validity of constrained data items, and a set of *transformation procedures*, which ensure the integrity of constrained data items.

Access control is specified by triples *(subject, transformation procedure, constrained data item)*, which specify exactly for each subject, which transformation procedures may be performed on constrained data items.

MAC and SOA

Mandatory access control systems that are variations of Bell-LaPadula or Biba, are not particularly suited for usage in SOAs due to the application of rather static and restrictive hierarchical security levels.

A more flexible approach is required that allows for dynamic (re-)definition, and definition of security policies at the level of service operations in a networked manner. The Clark-Wilson model offers more potential, however the management overhead of maintaining the access control tuples will be too high.

Moreover, the assumption that access is administered by a central administration, is conflicting with the peer-to-peer nature of SOAs.

2.2.8 Role-Based Access Control

Role-based access control models (Sandhu et al., 2000), (Ferraiolo et al., 1999), (Hamada, 1998) are a contemporary control technique, that introduces an indirection layer between users and permissions which logically separates the role that users play in an organization or process from the subjects. Hence, in a role-based approach permissions are assigned to *roles* and roles are assigned to *subjects*, as graphically illustrated in Figure 2.3. *Permissions* are used to specify the capability of subjects to perform a particular operation, e.g., specifying an incident report or assessing incidents. Typical in an RBAC approach, roles can be organized in role hierarchies and constraints can be defined between roles.

RBAC and SOA

An implicit assumption of the role-based approach is that the combination between roles and permissions is relatively stable in time (Botha and Eloff, 2001). The volatile environment in which web-services collaborate for implementing business processes however, requires an environment in which roles and permissions *are decoupled* to cater changes in the allocations of permissions to roles (Leune et al.,

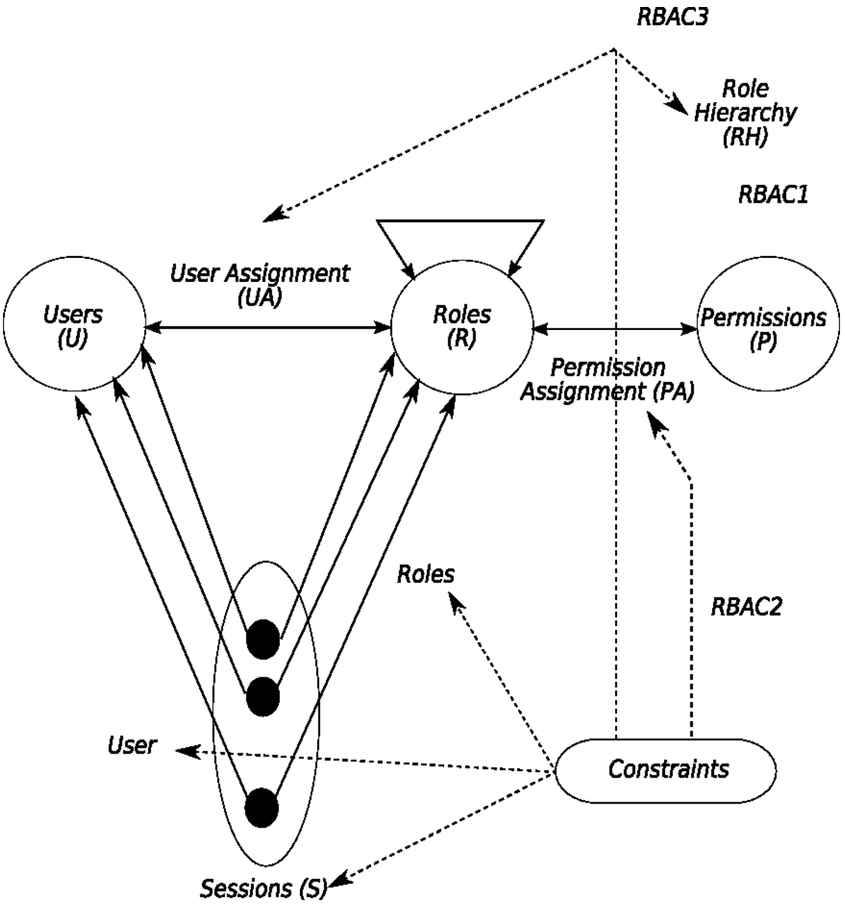

RBAC3
Role Hierarchy (RH)
RBAC1
User Assignment (UA)
Users (U)
Roles (R)
Permissions (P)
Permission Assignment (PA)
Roles
RBAC2
User
Constraints
Sessions (S)

Figure 2.3: Role-Based Access Control (Sandhu et al., 1996)

2004a). This demands dynamic generation of permission-to-role assignments, using knowledge about the roles that subjects play and about other contextual factors. Applying dynamic evaluation of authorization rules to deduce access control permissions not only allows for a better enforcement of the principle of least privilege, but also enables a near real-time active security implementation.

Another obstacle for applying RBAC in SOAs is that, similar to MAC, RBAC prescribes that the security administration takes place in a single, centralized, location. At the same time, this security model assumes that access control rules are typically evaluated by a central security agency, which introduces additional limitations on the flexibility and distributed nature that are implied in the service-oriented environment.

In conclusion, existing access control models suffer from shortcomings may severely hinder their applicability to the SOA domain. The EFSOC framework that is introduced in the next section tries to overcome these observed difficulties and allows for dynamic (re-)definition of authorizations. Before outlining this framework, we wish to introduce a running example, to demonstrate the workings of EFSOC.

2.2.9 Auditing

Business critical information systems should keep track of all events that take place, and store a description of such events in an *audit trail*. While establishing an audit trail will not prevent security breaches from taking place, audit trail data can be used to investigate the trail of events after they happened, and provide a means to assess the impact of security breaches.

In addition to monitor *what* happened, audit data can also be used to establish *who* performed certain actions, and hold them accountable for their actions.

Of course, auditing data is very valuable to attackers, since it will allow them to quickly find out many things about an organization, such as network topology, information systems that are used, key players, etc. It is therefore recommended that audit data is kept *at least* as secure as any other business-critical data, and that well-defined policies are in place about notifying users about the fact that an audit trail is collected, for which purposes it will be used, and how long it is stored.

Recent developments in legislation, such as the Sarbanes-Oxley Act of 2002 (SOX), establish a set of requirements for corporations that are designed to deter fraud and increase corporate accountability.

For example, Section 103 of the Act requires the retention of *all audit-related records* (including electronic) for a period of seven years. Section 802 of the Act proceeds by laying down criminal penalties for altering documents. It requires mechanisms to ensure that data, once recorded, remains unaltered.

Compliance with the Act requires not only the establishment of an audit trail (i.e., the fact that a document was modified by a certain user), it also requires that the document (including its changes) are kept for a period of seven years.

The impact of SOX on the way that businesses are organized and governed is gigantic. Applied to the service-oriented architectures, the SOX legislation might mean that all messages that are exchanged by services may have to be kept for a similar period of time.

2.2.10 Security and Grid Services

A field that has adopted service-oriented computing is *grid computing*. Grid computing applies the resources of many computers that are connected by a network to work on solving a single problem at a time. The Grid computing model assumes that resources should be allocated or re-allocated on demand, using remote distributed computing facilities, and regardless of the location of the physical hardware.

The Grid computing community acknowledges the need for security and addresses it explicitly. In Grid computing, security is based on the use of X.509 certificates, entity certificates and proxy certificates, which are all used for identifying subjects. Security in Grid Services is divided in message-level security, i.e., protection of SOAP messages, Transport-level security and an Authorization Framework. The Authorization framework provides the ability to outsource authorization decisions using SAML messages authorization based on access control lists, and a mechanism to implement custom authorization modules.

2.2.11 Discussion

Security is a broad field that can be roughly characterized in authentication, authorization, integrity, confidentiality and auditing. By authentication, we mean the process of establishing a subject's identity. Confidentiality and integrity are message-level protections that are concerned with ensuring that a message's contents are not disclosed to anybody than its intended recipient and about ensuring that it is possible to show that the contents of a message have not changed since it was digitally signed.

The most common approach to achieve authentication, integrity and confidentiality can be found in cryptography using asymmetric key-pairs. Asymmetric cryptography works by keeping a secret key that must only be accessible by its owner, and a public key that can be freely exchanged.

We discussed two key management approaches: public key infrastructures using hierarchical certificate authorities that function as trusted third parties, and the web of trust, which relies fully on peer-to-peer signing of public keys.

Next, we introduced access control as a means to prevent unauthorized access to resources. Three common access control models were discussed: discretionary access control using access control matrices, mandatory access control using access control lettuces and role-based access control.

Finally, we discussed the need for establishing audit trails and for protecting and retaining audit trail data.

Since the research focus of this thesis is on access control in service-oriented architectures we will now assume that authentication, message-level integrity and message-level confidentiality are readily available.

2.3 Service-Oriented Computing

Service-Oriented Computing (SOC) is the computing paradigm that utilizes services as fundamental elements for developing applications (Papazoglou and Georgakapoulos, 2003). The paradigm of service-orientation is a relatively recent enrichment to the discipline of designing distributed applications. Its vision encompasses a future in which application development is not constrained by organizational or technological boundaries.

Instead, applications will be developed from the viewpoint of services which offer some kind of value to their consumers. New services may be designed and built from scratch, or they can be assembled by combining existing services with a value-added content.

In other words, services may be perceived as self-describing, open components that support rapid, low-cost composition of distributed applications.

Service providers publish platform-independent descriptions of the services that they offer using a machine-readable format, such as the WSDL (Christensen et al., 2001) specification. The WSDL specification is discussed in more detail in Section 2.3.4. In addition to technical specifications detailing how to invoke each service, such descriptions may contain additional information outlining constraints or requirements for invoking the services.

Service descriptions are published in central repositories where they may be discovered by service consumers. Based on the information in the service description, a consumer can now decide whether the service is suitable for the requirements. If this is the case, the consumer may contact his providers directly to invoke the specified services.

2.3.1 Service-Oriented Architecture

The concept of service-oriented computing (SOC) manifests itself in the form of a service-oriented architecture (SOA). The main difference between the two is that SOAs provide a *reference architecture* for implementing service-oriented computing. The basic SOA is discussed in the previous section and is illustrated by Figure 2.4.

While the basic SOA provides enough reference material for the basic publish-discover-bind process, aspects like service composition, service management, transactions and security are not covered.

Such concerns are addressed by the extended SOA (ESOA) (Papazoglou and Georgakapoulos, 2003) that is depicted in Figure 2.5. The ESOA consists of three layers in which the most basic one encompasses the basic SOA. The middle layer is known as the *composition layer* and provides the roles and functions that make it possible to compose new services out of existing ones. Service composition is discussed in more detail in Section 2.3.3.

The service composition layers includes functions for the following aspects:

1. Coordination: control the execution of the services that form a composition by specifying and enforcing workflows.

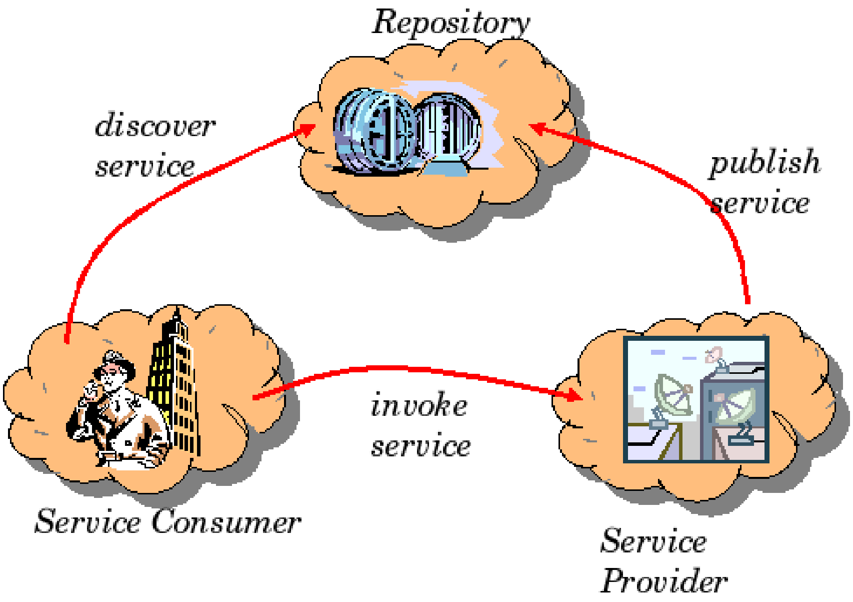

Repository
discover service
publish service
invoke service
Service Consumer
Service Provider

Figure 2.4: Publishing, Discovering and Invoking Services

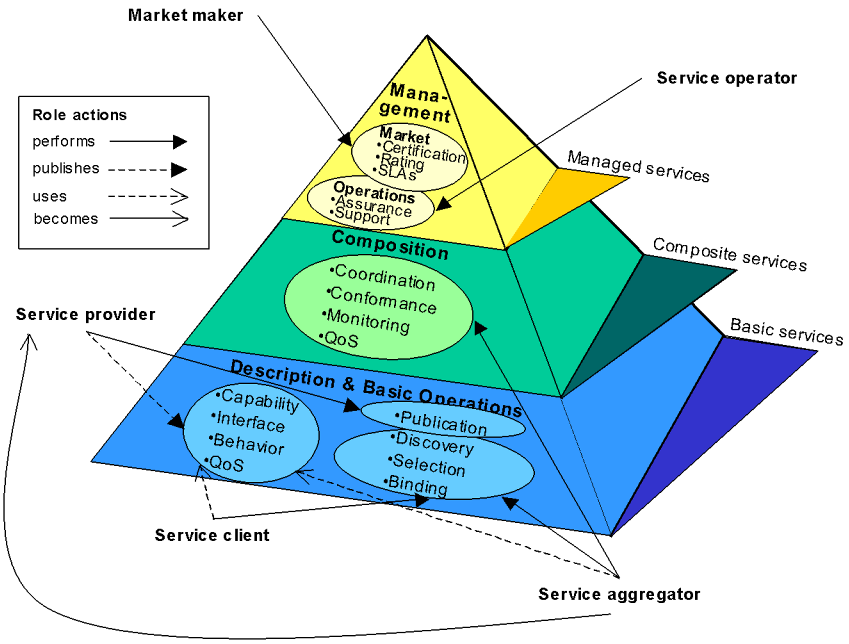

Market maker
Service operator
Role actions
performs
publishes
uses
becomes
Mana-gement
Market
•Certification
•Rating
•SLAs
Operations
•Assurance
•Support
Managed services
Composition
•Coordination
•Conformance
•Monitoring
•QoS
Composite services
Service provider
Description & Basic Operations
•Capability
•Interface
•Behavior
•QoS
•Publication
•Discovery
•Selection
•Binding
Basic services
Service client
Service aggregator

Figure 2.5: The Extended SOA (Papazoglou and Georgakapoulos, 2003)

2. Monitoring: allows services to monitor each other in order allow dynamic system which respond to each other.

3. Conformance: to ensure that compositions function the way that they are intended.

4. QoS composition: leverages, aggregates and bundles the component's QoS.

Considerations such as security are often positioned on the service composition layer.

The top of the ESOA pyramid is formed by the *ESOA, service management layer*. In particular, the service management layer provides functionality for service operations and for managing services in markets.

2.3.2 Loosely Coupled Message-Oriented Systems

Service-oriented computing is driven by the need for loosely coupled distributed systems, and the need to interact with those systems via platform independent interfaces. As such, Service-Oriented Computing is the logical next step in the evolution from tightly coupled EDI systems, via client-server architectures and object request brokers to loosely coupled message-oriented systems.

While web services are often considered to be loosely coupled, this is only partially true. It is important to realize that web services provide a service-oriented architecture. In other words, web services technology is 'just' an incarnation of the SOC concept.

Service-Oriented Computing, service are considered to be loosely coupled because their interfaces are described in an implementation-independent XML specification (called WSDL).

However, SOC also requires that services are location transparent (Papazoglou and Georgakapoulos, 2003). Web services require that interface descriptions are mapped onto implementations using bindings, and web services call each other on an implementation level, and not on an abstract level. By doing so, web services cannot be considered location transparent.

Furthermore, since web services must be bound to an underlying technology before they can be invoked, web services can be considered to adopt *late binding*. However, once a service is bound to an implementation, the services are still tightly bound.

We perceive this problem to be caused by the fact that web services are not truly message-oriented. While the WSDL documents describe the service's interfaces in a technology neutral fashion, the implementation of web services require a final binding.

2.3.3 Service Composition

One of the strengths of service-oriented computing lays in the fact that the paradigm allows services to be nested in other services. Service composition will allow service providers to quickly deploy new services that are based on existing ones, allowing them to quickly react to changes in the environment.

The ability to use services to create new services is dubiously labeled ‘service composition’, though we would have preferred the term ‘service aggregation’.

While deploying composite services provides service providers with a very convenient mechanism to quickly respond to changing circumstances, care has to be taken that a service does not become critically dependent on another one.

This can be illustrated by the following example. Assume that a medical hospital decides to outsource certain tasks, such as transcribing physician’s dictations, or digitally recording MRI scan results to an external service provider. These highly specialized tasks can only be performed by a handful of companies. When another service in the hospital is composed of operations provided by these external services, it becomes critically dependent on them. If, for some reason, one or more of those critical externally provided services would become unavailable, the hospital would suffer immediate consequences and patient’s lives may be endangered.

In addition to such considerations as outlined in the previous paragraph, service composition may also lead to security questions. For example, assume a situation in which an insurance company requests medical information from a patient’s records to assess the validity of a claim that he submitted. Such a request would be received by the hospital’s billing department. However, the billing department has no access to patient’s medical information. To be able to answer the insurance company, the billing department will forward the request to the medical records service, who will in turn send the requested information directly to the insurance company. Security procedures must be put in place to prevent the medical records service to send out information without there being a previous (valid) request by the billing department to do so.

Having outlined the potential problems that may arise as a consequence of adopting service composition, we do feel that it offers strong added value to the concept of service-oriented computing.

2.3.4 Web Services

Web services are the most common incarnation of the service-oriented computing concept. Many different notions of what a web service is are in existence. For example, in (Haas and Brown, 2004), the concept of a web service is defined by the World Wide Web Consortium (W3C) as

> ”. . . a software system designed to support interoperable machine-to-machine interaction over a network. It has an interface described in a machine-processable format (specifically WSDL). Other systems interact with the Web service in a manner prescribed by its description using SOAP-messages, typically conveyed using HTTP with an XML serialization in conjunction with other Web-related standards.”

This definition adopts a very technology-centered approach and restricts web services to those services that are described by a WSDL document. This technology-centered approach is in contrast with (Papazoglou and Georgakapoulos, 2003), who define web services as “a specific kind of service that is identified by a URI, whose service description and transport utilize open Internet standards”. This approach concentrates more on the interoperability aspect and on the standardization aspect of web services.

Vendor specific definitions, such as IBM’s definition of an architecture for web services:

> ”. . . A service-oriented architecture (SOA) is a component model that inter-relates the different functional units of an application, called services, through well-defined interfaces and contracts between these services.”[3]

This definition concentrates on a more generic definition of the service-oriented approach. However, without defining explicitly what a service is, the above definition is too broad. In contrast, Microsoft adopts a definition which states:

> ”XML Web services are the fundamental building blocks in the move to distributed computing on the Internet. Open standards and the focus on communication and collaboration among people and applications have created an environment where XML Web services are becoming the platform for application integration. Applications are constructed using multiple XML Web services from various sources that work together regardless of where they reside or how they were implemented.” (Wolter, 2001)

This definition not so much states what web services are, but focuses on the potential use of the services. For the remainder of this document, we will use the definition by Papazoglou et al.

2.3.5 The IBM and Microsoft Road-map for Web Services Security

The need for addressing security in web services has been acknowledged in an early stage by the IBM Corporation and the Microsoft Corporation. In April, 2002, the two software giants published a joint white-paper titled *Security in a Web Services World: A Proposed Architecture and Roadmap* (Web Services Security Roadmap, 2002).

The architecture that is presented in the article proposes a solution which places the entire security stack in the realm of SOAP headers. The *web services security specification* is graphically represented in Figure 2.6.

The specifications of which the road-map consists are:

1. *WS-Security*
   The WS-Security standard addresses the issue of attaching signature and encryption information to SOAP messages. For this, WS-Security relies on the

---

[3] `http://www-128.ibm.com/developerworks/webservices/newto/`

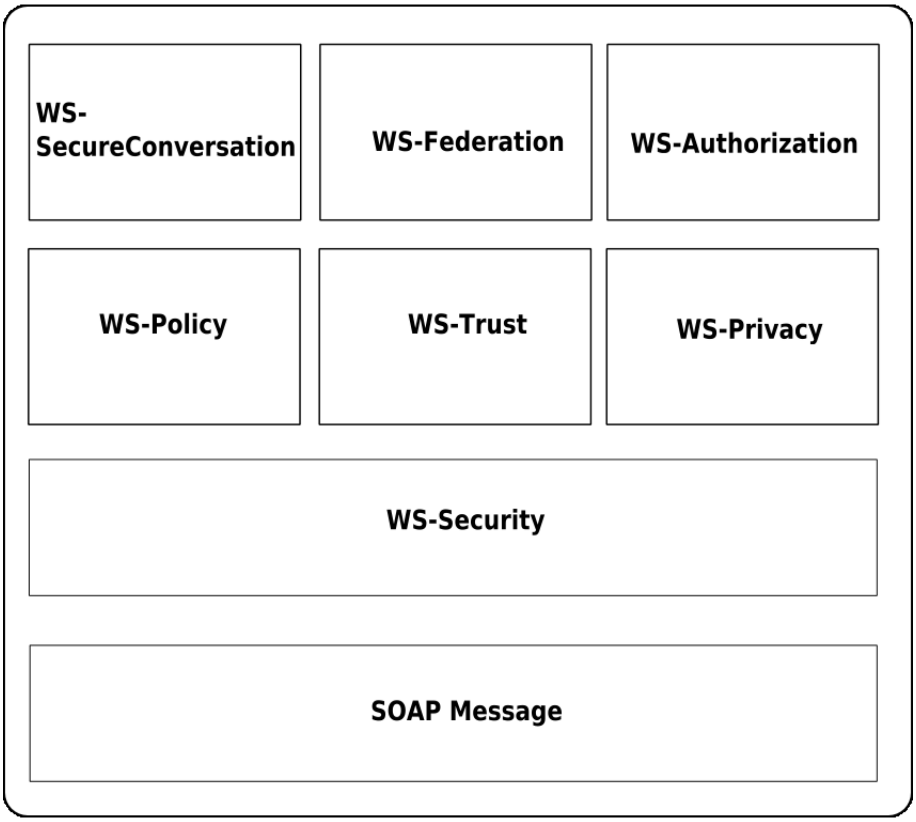

WS-SecureConversation
WS-Federation
WS-Authorization
WS-Policy
WS-Trust
WS-Privacy
WS-Security
SOAP Message

Figure 2.6: The IBM and Microsoft Security Specification for Web Services

XML Encryption (Reagle, 2002) and on the XML Signatures (Bartel et al., 2002) standard. WS-Security also includes a way to include security tokens, such as for example suggested by SAML (Farell et al., 2003).

2. *WS-Policy*
   WS-Policy (Bajaj and et al, 2004) is a generic standard to express things that a service should do (or should not do), and things that a service must do (or must not do). For example a WS-Policy statement could be used to specify that the SOAP message used that call the service is encrypted with a valid x509 certificate.

3. *WS-Trust*
   WS-Trust (Gudgin and et al, 2005) “uses the secure messaging mechanisms of WS-Security to define additional primitives and extensions for security token exchange to enable the issuance and dissemination of credentials within different trust domains.”

   In other words, WS-Trust provides a mechanism by which security information, such as identities, to be exchanged between different trust domains.

4. *WS-Privacy*
   The security road-map describes the WS-Privacy specification as follows: “This specification will describe a model for how a privacy language may be embedded into WS-Policy descriptions and how WS-Security may be used to associate privacy claims with a message. Finally, this specification will describe how WS-Trust mechanisms can be used to evaluate these privacy claims for both user preferences and organizational practice claims.”

   While security requirements often regulate *access control* to constrain which individuals have access to specific resources, the privacy domain assumes that information is available, but that its use and dissemination must be constrained.

   The WS-Privacy standard will allow web services to convey statements such as ‘by receiving his information, you agree not to disseminate it any further’. At the time of writing this document, WS-Privacy *does not enforce adherence* to privacy policies.

5. *WS-SecureConversation*
   The WS-SecureConversation standard (Della-Libera and et al, 2002) extends WS-Security and WS-Trust with the notion of a security context, and provides mechanisms for establishing and sharing them.

   According to the specification, “a security context is an abstract concept that refers to an established authentication state and negotiated key(s) that may have additional security-related properties”.

6. *WS-Federation*
   The WS-Federation standard (Bajaj et al., 2003) defines mechanisms that are

used to enable identity, account, attribute, authentication, and authorization federation across different trust realms.

In other words, WS-Trust provides a mechanism to exchange security tokens across trust domains. WS-Federation relies on WS-Trust to provide tokens that are used to exchange information about identities, security attributes, etc. within the federation.

7. *WS-Authorization*
   The WS-Authorization standard specifies how to manage and specify access control policies. WS-Authorization relies heavily on SAML.

At the most basic level of the architecture, the SOAP message can be found. This is also the weakest point of the architecture.

1. In theory, web services technology can be bound to any underlying technology. However, the road-map as presented by the two companies is strongly tied to using SOAP message for information exchange. This is illustrated by the fact that the security road-map is fully based on SOAP messages, and does not take any other message types into account.

2. The road-map completely bypasses the fact that security transcends message-level security. For example, access control requirements often include requirements which span that of a single message.

3. While the road-map identifies areas of interest, there is significant overlap between the different standards of which it is composed. For example, WS-Authorization and WS-Federation both make statements regarding security tokens, and the way that they should be managed.

4. Finally, the underlying conceptual framework on which the road-map is based appears to lack a common vocabulary and does not adopt a common processing model.

Furthermore, the architecture consists of eight components, of which at most three or four deserve the label ‘mature’.

2.3.6 SAML: Security Assertion Markup Language

SAML is a specification that defines a standard way to represent authentication, attribute and authorization information that may be used in a distributed environment by disparate applications (Hartman et al., 2003). The SAML specification defines the syntax and processing semantics of assertions made about a subject by a system entity (Farell et al., 2003). The specification defines both the structure of SAML assertions, and an associated set of protocols, in addition to processing rules involved in managing a SAML system.

When distributed systems, such as services, need to exchange security information they need to agree on a common representation of the information. The information must be adequately safeguarded so that its validity can be assured. SAML proposes to exchange security information in the form on security assertions. An assertion provides zero or more statements that are made by a *SAML authority*.

Assertions usually apply to a subject, such as the sender of a message. The SAML specification identifies three different types of assertions:

1. *Authentication assertions*, which state that the subject of the assertion was authenticated in a particular way at a particular time.
2. *Attribute assertions*, which provide additional information about the subject of the assertion. For example, the roles that a subject plays can be expressed in an attribute assertion.
3. *Authorization assertions*, which contain a request to allow subject of the assertion to perform a certain operation.

As mentioned, SAML also prescribes protocols for each assertion to follow. For example, access control decisions are implemented using a request/response protocol. A requester may send an authorization decision query which states “Should this operation be permitted for this subject, given this evidence?”. A response will come back in the form of a SAML assertion or a SAML encrypted response.

The SAML specification is a well-developed specification which is rapidly getting adopted by industry. Unfortunately, Microsoft Corporation has already announced that it will not support version 2.0 of the standard because it believes that WS-Federation is better suitable for its purposes.

2.3.7 XACML

XACML is an XML-based language for expressing access control policies. There is a strong relationship between SAML and XACML, as XACML started as a spin-off of the work in SAML. Furthermore, starting with SAML 2.0, an explicit note is made in the specification that SAML authorization assertions will not be developed further, and that the use of XACML is suggested.

XACML is designed to address the issue of security policy exchange in distributed environments, where there are multiple points of access control enforcement, yet limited points of access control decision making and administration.

XACML suggests an architecture as shown in Figure 2.7. The architecture exists of PEP’s (Policy Enforcement Points), PDP’s (Policy Decision Points), PAP’s (Policy Administration Points) and PIP’s (Policy Information Points).

Access control rules are maintained in administration points, where they may be grouped into policies and/or policy sets. When a request comes in, a PEP will collect information to be able to enforce an access control decision. To achieve this, a decision needs to be made at a PDP. A context handler may collect additional information from a PIP and send the information back to the PEP, which will enforce the decision.

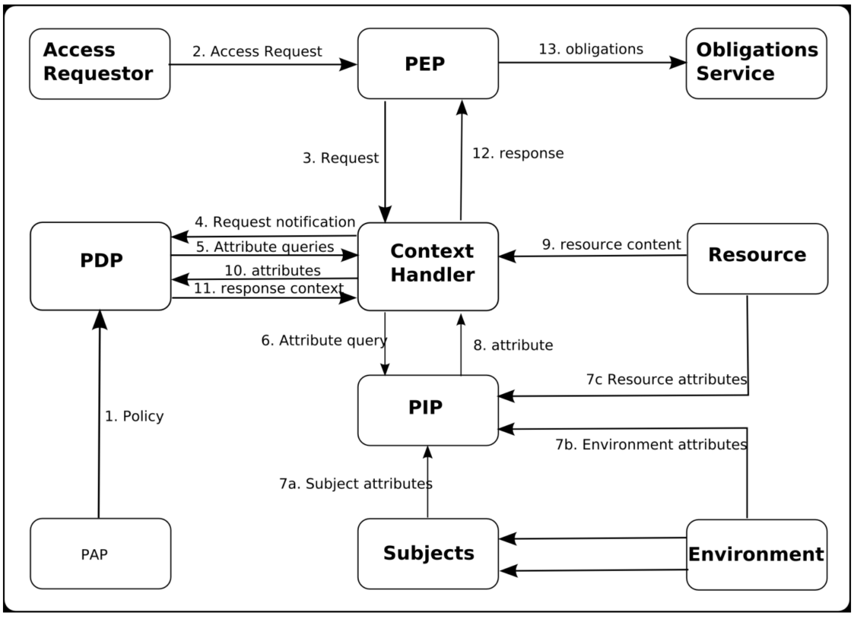

Access Requestor
2. Access Request
PEP
13. obligations
Obligations Service
3. Request
12. response
4. Request notification
5. Attribute queries
10. attributes
11. response context
PDP
Context Handler
9. resource content
Resource
6. Attribute query
8. attribute
7c Resource attributes
PIP
1. Policy
7b. Environment attributes
7a. Subject attributes
PAP
Subjects
Environment

Figure 2.7: XACML Dataflow Diagram (XACML, 2005)

XACML is best at place in an environment with many points of access control enforcement and/or decision making. In the context of web services invoking each other directly, XACML might be appropriate in a situation where one organization deploys many services that must adhere to the same policies.

2.3.8 Discussion

The basic service-oriented architecture that was introduced in Section 2.3 provides a minimal implementation of the service-oriented computing aspects. Most of the limitations, such as service composition and service management are addressed in the extended service-oriented architecture that was presented in Section 2.3.1.

While the ESOA provides a significant improvement over the basic SOA, it is still unclear how to position non-functional aspects such as security in that architecture. While there are security aspects in all layers (basic operations, composition and management), there is no explicit mention of it.

IBM and Microsoft developed a joint road-map for security in the web services SOA. The joint road-map is outlined in Section 2.3.5. However, despite many years of work, the security road-map is currently in an early stage of development; only the WS-Security and the WS-Policy standards have reached a stage of adolescence.

In addition, the security road-map is strongly tied to the web services SOA in that all security operations are implemented in basic SOAP headers. Finally, there is no common vocabulary or shared processing model for the components that make up the joint road-map.

We also discussed two initiatives that are related—to some extent—to the web service security road-map. SAML is a security markup language that can be used to exchange security assertions. SAML is used by the WS-Security standard to exchange security tokens. XACML is a generic language for XML-based exchange of access control policies, including a processing architecture, for environments in which there are many points of access control enforcement and only limited points of decision making and administration.

2.4 Event-driven processing

As technology continues to evolve at an accelerating rate, non-trivial computing systems will remain diverse and heterogeneous (Vinoski, 2002).

Message-Oriented Middleware (MOM) is a middleware concept that has been developed to address the need to integrate distributed and heterogeneous applications by the messages that they exchange.

MOM-based systems generally adopt an messaging paradigm which is asynchronous in nature (Tai et al., 2002; Maheshwari et al., 2004; Chappell, 2004). MOM provides an infrastructure that transmits messages and events to the widely spread components of a service, glueing them together in a logical coupling (Banavar et al., 1999).

Applications that are not designed to communicate with each other directly can still do so by sending and receiving their messages via a common layer of middleware. This allows application programmers to integrate their systems on the message level, rather than on the interface level.

Asynchronous message-based interactions are typical in an environment, like the SOA, where multiple highly distributed applications and services need to interact with each other, and which requires loosely coupled interfaces (van den Heuvel et al., 2005).

Asynchronous communication is often referred to as event-driven communication. A communications infrastructure based on the asynchronous model has the advantage of being loosely coupled and provides many-to-many communication. Furthermore, such a model is usually considered more scalable than the traditional synchronous model (Hsiao et al., 2003).

The event driven paradigm is ubiquitous in modern software (Hansen and Fossum, 2004). Event-driven systems can be found in a wide range of applications, such as sensor arrays, systems monitoring, trading systems on stock exchanges, etc. Event-driven programming is the dominant paradigm for designing and implementing applications that are driven by a graphical user interface. More recently, event-driven approaches are deployed more and more in Internet-enabled systems. For example, popular messaging software, such as Jabber, Yahoo!Messenger, AOL Instant Messenger and MSN Messenger all operate by exchanging events between members of the networks.

Event-driven interactions are characterized by a continuous flow of messages representing ‘things that happen’. Events are commonly represented by structured messages which are exchanged between the event generator and any number of event receivers, and carry contextual information in which the event is described, plus the circumstances in which it occurred. Such contextual information may be an explicit part of the event representation, or can be implicitly deduced from its circumstances.

The focus on message exchange is precisely that what characterizes event-driven systems. In an event-driven approach, participants in a conversation interact by exchanging events which flow dynamically between sender and receiver. Each event causes a receiver to respond to it and generate events in turn, resulting in a *global event cloud* (Luckham, 2002).

Most event-driven approaches provide stateless and asynchronous message passing. This requires an application programmer to decouple the act of generating an event from the act of reacting to one, even if the event generation should cause a near-direct reply. Adopting an asynchronous approach facilitates loose couplings, and is often seen as a solution to achieve reliability and performance (Brambilla et al., 2004).

In (Ceri et al., 1997) events are defined as primitive operations [monitored by active rules]. At the heart of this definition is the notion that events are elementary, non-dividable operations which may be observed. (Morgan, 2002) views an event as a ‘happening’ that causes, or is caused by, an ‘impact’ on a business process (Morgan, 2002). The notion that events are atomic is disputed by (Luckham, 2002), who considers an event as an object that is a record of an activity in a system.

(Luckham, 2002) distinguishes three aspects of an event. The first aspect is its *form*. An event is typically associated with data structure which describes information about the context in which it occurred. The second aspect is the *significance* of an event. An event signifies an activity in the real world. The third aspect of an event is its *relativity*, which represents the relationship in time, causality, or aggregation of one event to the other.

2.4.1 Properties of event-driven processing

Hansen and Fossum identify a number of characteristics of the relationship between the sender and the receivers of an event.

1. *Runtime registration*
   The decision who will receive an event is not taken until the event is generated. This is different from traditional programming, in which each operation (often in the form of a method on an object, or an procedure in a library) is called directly and by name.

2. *Multicasting*
   An single event may have multiple recipients.

3. *Multiplexing*
   A single event recipient may receive events originating from a large number of senders.

4. *Inverted semantics*
   In procedure-oriented design, programs are often implemented in units that

> have a client/server relationship. Client procedures achieve their higher-level goals in part by calling a collection of procedures that provide lower-level services. In many event-driven programs, a low-level event source triggers a higher-level action.

The run-time binding characteristic is a crucial technique to implement loose couplings between distributed systems.

The multicasting characteristic is typical for event-driven systems. In traditional programming, a message that is exchanged between systems often takes the form of some form of data that is sent via a network socket, or via a remote procedure call. In all cases, each message will have only one recipient. When a message must reach more than one destination, it must be sent multiple times. Event-driven programming provides the ability to multicast messages.

The inverted semantics characteristic is, in my opinion, the weakest of the three. Whether a program is 'lower-level' or 'higher-level' is often irrelevant, or in the eye of the beholder.

Based on the characteristics outlined above, we propose the following set of characterized of event-driven interactions:

1. *The synchronicity property*
   Message exchange can be either synchronous or asynchronous in nature. In a synchronous message exchange pattern, the sender of a message must wait until a response is received from the message receiver. In an asynchronous message exchange pattern, the sender continues processing after a message has been sent. The response of a message, if any, will come back as another asynchronous message. Valid values for this property are *synchronous* and *asynchronous*.

2. *The receiver cardinality property*
   The receiver cardinality property captures the ability of a single message to be sent to any number of receivers. For example, assume a message exchange pattern with a receiver multiplicity of one. If a sender attempts to notify three receivers of the same event, three separate messages must be sent. Valid values for this property are any discrete number, starting zero and counting upwards. The special value "0" signifies that there is no upper limit to the number of simultaneous receivers of a single message.

3. *The relativity property*
   The relativity property is a multi-valued property which represents the way in which single messages can be related to each other. Note that at a first glance this property is only applicable to synchronous message patterns. However, this is not the case, as there can still be a logical relationship between messages that are sent asynchronously. Valid values of this property can be no value at all, representing that it is not possible to relate individual messages to each other, or any combination of *causal* and *chronological*. A causal relationship is one in which it can be determined which event caused another

message to happen, and a chronological relationship between two events signifies that it can be determined in which chronological order two events have been sent. Note that all causal relationships are by definition also chronological.

4. *The temporal property*
   The temporal property of messages applies to the time-to-live of a message. It is often desirable to discard a message after a certain time has passed. Valid values of this property no value at all, to signify that messages never expire, or *time-to-live*, which specifies how long a message is valid for after it has been sent, or *expiration* which specifies a hard date/time combination after which a message is no longer valid.

5. *The cardinality property*
   The cardinality property constraints the number of times a message can be resent. For example, in a situation where a message may pass a number of hops this property can be used to restrict the logical distance it can travel. Valid values for this property are -1, which represents that there is no boundary, 0 which means that the message may never be sent, or any positive discrete number which represents the number of hops the message may travel at the most.

2.4.2 Enterprise Service Bus

The Enterprise Service Bus (ESB) concept is a new approach to integration that can provide the underpinnings for a loosely coupled, highly distributed integration network that can scale beyond the limits of a hub-and-spoke enterprise application integration broker (Chappell, 2004).

The enterprise service bus concepts advocates the use of standards for integration wherever it can. Especially when *open* standards are adopted, critical dependencies on individual vendors are reduced. As a result of adopting open standards, a more homogeneous integration environment is realized.

In an event-driven enterprise, business events that affect the normal course of a business process can occur in any order and at any time (Chappell, 2004).

An Enterprise Service Bus can provide an implementation backbone for an SOA. It establishes proper control of messaging as well as applies the needs of security, policy, reliability and accounting in an SOA architecture (Papazoglou and van den Heuvel, 2006).

Unfortunately, while the ESB concept provided an good opportunity to include security in its architecture, this has not happened to an adequate extent. For example, Chappell writes about security:

> The connections between nodes on the ESB are firewall-capable. The security between the ESB, and even between the ESB nodes themselves, is capable of establishing and maintaining the most stringent authentication, credential management, and access control.

However, how this is achieved, or even where in the architecture this security measures manifest themselves is not expressed. The lack of security in the architecture is also acknowledged by Papazoglou and Van den Heuvel, who write

> The ESB needs to both provide a security model to service consumers and integrate with the (potentially varied) security models of service providers. Both point-to-point (e.g., SSL encryption) and end-to-end security capabilities will be required.

For the ESB concept to gain widespread popularity, the issue of security must be addressed at the architectural level, and not left to the various vendor's implementations of ESB's. Doing so would result in a large variety of ESB's which are so heterogeneous in nature that the benefits that were gained by using it would be mostly mitigated.

2.4.3 Event-driven interaction patterns

Event-driven interactions are often characterized by asynchronous and stateless interactions. However, to have a meaningful conversation, it is necessary to be able to relate events to each other. This is illustrated by a simple example in which some actor generates an event which represents the need for some action to be taken. It is very useful if somebody else is able to respond to that event to indicate their intention to do so. To accomplish this, it must be possible to relate both events to each other. The way in which parties interact can be caught in interaction patterns, such as described the following sections.

Publish-subscribe

The most common event-driven interaction pattern is the publish-subscribe pattern. In the publish-subscribe pattern, one party publishes a certain type of event to announce that he is going to generate them at a later time. Another party may be interested in being notified whenever such an event is generated, and decided to subscribe to this. Whether or not event publications and subscriptions are facilitated by one or more event brokers, or take place in a peer-to-peer environment is not directly relevant for this discussion.

In other terms, producers publish information on a software bus (an event manager) and consumers subscribe to the information they want to receive from that bus. This information is typically denoted by the term *event* and the act of delivering it by the term *notification* (Eugster et al., 2003).

In an unbrokered publish-subscribe interaction pattern, full control of publishing events, generating them and sending them to the final recipients resides with the subject that generates the event. In case of brokered publish-subscribe interactions, that control resides with the event broker.

Figure 2.8 illustrates brokered publish-subscribe. On the left-hand side, event publishers announce their presence by publishing events, and generating them at a later time. The event broker manages the subscriptions of those events and notifies

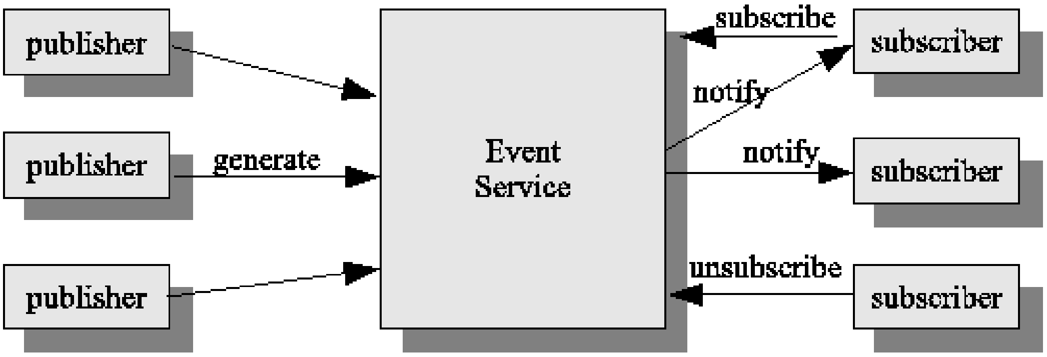


Figure 2.8: Brokered publish-subscribe interaction

the appropriate subscribers when they are generated. Illustrated by this figure is also the fact that publishers and subscribers are only indirectly coupled to each other, and at any time can modify their subscriptions. This provides a high level of flexibility and scalability which is critical for large-scale enterprise solutions.

Polling

In the polling interaction pattern, the sender of an event continually asks the event receivers for results. Polling is generally non-blocking, which means that the event sender may continue to process other tasks while waiting for an answer.

Polling assumes that a sender is aware of who the events are routed to. In a polling interaction pattern, control of the conversation resides with the receiver of the event, since he is able to withhold an answer at his discretion.

Fire and Forget

The most simple interaction pattern is the fire-and-forget pattern. Event senders who adopts this pattern do not expect any results and are content with simply sending off an event.

This pattern is characterized by the total lack of interaction between sender and receiver of the event.

2.4.4 Event message filtering

In the case of request-response interactions, subscribers will be notified of all events to which they are subscribed. Often, this is undesirable, which introduces the need for an additional mechanism that can be used to further restrict the events that a subscribers must process. Common approaches to achieve notification filtering are topic-based filtering, content-based filtering and subject-based filtering. Each of these approaches will be discussed in more detail in the following sections.

Subject-based filtering

Subject-based filtering is the most basic notification filtering mechanism. In addition to subscribing to events, an event consumer also explicitly specifies a set of event generators from which the event must originate. All events that do not originate from those subjects will not be delivered to him.

Subject-based notification has as a disadvantage that event generators and event consumers must be aware of each others existence, which increases the tightness by which they are coupled. In some situations, such as high-security approaches, this may be desirable.

A variation on subject-based notification filtering is role-based notification filtering. Rather than accepting events that are generated by a predefined set of subjects, events are filtered based on the roles that subjects play. By adopting this approach, subjects do not have to be aware of each other's identity, rather they need to know which roles subjects can play. Role-based access control, which is discussed in Section 2.2.8 is based on the same indirection between permissions, roles and users.

Subject-based notification filtering is most commonly deployed by event consumers and serves the purpose of reducing the amount of events that must be processed. However, subject-based filtering can also be deployed by the event producers. The same consideration as mentioned before—sacrificing the loose couplings—applies to producer-side subject-based filtering, however controlling who will receive an event after it is generated may be a requirement in high-security scenarios.

Topic-based filtering

Topic-based notification filtering is a mechanism in which event producers explicitly associate one or more topics with each event. Event subscribers subsequently subscribe to these topics, rather than to the events themselves. Using topic-based notification filtering, event consumers do not need to posses a priori knowledge about the exact events that they will receive, but can suffice with knowing which topics are available.

Topics are commonly organized in trees, in which each subtopic is more specific than the next. This allows event subscribers to either very accurately indicate in what kind of topics they are interested in receiving, or it will allow them greater flexibility in subscribing to a wider range of topics. Subscriptions can span multiple topic trees, allowing for even greater flexibility and accuracy in specifying subscriptions.

The principal problem in deploying topic trees for notification filtering is that all parties who produce and consume events must assign similar semantics to the topics. This is a highly complicated field, which is currently explored by ontology researchers all over the world. A second problem with using topic trees is the ownership of each tree. Questions, such as “Who owns the topics?” and “Under which circumstances can topics be added, removed or modified?” must be clearly answered before topic-based notification filtering can be successfully adopted.

Topic-based notification filtering provides a level of indirection between the event producer or event consumer and the event itself, in the sense that subscriptions to events are associated with one or more topics, and that each topic has one or more subscribers. As such, topic-based notification filtering provides a mechanism for loose coupling of distributed event processors.

Content-based filtering

Content-based notification filtering allows subscriptions to evaluate the whole content of notifications, and so it provides a more powerful and flexible notification mechanism than topic-based or subject-based mechanisms (based on (Mühl, 2002)).

Content-based filtering and routing implies that event consumers somehow specify a set of criteria to which they require events to adhere. Event producers are free to generate any kind of event, since the routing mechanism will determine where to deliver the messages.

In his thesis, Mühl describes a number of algorithms which can be used for content-based routing and evaluates them to find an optimal configuration.

Content-based notification filtering provides a thorough indirection between event producers and event consumers, which makes it usable in large scale loosely-coupled distributed systems.

2.5 Discussion

In the remainder of this chapter, we will take a closer look at the security implications of service compositions, followed by a discussion on trust and a discussion on delegation. We end the chapter with some observations on the security requirements for service-oriented computing, and design objectives for an event-driven secure SOA.

2.5.1 Trust

A principal problem in identity-based system is that of trust. Consider that the admission service sends a ‘patient admitted’ event as a result of previously receiving an admission event. The only way to detect that there is a causal relationship between those two events, is that the admission service explicitly makes known that this relationship exists. When it would omit that fact, there is no way of identifying the causal nature of the relationship. Rather, one would be limited to the observation that the admission event was sent earlier than the patient admitted event.

In another example, suppose that the travel service sends e_2, and includes the claim that it was sent as a result of a previously received event e_0. Furthermore, assume that the travel service never actually received e_0.

In both situations, it can be detected where or not an event e_0 has been relayed to the travel service before event e_2 is sent out, but the conclusion that those two events must have a causal relationship cannot be drawn.

This is illustrated furthermore when multiple e_0 events are sent (say, $e_{0.1}$ and $e_{0.2}$ before e_2's are generated. For reasons of its own, the travel service may decide that the second request that it received ($e_{0.2}$) will be processed before the first. If the causal relationship would be automatically deduced, chances are that the first e_2 is associated with the first e_0, whereas it should be associated with the second one.

The only true solution is not technical in nature. The trust relationship that must exist between EFSOC and the participating services must be established and maintained out-of-band. Only if services can be trusted when they claim that an event is sent as a result of a previously received event, than access control rules that surpass IP based network filtering rules can exist.

Because EFSOC and the participating services are involved in a trust relationship, trust relationships between services can be deduced. For example, when the Travel Service has established a trust relationship with EFSOC, and the Car Rental Service also has a trust relationship with EFSOC, some degree of trust may be deduced between the Travel Service and the Car Rental Service.

2.5.2 Delegation

One of the most fundamental requirements of EFSOC is that each subject is in full control of the way it is accessed. In other words, if a subject wishes to limit access to its resources, it should be able to (partially) communicate that desire, so it can be enforced. As governed by the principle of least privilege, subjects should only have the access that they require to fulfill their tasks and nothing more. Consequently, from a security management point of view, the amount of authorizations handed out to subjects should be limited as much as possible.

However, autonomy and least privilege may be at odds with each other. If subjects are able to delegate privileges to other subjects without restriction or coordination, managing those delegations becomes an additional problem that may have security implications. While in a centrally governed access control system, there is only one point of control, in a decentrally governed access control system, there are many.

Role-based access control is an access control model which may provide a solution for this problem. Compared to discretionary access control, which leads to a massive administrative overhead and an opaque collection of authorizations and authorization requirements, role-based access control introduces the role as an indirection between subjects and privileges. Role-based access control is successful because the relationship between roles and permissions turns out to be relatively stable over time, while the relationship between roles and users may remain volatile.

While introducing an subject-enacted delegation mechanism to service-oriented computing will provide a convenient level of flexibility, it introduces the same management problems that make discretionary access control unsuitable for large-scale operations.

This observation would suggest that delegation is a bad idea in the context of service-oriented computing, and, as a matter of fact, I believe that unlimited delegation should be avoided at all times. However, the additional level of flexibility that is gained by allowing delegations is tempting and possibly even necessary to be available in SOC scenarios. Consequently, I propose that delegations will be allowed, provided

1. delegations are temporary;
2. delegations are non-transferable;

The requirement for delegations to be temporary is fueled by the need to keep the amount of delegations manageable. Temporary delegations will keep the total amount of delegations relatively low which makes them transparent. As a result temporary delegations will lead to a more manageable situation. In addition to this practical consideration, there is also a more fundamental one: if delegations are not temporal (i.e., permanent), then why is that not reflected in a corresponding access control rule? The answer to that question is that most of the time, updating access control rules takes longer and involves more effort than delegating a privilege.

Secondly, delegations must be non-transferable. In other words, if a service delegates certain privileges to subject, that subject should not be able to re-delegate those permissions. This requirement originates from the autonomy principle, which states that a subject must be in *full* control of the way it is accessed. If delegations can be re-delegated, the level of control that a service has is removed by one degree, and can no longer be considered to be full.

Barka’s Framework for Role-Based Delegation Models

Barka proposes a comprehensive framework for role delegations in (Barka, 2002). The author distinguishes a number of characteristics which he postulates are capable of fully describing a role delegation framework. The properties are:

1. *Permanence (temporal/permanent)*. The permanence characteristic determines if a delegation remains valid indefinitely, or that it has a life time associated with it.
2. *Monotonicity (monotonic/non-monotonic)*. The monotonicity characteristic determines if by delegating the permission, the delegator maintains the delegated privileges (monotonic), or that by delegating it, the delegator forfeits his privileges (non-monotonic).
3. *Totality (total/partial)*. Barka’s framework refers to role delegations. The totality characteristic determines if it is possible to delegate the entire role, including all of its privileges (total), or that it is possible to delegate some of the privileges that come with a role (partial).
4. *Administration (self-enacted/agent-enacted)*. The administration characteristic determines who initiates and executes a delegation. If a subject is able to actively delegate privileges himself, the model is considered to be self-enacted. If subjects are able to nominate a third party to execute the delegation, the model is considered to be agent-enacted.

5. *Levels of delegation (single step/multi-step).* This characteristic determines whether a delegation can be re-delegated and if so, how often.

6. *Multiple delegation.* Using this characteristic, subjects can limit the amount of times a delegation can be given.

7. *Agreement (bilateral/unilateral).* This characteristic describes whether a delegator and a delegatee must agree on the act of delegation (bilateral), or that the delegator can force a delegation onto another subject (unilateral).

8. *Revocation.* Barka describes a number of delegation revocation properties.

The subject autonomy property required that a subject is in full control of the way it is accessed. Therefore, the only one who can regulate to a subject s is the subject s itself. As a result, s is the only one who may *delegate* privileges. Adopting this line of reasoning, delegations in EFSOC have to be non-monotonic, single step, partial and self-enacted. Furthermore, delegations are the only way that subjects are able to grant specific rights to other subjects. For example, take an example in which a subject wishes to restrict access to one of its operations. It is able to communicate this desire to EFSOC by giving out a *delegation entitlement*.

2.6 Conclusions

To develop a SOA which includes access control requirements from the beginning, rather than as an afterthought, we identified a number of design objectives. To realize these objectives, a number of requirements is elicited and discussed in Section 2.6.2.

2.6.1 Objectives for a secure SOA

We identify the following design objectives:

1. *Service Autonomy*
   Services are autonomous and must remain so. The power of service-oriented computing lies in their loosely coupled nature. Such loose couplings will be lost when services have to give up their autonomy. As a result, by remaining autonomous and by keeping the couplings between services as loosely as possible, services may react to changes while minimizing programmer's direct interventions (Paolucci and Sycara, 2003). Therefore, services must be in full control of their own security policies.

2. *Containment*
   Containment is a property that is barely mentioned in literature. Yet, we feel it is of the utmost importance. Containment addresses limiting the extent to which an intruder can affect services after one or more security mechanisms have failed. Containment can be easily compared to fire doors in the physical

world. Fire doors serve the purpose of slowing down a fire in one part of a building, hopefully preventing it from spreading further. In the world of service-oriented computing, containment means that when one service gets compromised, the damage must stay limited to that one service and not spread to others. As such, containment is very closely related to the availability objective.

3. *Separation of Duty*
   While traditional access control generally focuses on technical measures to prevent unauthorized access to resources, separation of duty attempts to achieve the same objective by deploying organizational measures. While separation of duty can be (partially) enforced by access control technology, the measure itself is firmly grounded in the business domain. Separation of duties means that certain services can only be provided when a number of subjects (each playing separate roles) collaborate. The underlying assumption is that by separating the roles that subjects may play, separation of duty may be achieved relatively easily.

   The literate on role-based access control often distinguishes two types of separation of duty. Static separation of duty prevents two roles from every being *assigned* to the same subject, while dynamic separation of duty allows two separate roles to be assigned to the same subject, but that subject cannot have the roles active at the same time. This objective closely ties in to the well-known concepts of *least privilege* and *active security* (Bacon et al., 2002).

4. *Availability, Integrity and Confidentiality*
   All literature that addresses information systems security in any form must include a discussion on the so-called A-I-C triad (Hansche et al., 2004). The A-I-C triad encompasses Availability, Integrity and Confidentiality. When the A-I-C triad is positioned in the context of service-oriented computing, the following meaning can be given to each of its components. *Availability* measures provide the assurance that services are available for its clients whenever they need to be. *Integrity* of services affects the extent to which the messages that are exchanged as part of a service's input and output are altered after they are sent; either intentionally or unintentionally. The goal that is pursued by integrity is that once a message is written, it cannot be changed after the fact without noticing that it did.

   Confidentiality is closely related to Integrity, as it attempts to hide the contents of messages from everybody except the intended recipient. Integrity and confidentiality are often achieved by deploying cryptographic techniques.

5. *Auditability*
   A first-order business requirement is auditability. Any business operation must be traceable in history to its originator, and all parties involved in the operation. Auditability can be used to ensure *non-repudiation*.

6. *Message-context level access control*
   EFSOC should transcend traditional access control approaches, such as DAC, MAC and RBAC, and provide access control on the level of message context.

7. *Authentication*
   Authentication is the process by which the identity of service providers and service invokers is established. In role-based approaches, authentication is also used to establish the roles that can be played by a given subject. Authentication is a necessary input for access control, as it is implied in its definition.

While the design objectives listed in this section are ambitious, we feel that they are realistic and necessary.

2.6.2 Requirements for Access Control and Service-Oriented Architectures

In the preceding sections, we discussed service-oriented computing and access control as two separate, yet related research fields. The previous section showed a listing of basic requirements for a secure SOA. In this section, we take a closer look at the access control aspects of these design objectives.

1. *Decentralized administration of access control policies*
   The service autonomy principle requires that each service is in full control of all messages it sends out, or receives. To require that decentralized administration of its access control policies is supported a logical consequence from that principle.

   As a result, each service should be able to specify access control policies that are able to constrain what happens to the messages that it sends out, as well as specifying what kind of messages it is willing to receive (and the conditions under which this is allowed to happen).

   Current approaches to distributed access control are often based on the premise of a centrally administered access control policy, which is subsequently distributed to decentrally located nodes. Each of the nodes is responsible for interpreting the access control policy and enforcing it.

2. *Active security*
   Service-oriented computing is expected to thrive in highly dynamical environments in which services may be created, modified, composed or remove on an ad-hoc basis. Active security will ensure that, even if circumstances change rapidly and often, the access control system is able to keep up with the pace.

3. *Reliable audit trail*
   For everything that takes place, a reliable audit trail must be established. It must be possible to find out which service sent which message, at which time, and to which recipients.

Most approaches to information security are fragmented; separate attention is paid to access control, authentication, accounting, etc. EFSOC provides an approach that unifies access control and accounting.

4. *Secure transport layer*
   With the current state of technology, there is no reason why services (esp. web services) should not use a secure transport layer. For example, web services typically run in an engine that is part of a web server, which is capable of using SSL certificates. Not only provides using an SSL certificate a way for services to authenticate themselves to each other, it also provides a mechanism to securely exchange (public) key information so that the transport layers can be encrypted and digitally signed.

5. *Secure message layer*
   The same reasoning as for the secure transport layer applies to the message layer. With wide spread availability of cryptographic software, there is no reason why messages that are transport are not also digitally encrypted and or signed. Examples of enabling technologies for achieving message layer security are WS-Signatures, XML Encryption and XML Signatures, or PGP.

2.6.3 State of the Art in Research

Security and access control has been in the spotlight of academic research for a significant portion of time. While service-oriented architectures is a relatively new field, several authors have expressed their opinions about security and access control in academic fora.

Altunay (2005) states that workflow engines should have decentralized access control models that leave the final access decision to each workflow participating entity. Moreover, she states that the workflow engine should not assume any knowledge about the internal security policies of each workflow participating entity. This corrobates the service autonomy property. In the EFSOC framework, services will able to specify their own access control policies.

Steele and Tao (2005) observe that RBAC is not sufficient for deployment in highly dynamic service-oriented architectures. The authors attribute that observation to the fact that “most RBAC implementations rely on the manual setup of pre-defined user-ID and password combinations to identify the particular user”. The solution presented assumes a unification of authentication and authorization. However, we do not feel that their solution provides many benefits over existing approaches. Furthermore, the solution provided by the authors lacks a common reference model. Finally, we feel that several commonly accepted separations of duty are violated. For that reason, we do not adopt the unification of authentication and access control. Instead, we provide an additional level of indirection, in which permissions will be mapped to roles using access control rules, rather than using static assignments.

Deubler, Grünbauer, Jürjens and Wimmel (2004) describe an approach for sound development of secure service-based systems. The approach concentrates on security requirements in the service modeling phase, in particular on enforcing access control resp. authorization. The approach assumes that service-interactions can be modeled before they are used, and that access control requirements are fully known in advance. As EFSOC provides a dynamic platform for changing service interactions, we believe that modeling the whole interaction ahead of time will be too time consuming. Rather, we provide an access control mechanism that is based on rules that are evaluated.

Interestingly, Bertino, Damiani and Momini (2004) describe an access control model that is specific for a set of services used to manage spatial data on the web. Rather than choosing a role-based model, the authors chose a discretionary approach. However, the extension of the authorization model to also encompass efforts gained by role-based access control are left as an open question. The approach followed in EFSOC combines a role-based approach with a discretionary approach.

Geihs, Kalcklösch and Grode (2003) present a single sign-on in service-oriented computing. In their paper, they introduce three different kinds of single sign-on (SSO): SSO based on a not commonly known secret, SSO based on a token and SSO based on biometrics. SSO based on biometrics is principally a special case of the more generic SSO based on a token.

SSO in SOA assumes that identities are known (and shared between services). While we acknowledge the need for a good authentication mechanism, we assume it to be available.

2.6.4 Summary

In the previous sections, we have provided the reader with background information on the fields of service-oriented computing, services security and event-driven interactions. In Section 2.3, we introduced the SOC concept. In it, we paid special attention to the IBM and Microsoft joint road-map for web service security, and concluded that it lacks in coherence and maturity. We also discussed SAML and XACML, and concluded that both initiatives have useful elements. In particular, the ability of SAML to express security attributes and XACML architecture in which enforcement, decision making and administration of access control is decoupled.

Section 2.2 addressed service security. We decomposed the security domain into authentication, authorization/access control, confidentiality, integrity and auditing. Each of the sub domains was discussed briefly. More attention was paid to access control, since that is where the focus of this research is. We discussed several access control models (discretionary access control, mandatory access control and role-based access control) and outlined their disadvantages and their benefits in service-oriented computing.

Event-driven interactions were discussed in Section 2.4. Specific care was given to introduce a number of interaction patterns, and in introducing a set of properties of event-driven interactions.

The aspects discussed in the previous culminate in Section 2.6, which outlines a number of generic objectives for a secure SOA, and in Section 2.6.2 which lists a number of specific security requirements. These requirements are: decentralized administration of access control policies, active security, establishment of a reliable audit trail, use of a secure transport layer and of a secure message layer.

The following chapter introduces EFSOC, the Event-Drive Framework for Service-Oriented Computing. The EFSOC framework is a SOA that realizes all requirements outlined that were listed in this chapter.

Chapter 3

The EFSOC Service-Oriented Architecture

> “Qui si convenien lasciare ogni sospetto;
> ogi viltà convenien che qui sia morta.
> Noi siam venuti al loco ov’io t’ho detto
> che tu vedrai le genti dolorose
> c’hanno perduto il Ben dell’ intelletto.”
>
> *Dante Alighieri, The Divine Comedy, Vol. 1 (Inferno III, The Gate and Vestibule of Hell). (1321)*[1]

3.1 Introduction

In this section, we present the EFSOC Service-Oriented Architecture. The framework is based on the observations that were discussed in the previous chapters and aims to provide access control for an event-based service-oriented architecture.

This research drops the premise that services are required to invoke each other directly. Instead, we assume that services publish their interfaces to a *service broker*, similar to web services publishing their WSDL document to a UDDI repository. Other services may discover these interfaces via that same service broker, but rather than invoking the newly discovered services directly, service consumers notify the broker of their requests. It will then handle the actual invocation of the appropriate service, and relay the results (if any) back to the originator of the request.

Adopting this approach facilitates the service broker (or the service brokers, in case of a distributed solution) to establish an certifiable *audit trail* in a single location, which can be used to settle differences of opinion which may arise in the future.

Furthermore, by routing the service interactions through a broker, it becomes possible to specify and enforce access control policies unambiguously and at a sin-

---

[1] “All fearfulness must here be left behind; all forms of cowardice must here be dead. We’ve reached the place where, as I said to thee, thou ’lt see that the sad folk who have lost the Good which is the object of the intellect.”

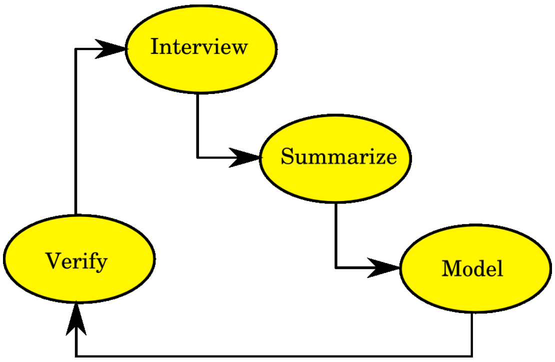


Figure 3.1: Case study elicitation process

gle point. An event-driven approach with distributed service brokers provides a flexible mechanism for ensuring increased availability of services, the ability to implement competing services and provides a reliable infrastructure for service interactions.

In the last part of this chapter, we show how the EFSOC Service-Oriented Architecture can be implemented as a value-added layer on top of existing web services solutions.

3.2 Case study

Throughout this chapter, we will be using the case study that was performed at Northside Hospital as running example.

In a number of interviews with domain specialists, we identified the information flows and the actions that are taken when a medical doctor concludes that a patient needs to have surgery.

3.2.1 Elicitation Process

A case study is a form of qualitative research. In a case study, a particular individual, program, or event is studied in depth for a defined period of time. In the case-study that is presented in this section, we studied the information flows regarding a hospital patient.

The information elicitation process used for this case study consisted of a number of iterations of phone interview, transcript, modeling in UML, as shown in Figure 3.1. The resulting models are verified by a next interview.

The telephone interviews typically took between one and two hours, and involved a number of information technology specialists that work for the hospital. Using a free discussion format, guided by several predetermined questions that needed to be answered, each discussion was recorded to a computer hard drive. After the interviews, the interviews were summarized in a running text.

The objective of the case study was to elicit a description of the activities that take place from the point that a patient visit a physician, continuing through medical testing, and finally resulting in surgery, including a description of the actors involved, and the information that was exchanged.

We were not interested in the exact form and content of the data that was exchanged. Instead, we looked at the function that the data played in progressing the overall process.

3.2.2 The HIPAA Privacy Rule

The way that medical information that can be linked to individuals is handled is subject to strict regulations in the United States of America. The 'Standards for Privacy of Individually Identifiable Health Information' is a set of standards for the protection of certain health information. This so-called 'Privacy Rule' implements requirements of the U.S. department of Health and Human Services Insurance Portability and Accountability Act of 1996. A major goal of the Privacy Rule is that individuals' health information is properly protected while allowing the flow of health information needed to provide and promote high quality health care and to protect the public's health and well being (HIPAA Privacy Rule, 2003).

The basic principle of the rule is that protected information (i.e., health information) may not be used or disclosed, except when the Privacy Rule explicitly requires or permits it, or when the individual to which this information applies formally authorizes in writing.

The privacy rule *requires* disclosure in only two situations: (a) to individuals when they require access to of the information that applies to them, or (b) the the department of Human and Health Services when it is undertaking a compliance investigation or review or enforcement action.

Furthermore, disclosure is *permitted* without prior authorization for treatment, payment and health care operations, public interest or when the data is filtered and anonymized for the purpose of research.

The HIPAA privacy rule affects operations of a medical institution at its administrative core. Especially the access control requirements that limit the kind of information that is permitted to flow between departments, and even persons, working for an institution impact all layers of operations.

3.2.3 Northside Hospital

Northside Hospital is a full-service community hospital that was opened in July 1970. Northside is a not-for-profit organization that has 455 beds (with growth over the next year to 539 beds) and over 1,800 physicians on staff. In addition, Northside has a staff of over 6,000 employees, which facilitate delivery of quality patient care to the more than 450,000 patients annually.

3.2.4 Hospital Policies

Before the process is described, a number of hospital policies is discussed.

1. Physicians, physician assistants and nurse practitioners at physician's offices at Northside Hospital are not employees of the hospital, but are private, independent practitioners qualified to hold privileges to practice at Northside Hospital.

   Northside Hospital does employ its own registered nurses and nurse practitioners.

2. Only clinical staff will have access to medical records.

3. Physicians will have access to their own patient records, as well as the records of their partners.

4. Physicians *can* access patient records of patients that are not treated by them, or their partners. However, HIPAA regulations prohibit using or disclosing health information if it not required for treatment.

   Consequently, the Medical Records Department audits all such accesses and determines if there was a medical need to do so. If there was no medical need, disciplinary actions will be taken against the physician in question.

5. Nursing staff does not have the ability to look up any medical information, other than information about patients that are currently in the hospital, on the floor and in the section that they work in.

6. Physicians are able to access most medical information from outside the hospital.

7. Physician staff is able to access limited medical information.

In this case study, we captured the information processes surrounding surgery patients. It starts when a physician determines that a patients needs to undergo surgery. The case study is illustrated with an activity diagram in Figure 3.2.

1. *Processing patient information*
   If, during the course of a treatment, a physician decides that surgery is in order, the physician's office will determine a date by requesting a reservation for an operating room from the surgery scheduling department.

   After the surgery is scheduled, scheduling information is provided to 'patient access'. Patient access contacts patients to collect pre-admission information, which involves capturing *demographic information*, which includes insurance information, billing information, etc. If the patient is already known to the hospital, the information may be available already, and this action is limited to validating the information that is already there, and extending it where necessary.

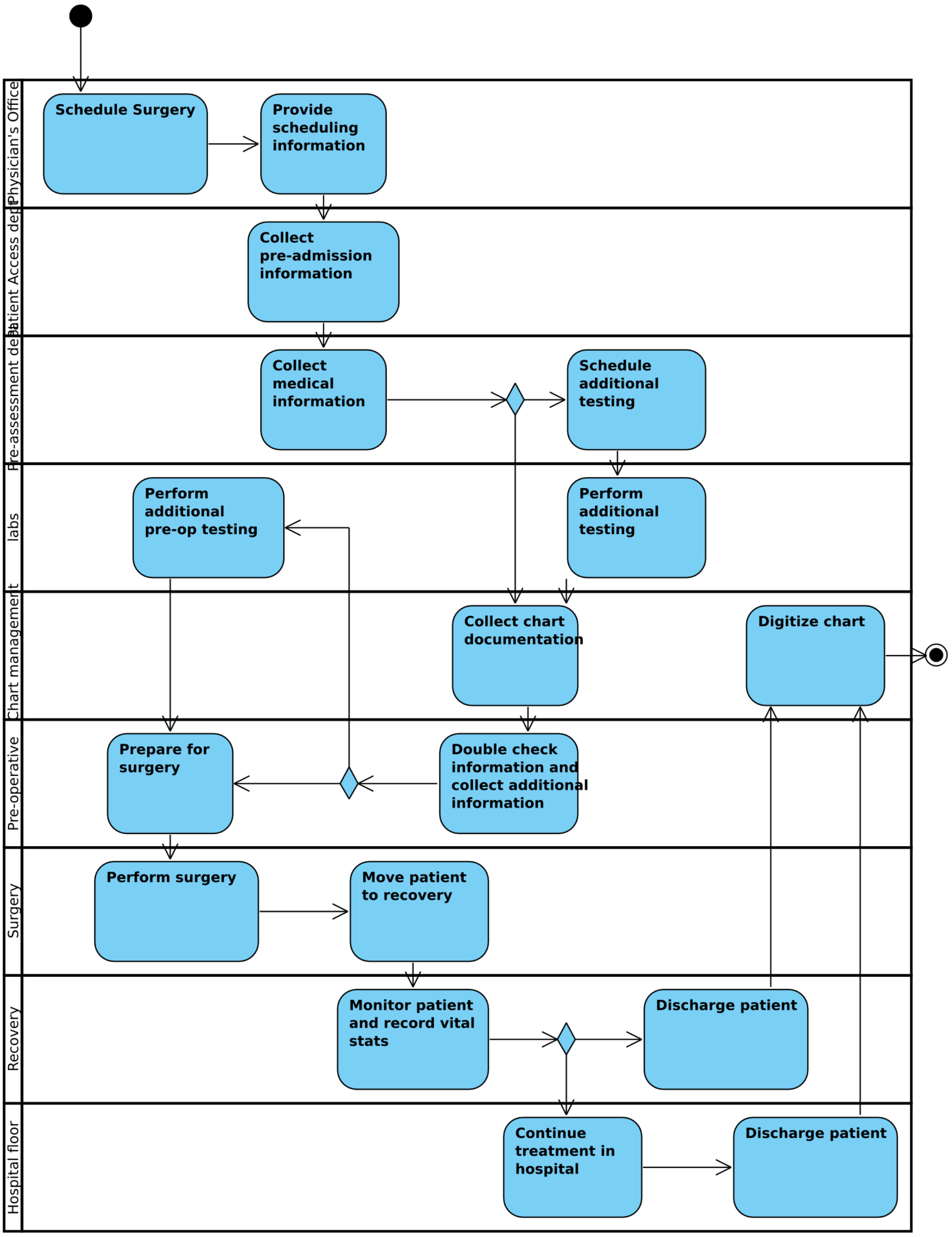

Physician's Office
Patient Access dept
Pre-assessment dept
labs
Chart management
Pre-operative
Surgery
Recovery
Hospital floor
Schedule Surgery
Provide scheduling information
Collect pre-admission information
Collect medical information
Schedule additional testing
Perform additional pre-op testing
Perform additional testing
Collect chart documentation
Digitize chart
Prepare for surgery
Double check information and collect additional information
Perform surgery
Move patient to recovery
Monitor patient and record vital stats
Discharge patient
Continue treatment in hospital
Discharge patient

Figure 3.2: Northside Hospital Activity diagram

2. *Medical Testing*
   When the basic patient demographics have been collected, the process continues to a pre-surgery assessment (PSA). PSA is staffed by registered nurses (RN's), who are responsible for capturing the patient's *medical information*. The medical information is collected during a telephone assessment and includes elements like prior surgeries, family history, allergies, prior diagnosis, reason for surgery, etc.

   During the PSA the RN will determine whether the patient will need to come in for further testing, or whether he can come in the day of surgery, perform additional tests then and move on to surgery.

   If further testing is required, the RN will schedule an appointment for testing for the patient so he can come in prior to surgery, or they will come in the morning on the day of surgery and fast-track the tests.

3. *Pre-surgery*
   A physician is required to dictate history and physical (H&P) prior to surgery. If the dictation is done in-house, chart management can retrieve those dictations from the system. If the dictation is done in the physician's office, H&P is brought in on hard-copy only. Finally, if no H&P is available, the physician is required to fill out a summary form before surgery.

   Once the patient is pre-assessed, 'chart management' collects all required chart documentation prior to the day of surgery. This includes lab results, radiology results, history and physical (H&P), and any other paperwork.

   On the day of surgery, the patient will come in at the appointed time.

   When the patient comes in, he will meet with a nurse to go over all the information that was provided earlier and make sure it is all correct. Additional paperwork is also filled out at this time.

   If there are any tests that need to be done prior to surgery, patients will go in to the pre-testing area and their test will be performed. If the results are not available quickly enough, pretesting will call the lab for them.

   After pre-testing, the patient will move on to the pre-op area, where they get prepared for surgery. This includes additional paper work, a visit by anesthesia for consent, etc. The patient is accompanied by a nurse the entire time in pre-op.

4. *Surgery*
   From pre-op, the patient goes to surgery. During/after surgery, operative notes are dictated that become part of the patient's medical records. In the operation room, a circulating nurse keeps track of everything that is in the operating room. At the moment this is done paper-based, but it is expected to change to electronics-based in a few months.

   All supplies and actions involving those supplies is captured on a case card by the circulating nurse. After the surgery is done, the case card is sent to the surgery billing office. Other roles that are part of the OR crew are techs, scrub techs, circulators, nurses, physician's assistants (PA's), nurse technician's (NT's), physician, anesthesiologist.

5. *Post-surgery*

   Patients are moved to recovery when surgery is completed. In recovery, the anesthesiologist will check on the patient and document his findings and nurses will monitor the patient's various vital signs and recovery information.

   After recovery, patients are discharged to home (80% of cases), or if they stay in the hospital go to the appropriate floor (ICU or regular bed).

   If the patient stays in the hospital (in-patient), the responsibility for the patient is transferred back to the nurse on the floor and the admitting physician. In an out-patient scenario, a physician will give a prescription to the patient ahead of time so he can have it filled before he goes in. That way, the medication is available immediately after discharge.

6. *In-patients*

   When in-patients need prescription drugs, such as pain killers or anti-rejection medication, there is an automated procedure that receives the prescription and checks it for conflicts with other medications, medical conditions or allergies. If there are no objections, a fully robotic system will collect the appropriate medications from storage, and send it to the floor where the patient resides.

   The pharmacy has access to the medical information that was captured during the pre-assessment activity.

   A nurse will then collect the medication using a special key, and visually inspect the medicin's label to make sure it matches to the information contained in the patient's records. Before the medication is administered, a doctor will perform a final check.

   If necessary, nurses have access to results of lab tests, dictation results, etc. Lab reports and radiology results are immediately available online.

   The primary nurse for a patient can also place orders using an online system.

7. *Discharge*

   After a patient is deemed well enough to leave the hospital, he is discharged. During the whole process that started when the patient arrived at the hospital for his surgery appointment, up to the point where he is discharged, all records are currently kept on hardcopy. These medical records are only scanned into the system when the patient leaves the hospital. Test results (lab works, radiological testing, etc) is available electronically.

8. *Billing*

   Billing information is inaccessible to physicians or to nurses, however it will be provided to physician's offices for their own billing.

All activities that take place while the patient is in the hospital are coded onto medical charts. That medical information is entered into an information system by ‘coders’, who can only access documents that are relevant for coding for reimbursement.

Coders have access to H&P, discharge summaries, radiology reports, dictations. They will not have access to patient guidelines, etc.

There are also individuals who analyze records for missing documentation, who will only have access to the documents that they are analyzing for.

Once the information is coded completely, it is sent to the billing department. The billing department does not have access to medical information at all. If they need additional medical information (for example, for an insurance claim), they need forward that request to go through the medical records department who will send the response to the original requestor.

In addition to the medical information and the business-related information there is also in information flow for materials management. This is done fully automatically. Materials management does not have access to the chart at all.

9. *Remotely accessing patient information*
   Physicians and physicians staff are able to obtain access to patient information from remote locations. They do this by logging in to the hospital’s Physician’s Web Portal. The portal provides a front-end to many other systems, such as radiological systems (for X-rays), laboratory systems (for blood and urine testing), and to the hospital’s core systems.

   Staff members have a more limited access to the portal, since they do not require access to a patient’s full medical records.

   At the time, there are no automated connections to outside organizations, with the exception of the transcripts of doctor’s patient records.

3.2.5 Running example

Throughout the remainder of this thesis, we will adopt a running example, which is based on a simplification of the case study. The running example will consider only four subjects, who play only a few roles. There will be two physicians, one nurse and one patient. One of the physicians is treating the patient, the other one is not. The nurse works at the hospital floor on which the patient resides. In the scope of the case study, we identify two services: a charting service, which is used to track patient’s medical information, and the pharmacy service, which physicians can use to prescribe medication to patients, and which dispenses medication to the nurses so that they may administer it to their patients.

In the running example (see Figure 3.3), we distinguish a number of subjects: John, Mark, Mary and Sue are people in the hospital. John and Mark are physicians, Mary is a nurse on the oncology ward and Sue is a patient who is treated for cancer by John. In addition to these human subjects, we also distinguish two services: the charting service and the pharmacy service.

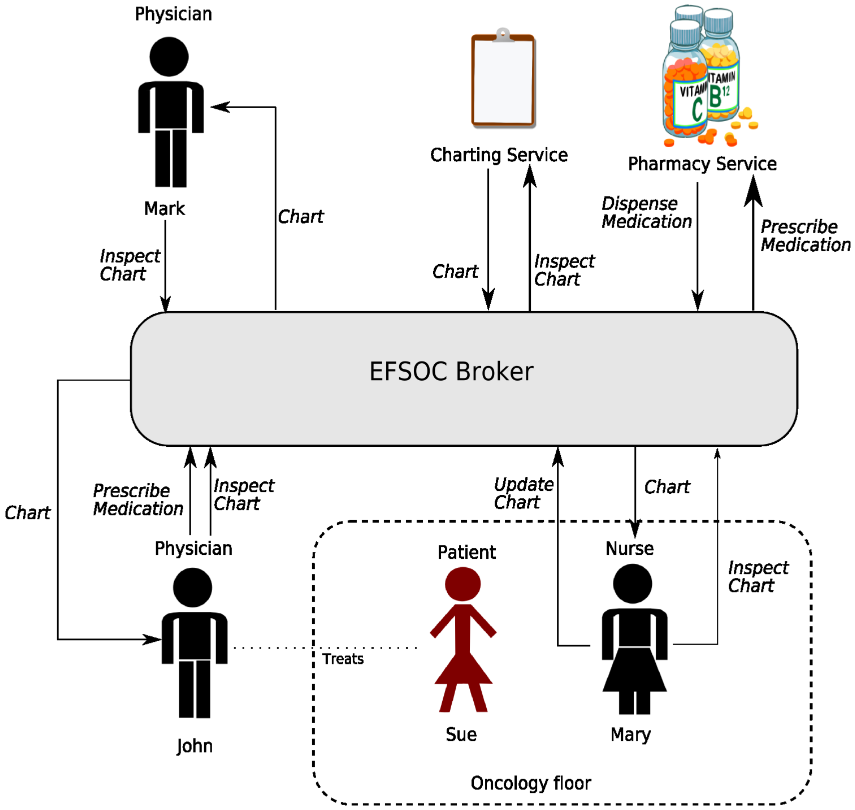

Physician
Mark
Inspect Chart
Chart
Charting Service
Chart
Inspect Chart
Pharmacy Service
Dispense Medication
Prescribe Medication
EFSOC Broker
Chart
Prescribe Medication
Inspect Chart
Physician
John
Treats
Update Chart
Chart
Patient
Sue
Nurse
Mary
Inspect Chart
Oncology floor

Figure 3.3: Running example

The running example assumes that Mary, in her role of nurse on the oncology ward, and John, in his role of treating physician, both need to inspect and update Sue's chart. John also needs to be able to prescribe medication for Sue, which will be administered by Mary.

3.3 Concepts

The EFSOC conceptual model combines aspects originating in role-based access control and messaging using event brokers and applies them to a service-oriented architecture. The EFSOC model is graphically represented in Figure 3.4. At its most basic level, it is comprised of the following concepts:

1. *Subject*; A subject is any entity that interacts with a service. In other words, anyone or anything that has the ability to publish, discover, invoke or compose new services and/or operations is considered to be a subject. As an example, consider the person called 'john'.

2. *Role*; A role represents a business function that is simultaneously or successively assumed by different subjects. In the example, we identified a role called 'physician'.

3. *Principal*; A principal is an entity that is subject to access control. This entity can be a subject or a role.

4. *Event*; An 'occurrence' that has the potential to prompt a subject or a role to exhibit certain behavior. In the case study that we performed, an event is generated when a physician schedules a surgery on behalf of a patient.

5. *Operation*; An operation is an atomic unit of work which interacts via a well-defined message-based interface. There are two types of operations: event operations and role operations. Event operations are operations that apply to events, such as 'send' or 'subscribe', while role operations are operations that apply to roles. For example, an example of a role operation is 'assign', which in case of the example is represented by the fact that 'john' is assigned to the role of 'physician'.

6. *Service*; In economic terms, a service is seen as work done by a person or group that benefits another. In the context of service-oriented computing, services consist of atomic operations which provide some kind of (computational) benefit, or are aggregations of other services, or a combination of both.

The access control system must be able to restrict the creation, modification or removal of subjects, roles and events, and it must be able to restrict the execution of event operations and role operations. The grounds to base these restrictions on must be able to be freely specified for each service cluster.

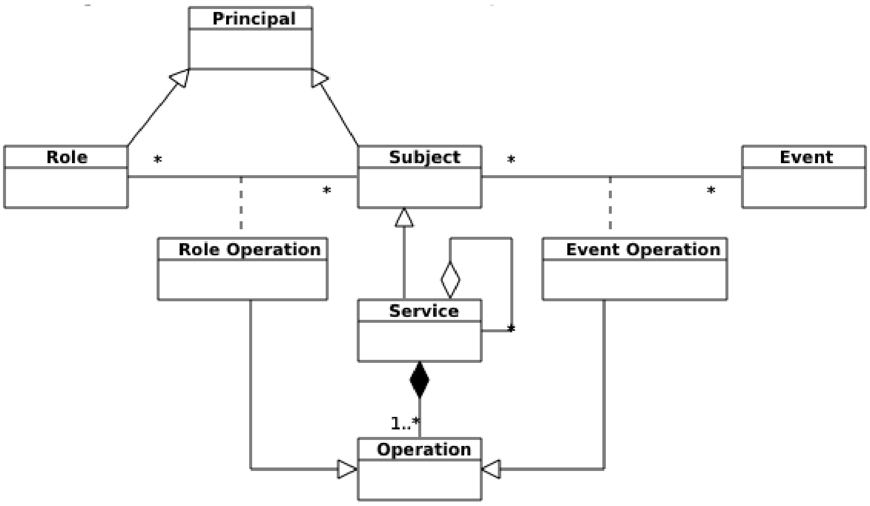


Figure 3.4: The EFSOC Conceptual Model

EFSOC can be seen as a state-machine, with each state representing a set of roles, subjects, role assignments, etc., at any given point in time. Since the values of the EFSOC constructs can only change via a predefined number of operations, state transitions will only take place in a controlled fashion. As such, to realize a secure system, we need to ensure that the initial state of the model is secure, and that all subsequent state transitions are secure. We do this by describing all possible state transitions (i.e., role operations and event operations) in the following sections.

3.4 Event operations

In SOC, services interact by exchanging messages. Messages may result in the invocation of a service's operation, or may be sent by a service as the result of the execution of an operation.

Events, which represent such service interactions, may be manipulated via *event operations*. Event operations represent the possible ways in which such interactions can take place. For example, to represent that an event may be sent by a subject, an event operation type 'send' is defined.

The EFSOC event model is depicted in Figure 3.5.
The following event operations are supported by EFSOC:

1. *send* is used to send events;
2. *receive* is the counterpart of the send operation and is executed just before events are delivered to their recipients;
3. *publish* is used to announce a new event body type;
4. *subscribe* informs EFSOC that a subject wishes to be notified when events of a given type are generated;

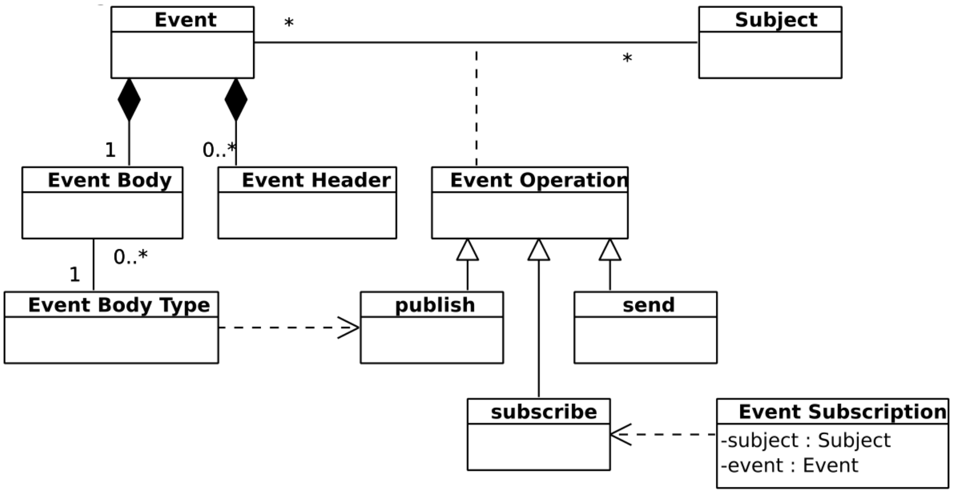


Figure 3.5: EFSOC Event Model

5. *unsubscribe* informs EFSOC that a subject no longer wishes to be notified when events of a given type are generated;

6. *unpublish* informs that events of a given type will no longer be generated.

Each of these event operations will be discussed in more detail in the following sections. Events consist of a number of event headers, and one single, typed, event body. The event headers contain information about the event (so-called meta-data), and will only be written by the EFSOC service.

Since subjects should be able to deploy message-level security, i.e. encrypt the bodies of the events that they send, the event body is not accessible to EFSOC (with some exceptions, which are described in Section 4.8.4).

3.4.1 Publishing events and subscribing to events

Subjects publish events to indicate that they would like other subjects to provide certain services, or to indicate the willingness to provide services themselves. The distinction between these two perspectives (which are not necessarily disjunct), is made by subscribing to events. Anyone is able to publish events. However, a subject who has subscribed to an event indicates that he is a candidate to observe events as they happen and that he is willing to react to them.

EFSOC provides two way of publishing events. First, subjects can publish events directly. Secondly, it is possible to publish an entire service description, for example in the form of a WSDL-document. The EFSOC service will analyze the document description and extract the event descriptions from the message types. Because it is possible to subscribe to events while publishing them, the second type of event publication will usually take place when a subject publishes a service that it is willing to provide.

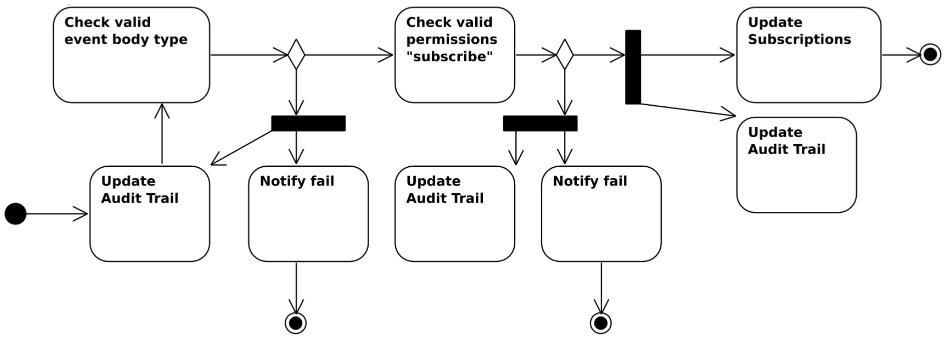


Figure 3.6: Subscribe to event activity diagram

When a subject subscribes to a particular event type, the EFSOC system will determine if it already knows the event type. If this is not of the case, the event subscription will fail. If the event did exist, a permission check will be made to evaluate whether or not the subject is allowed to subscribe. If the subject is allowed to subscribe, the list of available subscribers will be updated.

Another approach that could be followed when a subject attempts to subscribe to a non-existing event type is to implicitly publish it at that time. We choose not to do it in EFSOC, as we believe that any action that is implicitly taken may result in an opaque state of the system, which is not beneficial to its overall security.

The event subscription business logic is depicted in the UML Activity diagram in Figure 3.6.

3.4.2 Sending and receiving events

Sending and receiving is a straightforward process. When a subject generates an event, a security check will be performed to determine whether the subject has the proper permissions to send the event in the first place. If the subject does have adequate permissions, the event service will determine which subjects are going to handle the event for the subject. It does so by first determining who are subscribed to the particular event type. With the selection of subjects that follows from that check, additional checks, which are driven by business rules, may be performed. As a result, a possibly empty set of subjects remains. For each of these subjects an additional security check is performed to determine if they have enough rights to receive the event. Of those who are entitled to receive the event, the appropriate event handler will be called. Each of the steps that are taken will be recorded into the audit trail.

The process of sending an event is graphically illustrated in Figure 3.7.

3.4.3 Events in context

EFSOC maintains an accounting trail of all operations that are executed. As a result, if the need arises to establish an audit trail, all events that are sent were recorded

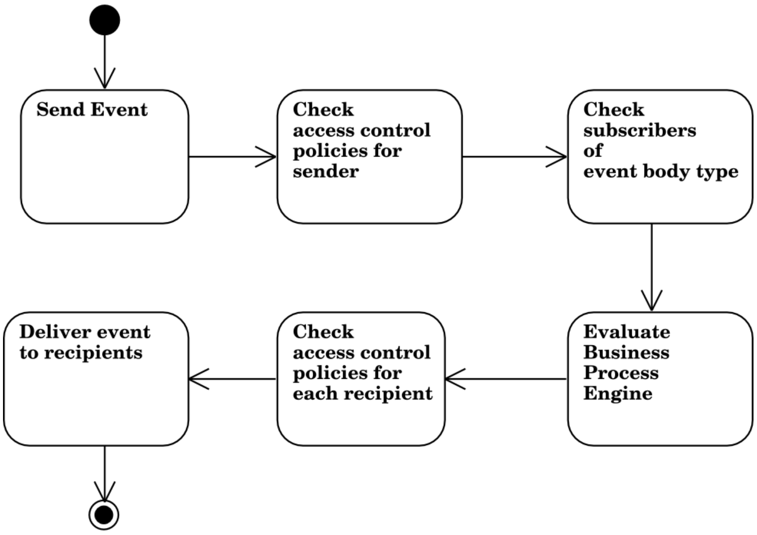

Send Event
Check access control policies for sender
Check subscribers of event body type
Evaluate Business Process Engine
Check access control policies for each recipient
Deliver event to recipients

Figure 3.7: Sending events

and are available for inspection.

As a consequence, events can be viewed in their relationships to other events. Such relationships may be used to specify certain constraints. EFSOC distinguishes a number of relationship types and/or constraints:

1. *Chronological relationships*

   Events are chronologically related because event operations are executed at a particular point in time. A chronological constraint can be that an event cannot be sent between 5pm and 7am on working days.

2. *Sequential relationship*

   Events are sequentially related if they occur after each other, in time. Examples of sequential relationships are ’event a is sent after event b’, or ’event a and event b are sent simultaneously’.

3. *Causal relationship*

   A causal relationship if a special kind of sequential relationship. Two events are causally related if one event is the result of another. A prerequisite for this is that the causing event must take place *before* the caused event.

   Causal relationships are relationships that cannot be automatically deduced. Rather, if a subject sends an event that is caused by another, he is required to explicitly mention that fact. A discussion of the trust-implications of supporting the causality header can be found in Section 2.5.1.

3.5 EFSOC and the Enterprise Service Bus

The ESB enables an SOA by providing the connectivity layer between services. The definition of a service is wide; it is not restricted by a protocol, nor does it require that a service is described by WSDL (Schmidt et al., 2005). Instead, the ESB concept—much like EFSOC—requires that a meta-data description of a service is published to a central repository.

The ESB must be viewed as an infrastructure to connect services. EFSOC's event operations provide a similar functions. The ESB-concept is sufficiently generic that EFSOC's event processing capabilities can be viewed as an ESB as well.

Most ESB-implementations provide support for subscription-based notification. However, since the ESB is considered as a pure connectivity layer, security requirements are often pushed out of scope, and must be added on at a later point.

EFSOC provides a platform that embeds security requirements, and especially access control facilities, from the start, rather than as something that must be added at a later point.

3.6 Role operations

Roles may be manipulated via *role operations*. For example, to assign a role to a subject, the role operations ‘assign role’ is defined.

The following role operations are provided by EFSOC:

1. *assign role* is used to map subjects to roles;
2. *activate role* is used to activate a subject-role mapping;
3. *assign attribute* is used to assign a value to a role attribute;
4. *revoke role* is used to unmap subjects from roles;
5. *deactivate role* is used to deactivate a subject-role mapping

Each of these role operations will be discussed in more detail in the following sections.

3.6.1 Subjects and roles

Role-based thinking has attracted a large amount of attention in the access control community, and more specifically in the role-based access control field. (Sandhu et al., 2000) (Ferraiolo et al., 1999) (Hamada, 1998) describe several variations of models for role-based access control. The principal reasons for adopting a role-based approach are often cited to be a reduction of cost of administering access control policies (Gavrilla and Barkley, 1998), which leads to a higher quality of work, and better alignment with the way that organizations operate.

While much work has been done in the area of role-based access control, the concept of a role has mostly been intangible and implicitly assumed to be known. In the context of the service oriented architecture, the concept of a role is commonly interpreted as a reflection of the type of a participant (i.e., service provider or service requester). However, this view is too coarse in granularity, and does not provide any relationship with the business domain.

In this thesis, we will consider a *role* as a function that is simultaneously or successively assumed by different subjects. Consequently, at any one point in time, a role can be viewed as a set of subjects. The semantics of a role depends on the service cluster in which it is deployed and is assumed to be commonly accepted by the participants in the cluster.

Several authors have pointed out that considering a role as merely an atomic label, as is often done in role-based access control, is not adequate (Li et al., 2004; Lupu and Sloman, 1997; Giuri and Iglio, 1997). While RBAC allows direct inheritance of roles, such inheritances are all-inclusive. In other words, when a role is defined in an inheritance, it will inherit *all* permissions that the other role has.

Lupu states that an organization may contain large numbers of roles with few differences between them, and advocates the use of role classes from which instances can be created. Giuri advocates the use of role templates, which extend the concept of role to encapsulate and compose parameterized privileges. Li introduces a role-based trust management framework in which a role name is constructed by applying a role identifier to a tuple of data terms.

We suspect that roles consisting of atomic labels are sufficient when the roles are significantly semantically different, and when assignments of subjects to roles are fairly static. When the role assignments change often, or when there are only small semantic differences between the roles, parameterized roles provide a better solution. However, insufficient research has taken place to state this as a conclusive fact.

The EFSOC framework supports parameterized roles by providing the ability to associate attributes with each role, as shown in Figure 3.8. The diagram shows the two role operations that were mentioned earlier (assign and activate) and introduces a further refinement of the assign operator by distinguishing between assigning roles to subjects and values to role attributes.

The following example will show the usefulness of role attributes. Assume the situation in which Sue is treated by John. Without using role attributes, this piece of knowledge would have to be represented by either introducing a special role ‘sue’s physicians’ or ‘john’s patients’. Consequently, specialized roles would have to be introduced for each patient, or for each physician. By doing so, one of the paramount benefits of a role-based approach—the reduced management overhead—would be eliminated almost completely.

Instead, we introduce an attribute ‘patient’ to the ‘physician’ role. Now, John can be assigned the role physician, and the role attribute ‘patient’ will be assigned the value ‘sue’.

To capture the fact that one physician is able to treat multiple patients, EFSOC provides the ability to implement multi-valued role attributes.

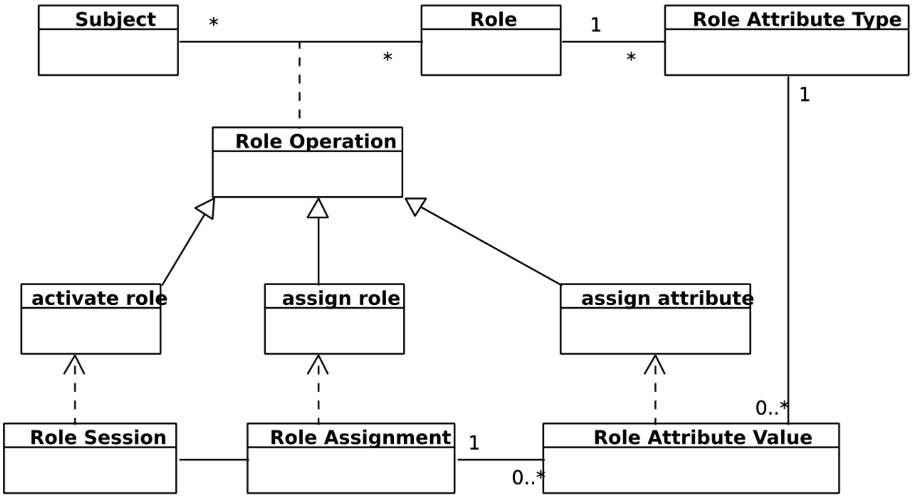


Figure 3.8: EFSOC Role Model

3.6.2 Role assignments and role sessions

Roles must be explicitly assigned to or revoked from subjects. Such role assignments can be updated at run-time. When a subject is assigned to a role, that subject is considered to be a *member* of that role.

Role memberships can be assigned semi-permanently, in which case they represent a subject's function in a business process, or assigned ad-hoc and temporarily, in which case they apply to a specific interaction. A *role assignment* represents the fact that a subject may play a certain role and is represented by a subject-role mapping.

Role assignments give a subject the *ability* to assume a certain role. Before such a role assignment can be used, a role session needs to be created. A *role session* represents the run-time usage of role assignments. More than one role assignment may be active at any time, and therefore each role session may be associated with multiple role assignments.

As a consequence, the relationship between role assignment and role sessions is constrained. A subject can only have roles active which are indeed assigned to him.

A recurring term in existing access control literature is that of a principal. The *principal* is anyone who can receive authorizations to execute privileged operations (Saltzer and Schroeder, 1975; Gasser et al., 1989).

EFSOC ties access control closely to operations on messages and roles, and as such bases its access control structure around those on who's behalf operations are executed. Since all subjects must play one or more roles, and all roles and subjects may be principals, all permissions are eventually granted to subjects.

However, in line with the findings of the research groups who pursued role-based access control (Sandhu et al., 1996), EFSOC also provide facilities for granting permissions to *roles*, which function as dynamically changing groupings of subjects.

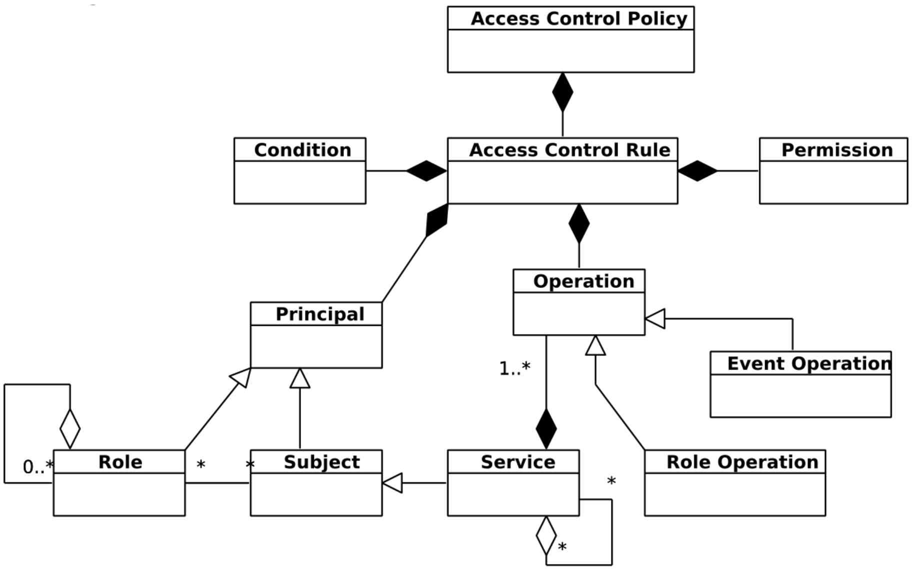


Figure 3.9: Access Control Model

3.7 Access control

The EFSOC service provides access control on a number of different levels: transport-level access controls which regulate 'technical' aspects of the event relay, such as network addressing, SSL/TLS protection, encryption schemes, etc. and message-level access controls, which apply to the events themselves. Our solution groups access control requirements in access control policies, which are administered and evaluated *per subject*.

This section addresses the different elements that make up access control requirements, and the format in which they are crafted.

Access control lists are usually managed by a single security administrator. However, since each service is able to specify access control rules for its own incoming and outgoing events, that assumption is invalid in our approach.

Details regarding the processing of access control rules are further provided in Section 3.7.1.

EFSOC access control rules consist of four parts, as graphically represented in Figure 3.9.

1. *An operation*
   The operation to which the access control rule applies. For example, if the access control rule attempt to allow certain subjects to send a particular event, the operation section would contain the value *send*. Valid values for this parameter include the list of event operations and role operations.

2. *A permission*
   Whether or not the permission will be granted. Valid values are *permit* and *deny*.

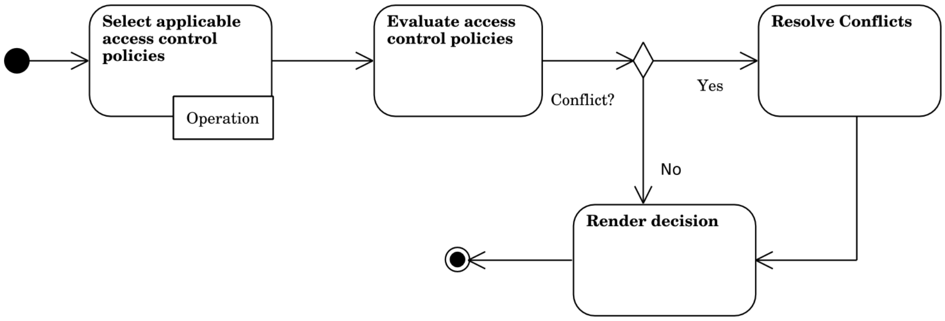

Select applicable access control policies
Operation
Evaluate access control policies
Conflict?
Yes
Resolve Conflicts
No
Render decision

Figure 3.10: Taking access control decisions

3. *Principals*
   A list of zero or more principals to which the rule applies. If no principals are listed, the rule will always apply. If one or more principals are listed, the rule will only apply to those principals. Principals can be specified by listing subjects or roles.

4. *Conditions*
   The conditions under which the rule applies. There can be many different conditions, which may apply to the message itself, to the transport level, or to message context elements, such as temporality or causality. The conditions that are supported by EFSOC are further elaborated in Section 4.8.4.

3.7.1 Taking access control decisions

For any access control mechanism you have to know precisely in which order different access criteria are checked (Gollmann, 2006).

As a basic principle in EFSOC, access control decisions are made by evaluating access control policies (see Figure 3.10).

The access control deciding process begins by determining which access control policies are applicable. Since access control decisions are taken based on the operations being performed, the selection criteria depend on that operation.

Once it has been determined which access control policies apply, they are evaluated. EFSOC does not require that only one access control policy may apply at any point in time, which introduces the possibility that evaluation of the appropriate policies may lead to a potentially conflicting situation. If that is not the case, a decision is made.

Since we need to ensure that all state transitions are done securely, conflicts must be resolved. In case of conflicting results of policy evaluations, negative decisions will take precedence over positive ones. In other words, if just one of the policies denies permission, the entire evaluation will evaluate to a denial of access.

Access control policy selection depends on the operation that is being executed. For each of the operation that EFSOC provides, we discuss the appropriate access control policy selection criteria below.

1. *publish event body type*
   The publish event body type operation is executed when a subject attempts to introduce a new event body type. The policies that are relevant are listed below.

   (a) The EFSOC global policy
   (b) The subject attempting to publish the event body type.

   The subject of both policies will be the subject attempting to publish the event body type. When a new event body type is published successfully, the subject who published it will be considered the event body type's owner. Each event body type may only be published once at a time.

2. *unpublish event body type*
   The unpublish event body type operation is executed when a subject attempts to remove an existing event body type. The policies that are relevant are:

   (a) The event body type's owner
   (b) The subject attempting to unpublish the event body type

   The subject of the policy will be the subject attempting to unpublish the event body type. If an event body type to which subjects are still subscribed is unpublished, they will be unsubscribed automatically. All subjects that are unsubscribed implicitly will be notified of the unsubscription.

3. *send event*
   The send event operation is executed when a subject sends out a new event. The policies that are relevant are:

   (a) The sender's own policy.
   (b) The event body type owner's policy.

   The subject of both policies will be the subject who sent the event. Only event body types that are published when the event is sent will proceed.

4. *receive event*
   The receive event operation is executed when a subject is about to receive an event. The policies that are relevant are:

   (a) The receiver's own policy.
   (b) The sender's policy.
   (c) The event body type owner's policy.

   The subject of all three policies will be intended recipient of the event message. In order to receive an event, the recipient must be subscribed to it. Additionally, the event body type must be published are the time of reception.

5. *subscribe event body type*
   The subscribe operation is executed when a subject intends to receive messages of a specific event body type. The policies that are relevant are:

   (a) The subscriber's own policy.
   (b) The event body type's owner's policy.

   The subject of both policies will be the subscriber. The event body type may only be subscribe to if it is published at the time of subscription.

6. *unsubscribe event body type*
   The unsubscribe event body type operation is executed when a subject wishes to unsubscribe from a specific event body type. The policies that are relevant are:

   (a) The unsubscriber's own policy.

   A subject may only unsubscribe from event body types to which he previously subscribed.

7. *discover event body type*
   The discover event body type operation is executed when a subject attempts to discover an event body type. The policies that are relevant are:

   (a) The subject's own policy.
   (b) The policy defined by the owner of the event body type.

   In both cases, the subject attempting to execute the discover operation will be the subject of the rule. Event body types may only be discovered if they are currently published.

8. *publish role*
   The publish role operation is executed when a subject attempts to introduce a new role. The policies that are relevant are:

   (a) The EFSOC global policy
   (b) The subject attempting to publish the role.

   The subject of both policies will be the user attempting to publish the role. When a new role is published successfully, the subject who published it will be considered the role's owner.

   Each role may only be published once at a time.

9. *unpublish role*
   The unpublish event body type operation is executed when a subject attempts to remove an existing event body type. The policies that are relevant are:

   (a) The event body type's owner

   (b) The subject attempting to unpublish the event body type

   The subject of the policy will be the user attempting to unpublish the event body type. If a role to which subjects are still assigned is unpublished, they will be unassigned automatically. All subjects that are unsubscribed implicitly will be notified of the unsubscription.

10. *assign role*
    The assign role operation is executed when a subject attempts to be assigned to a new role. The policies that are relevant are:

    (a) The role owner's policy
    (b) The policy of the subject who will be assigned the role.
    (c) The policy of the subject who attempts to assign the role.

    The subject of all policies will be the user attempting to be assigned to the role. Each role may only be assigned to the same subject once at a time. Roles must be published when they are assigned.

11. *revoke role*
    The revoke role operation is the opposite of the assign role operation. It will take away a role assignment from a subject. The following policies are relevant:

    (a) The policy of the subject who will be revoked from the role.
    (b) The policy of the subject who attempts to revoke the role assignment.

    The subject of the policy will be the user attempting to be revoked from the role. A role may only be revoked from a subject if that subject is assigned to the role.

12. *activate role/deactivate role*
    The (de)activate role operation is executed when a subject attempts to (de)activate an assigned role. The policies that are relevant are:

    (a) The role owner's policy
    (b) The policy of the subject who attempts to (de)activate the role.
    (c) The policy of the subject who's role is (de)activated.

    The subject of all policies will be the user attempting to (de)activate to the role. Roles may only be activated by subjects who are assigned to the role. Once a role is activated, it may not be activated again, unless it is deactivated first. Only active roles may be deactivated.

13. *assign role attribute*
    The assign role attribute is executed when a role attribute is associated with a role. The policies that are relevant are:

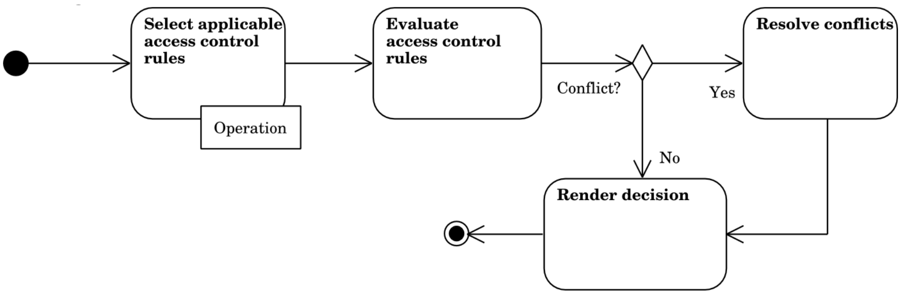

Select applicable access control rules
Operation
Evaluate access control rules
Conflict?
Yes
Resolve conflicts
No
Render decision

Figure 3.11: Evaluating access control policies

(a) The role owner's policy.

(b) The policy of the subject attempting to set the role attribute value.

The subject of the policy is the user attempting to assign the role attribute. Role attribute types may only be assigned to subjects to whom the corresponding role is assigned.

14. *assign role attribute value*
The assign role attribute value operation is executed when a role attribute value is set. The policies that are relevant are:

(a) The role owner's policy.

(b) The policy of the participant in the corresponding role assignment.

(c) The policy of the subject attempting to assign the role attribute value.

3.7.2 Evaluating access control policies

Access control policies are evaluated in much the same way as the overall access control decision is made. Figure 3.11 illustrates this graphically.

The process begins by selecting appropriate access control rules. The selection is made by two criteria: the operation to be performed, and the subject performing it. Each applicable access control rule is evaluated and the results are stored.

After all applicable rules have been evaluated, a determination is made whether or not a conflict has arisen. If there is a conflict, it is resolved and an access control decision is taken. If there is no conflict, the access control decision is taken immediately.

Rule evaluation conflicts are resolved by determining the *priority* of each rule. Rules with a higher priority will take precedence over rules with a lower priority.

Since using prioritized rules is *optional* in EFSOC, conflicts may still take place. For this reason, each policy *should* have specified a default behavior that will be applied in case rules are conflicting. If no default behavior for conflicting rules is specified, and a conflict arises, permission will be *rejected*.

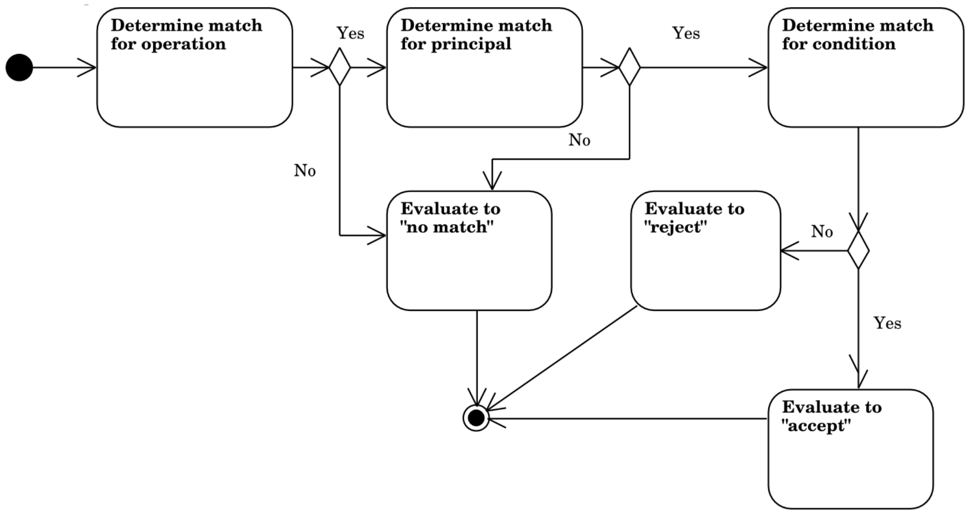

Determine match for operation
Yes
Determine match for principal
Yes
Determine match for condition
No
No
Evaluate to "no match"
Evaluate to "reject"
No
Yes
Evaluate to "accept"

Figure 3.12: Evaluating access control rules

3.7.3 Evaluating access control rules

Access control rules are easier to evaluate than entire policies. Each rule will only result in one decision: permit or reject. As such, there is no need to address conflict resolution in rule evaluation.

Figure 3.12 illustrates the rule evaluating algorithm. Evaluation begins by determining whether or not the rule applies to the requested operation. This filtering should already have been done by the rule selection activity in the policy evaluator, but since the rule base may have changed since then, it is performed again.

Next, a decision is made regarding the principal of the rule. If the subject of the event matches the principal of the rule, evaluation will proceed. If it does not match, the rule will be deemed to be inapplicable, and processing will stop without rendering a decision.

If the rule is deemed applicable (operation matches and principal matches the subject of the event), the rule's condition will be evaluated. If the condition evaluates to `true` permission will be granted, else it will be rejected.

3.8 Architecture

The concepts introduced in the previous section provide a basic vocabulary which describes the EFSOC Service-Oriented Architecture. In this section, we use these concepts to design a system which suggests how they can be used in application scenarios.

Figure 3.13 graphically represents the different components of which an EFSOC event service consists. The components are:

1. *Transport-level access control module*
   The transport-based access control module enforces access control rules that

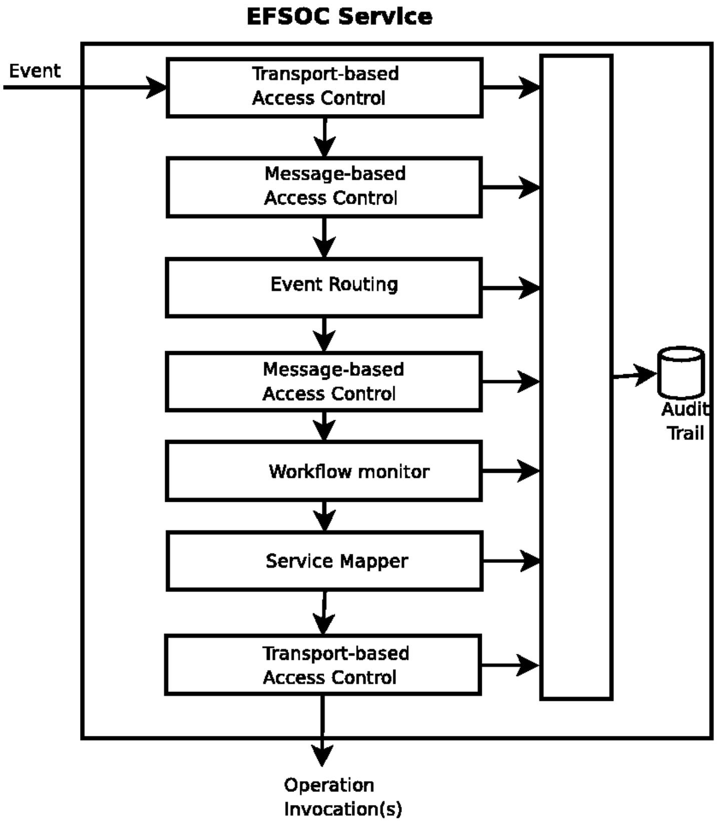

EFSOC Service
Event
Transport-based Access Control
Message-based Access Control
Event Routing
Message-based Access Control
Workflow monitor
Service Mapper
Transport-based Access Control
Audit Trail
Operation Invocation(s)

Figure 3.13: EFSOC Service Architecture

contain transport-level conditions, such as requirements on target IP address or source IP address in case of a TCP/IP-based service, protection status of the connection (for example, requirement to possess an SSL certificate issued by a certain certificate authority), etc.

2. *Message-level access control module*
   The message-level access control module enforces access control rules that contain message-level conditions and message-context level conditions. For example, requirements on this level may specify that the event body must be digitally encrypted and signed using WS-Security, or include requirements with respect to the causality of messages, or (in case of unencrypted event bodies), requirements on the contents of the event body.

3. *Event routing module*
   The message routing module provides the facilities for publication and subscription to event body types. Using the event routing module, EFSOC is able to determine which subjects are eligible to receive an event based on their subscriptions.

4. *Workflow monitor*
   While the workflow monitor is part of the EFSOC architecture, we do not elaborate on it in this thesis. The goal of the workflow monitor is to further refine the set of recipients that is determined by the event routing module based on a set of previously defined workflow specifications.

EFSOC distinguishes between business rules and security rules. Since event bodies may be encrypted in such a way that the EFSOC system cannot decrypt the contents of the body, security rules can only address the principal of the rule (role or subject), the rule's event body type, the operation being carried out, and any possible additional event headers. Rules that take any of these variables in account are considered security rules.

When EFSOC does have access to the event body data, additional conditions can be specified that apply to that event body data. Rules containing conditions on such data are called business rules.

For example, assume an hypothetical situation in which a physician is allowed to prescribe a particular kind of drug, but not another. The fact that the physician may prescribe medication is a security rule, since it applies to the principal role *physician*, the event body type *PrescribeMedication*, and the operation *send*. Any additional requirements, such as the requirement that a surgeon may not prescribe pain killers, would require the event body contents to be inspected. As a result, such rules are business rules.

If event bodies are not encrypted, EFSOC is able to enforce business rules. If event bodies are encrypted, EFSOC will not be able to evaluate rules containing business rules. It is therefore recommended that business rules are implemented by services themselves, rather than by EFSOC's access control engine.

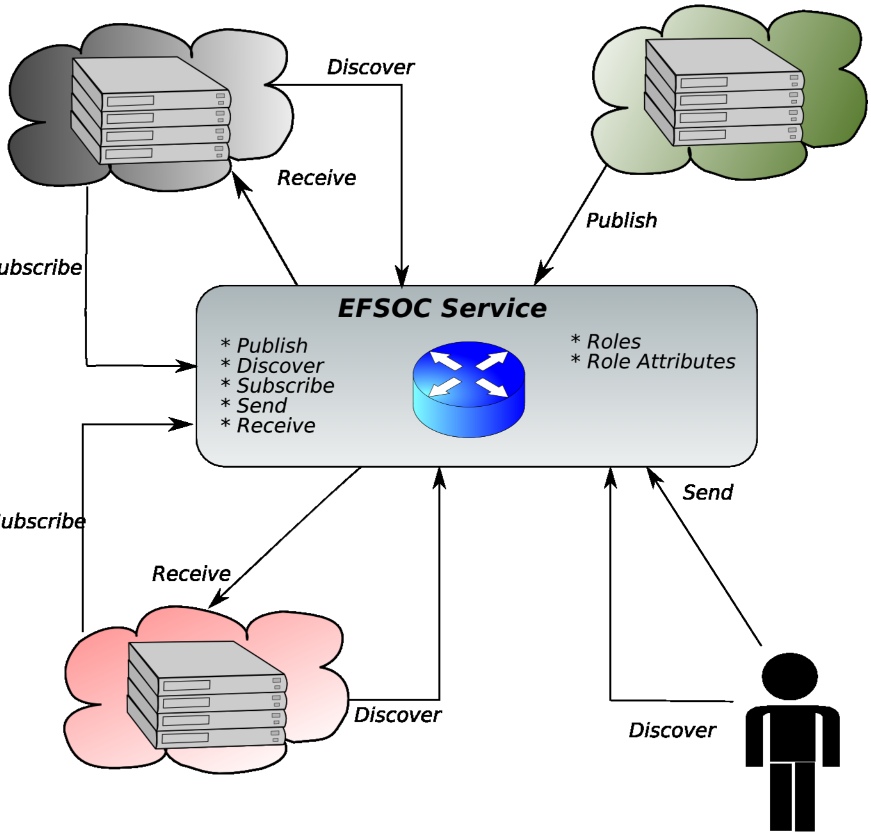

Discover
Receive
Publish
Subscribe
EFSOC Service
* Publish
* Discover
* Subscribe
* Send
* Receive
* Roles
* Role Attributes
Subscribe
Receive
Send
Discover
Discover

Figure 3.14: EFSOC Event Processing Overview

5. *Service mapper*
   The service mapper provides the binding of the EFSOC SOA to the underlying implementation framework. In case of an implementation which relies on web services, the service mapper will map an event to a service invocation.

Figure 3.14 illustrates the way that EFSOC would perform in the context of the running examples.

First, a service will publish a certain event body type, which makes it possible for others to discover it. Based on the discovery, subjects can subscribe to the event body type and send and/or receive messages.

In the running example, we assume that a number of events is available for inspecting and updating charts, and for prescribing and dispensing medication.

Assume that John attempts desires to inspect Sue's chart. He does so by generating an 'inspect chart' event, which contains information about the patient who's chart he wishes to inspect. Based on previous event subscriptions, EFSOC routes the 'inspect chart' event to the charting service.

On reception of an event, a first simple access control decision must be made to decide whether the event originates from a source on the network that is authorized to interact with the service. In case of an EFSOC service which is deployed on a TCP/IP-based network, such checks generally involve validating the source IP address of the packet.

The next check that is performed is a content check, and the event message will be inspected. This inspection step forms the core of the EFSOC access control approach, since it may involve checking the message itself for certain criteria, but also for checking the identity of the principal, or the roles that he plays, etc.

If the message passes this check as well, the event routing module is activated, which checks which services have subscribed to the particular event.

Having decided that, an additional message-based check is performed. The previous message check applied to the originator of the message, whereas this check will be executed for each of the intended recipients.

Finally, after checking the message itself, the service mapper will attempt to retrieve which operation needs to be called as an event handler. This check needs to be executed for each intended recipient.

The final check that is performed is a transport-level access control check. Unlike the first transport level access control decision, which applied to the originator of the message, this check will be executed for each of the potential recipients.

If the messages passes through all checks, the EFSOC service will then call the appropriate operation.

All actions are logged into the audit database, where they will be available for later inspection, if so necessary.

3.9 Example: Applying Access Control Policies

Consider again the running example. Before anything can happen, subjects need to *publish* the appropriate events, and others must subscribe to them. By publishing an event, the publisher takes ownership of it.

In case of the example, the following events are published:

Event body type name	*Owner*	*Body*
prescribe medication	pharmacy service	patient medication dosage number of doses
dispense medication	pharmacy service	patient medication dosage number of doses
inspect chart	charting service	patient
update chart	charting service	patient medical data

After publication, the pharmacy service *subscribes* to the prescribe medication event body type, indicating that it wishes to handle medication prescriptions. Mary (the nurse) subsequently subscribes to the dispense medication event body type to indicate that she is capable to administer drugs to her patients.

The charting service will subscribe to both the inspect chart event body type as

no.	*priority*	*description*
1	1	Default permission: refuse
2	1	Permission by conflict: refuse
3	1	Owner: pharmacy service
1	1	operation: send principal: role.physician permission: permit condition: event body type is prescribe medication and sender must treat patient
2	1	operation: send principal: role.nurse permission: permit condition: event body type is dispense medication and receiver must be nurse on patient's floor

Figure 3.15: Pharmacy's access control policy

well as to the update chart event body type, since it is responsible for maintaining all medical charts.

Having published the event body types, the respective owners will establish access control policies. These policies include:

1. *Pharmacy Service Policy*
   The pharmacy's policy dictates that medication may only be prescribed by physicians, and that medication may only be dispensed to nurses who are working on the same floor that the patient to who the medication is prescribed is being treated. The policy takes the form as shown in Figure 3.15

   The rows under the first set the double line in the table above depict the policy's preferences. The rows under the second set of double lines depict individual access control rules.

2. *Charting Service Policy*
   The charting service policy requires that all physicians may inspect any patient's chart, but that inspection by nurses is limited to the charts of patients on their floor. While physicians may inspect all charts, they may only update the charts of their own patients. Likewise, nurses may update the charts of patients on their floors.

   The policy takes the form as shown in Figure 3.16.

An interesting observation regarding these policies is that both policies constrain sending of events, rather than the reception of them. Since the pharmacy service "owns" the prescribe medication event and the dispense medication event, they are capable of doing so. The same is true for the charting service, which owns the inspect chart event and the update chart event.

The fact that those services own the respective event body types is not surprising, since they have the most interest in regulating the way that they are used.

Charting service access control policy		
no.	*priority*	*description*
1	1	Default permission: refuse
2	1	Permission by conflict: refuse
3	1	Owner: charting service
1	1	operation: send principal: role.physician permission: permit condition: event body type is inspect chart
2	1	operation: send principal: role.nurse permission: permit condition: (event body type is inspect chart or event body type is update chart) and sender must be patient's floor
3	1	operation: send principal: role.physician permission: permit condition: event body type is update chart and sender must be treating patient

Figure 3.16: Charting Service Access Control Policy

Now that all events have been published and subscribed to, the services can start to exchange information. For example, assume that John wishes to record the fact that Sue's hemoglobin count is too low on her chart. Doing so will result in sending an *update chart* event.

According to the algorithm explained in Section 3.7.1, the first step that EFSOC will take is to determine the appropriate policies. In case of a 'send' operation, such as is the case here, the sender's own policies apply, as well as the event body type's owner's policy. John has not specified a policy, but the update chart event's owner (the charting service) has.

It requires that the event may only be sent if Sue is currently under John's treatment. Since Sue is listed as a valid value of the role attribute 'patient', and since John is a physician, the policy will evaluate to 'permit', and the event will be sent.

Next, the 'receive' operation is executed for all subjects that have subscribed to the event's body type. In this case, the only subscriber is the charting service itself. The appropriate policies are selected by taking the receiver's policy, the sender's policy and the event body type's policy. The receiver's policy and the event's body type owner's policy are identical and do not constrain the event body type from being received. Additionally, since the sender (John's) policy does not specify any constraints, the event will be delivered to the charting service. The charting service will then proceed to update the chart on file.

3.10 Discussion

A solution can be considered secure when the initial state of the populated model is secure, and when all subsequent state transitions do not violate security constraints. A state is considered secure when all of the model's assumptions are unviolated, and when the model is populated with a desired situation.

Since an instantiated EFSOC model can only change its population by execution a limited number of operations, and each of those operations is subject to access control rules, the approach is secure, providing the initial state is a desired state.

3.10.1 EFSOC and Web Services

EFSOC provides a conceptual service-oriented architecture which is very similar to the more well known web services SOA. Unlike the web services SOA, EFSOC does not provide an actual implementation. Rather, it is a framework which can be implemented using existing technologies.

The principal difference between the approach followed by web services and the approach followed by EFSOC, is that EFSOC decouples services when they interact, whereas the web services paradigm advocates that services invoke each other directly.

While the web services paradigm eliminates the potential overhead that a service broker/mediator introduces into the architecture, I believe that the benefits of doing so outweighs the drawbacks. The principal benefits that I perceive are:

1. In a web services environment, services are critically dependent on each other, and on their implementation. If a service implementation changes locations, or disappears altogether, all services that depends on it must take corrective actions. In the EFSOC architecture, this is not the case, as EFSOC services depend on events, rather than on service descriptions.
2. Using a service broker, like in the case of EFSOC, provides a logical extension point for added value actions, such as auditing and logging, security, transaction monitoring, workflow enforcement, etc. Rather than burying a service-enabled application in an enormous stack of layers, protocols and specifications; each of which introduces additional complexity.

Having said this, the EFSOC SOA and the Web Services SOA are complementary. In designing the EFSOC service, we ensured that existing web services can rely on their tools and programming libraries, yet still obtain the benefits of using an EFSOC approach.

1. Services may publish their WSDL documents by treating the EFSOC service as a UDDI service. The EFSOC service will analyze the WSDL document and extract the appropriate events and call-back handlers from the WSDL document.
2. In EFSOC, services request an operation to be performed by generating an event. Sending that event can be done by EFSOC's web service.

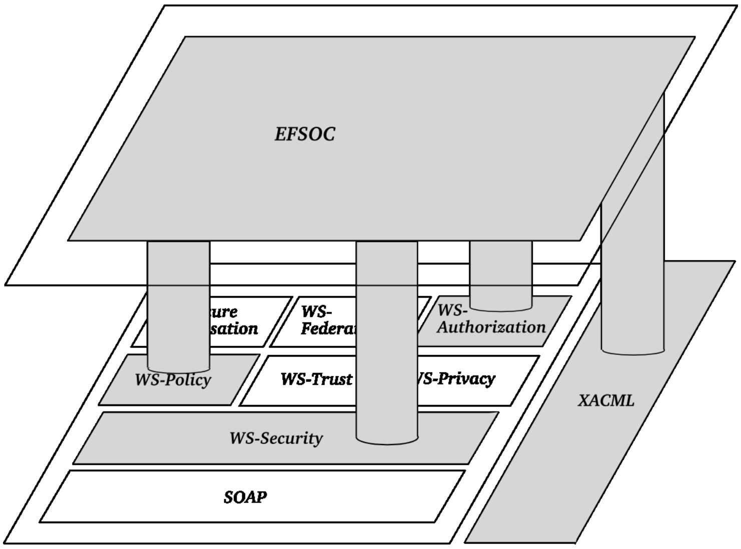


Figure 3.17: EFSOC in relation to the WS Security Roadmap

EFSOC messages should be implemented as SOAP messages. As a result, most (if not all) web services standards that are specific to protecting message content can still be used. For example, standards such as WS-Security and WS-SecureConversations are compatible with EFSOC.

The relationship between EFSOC and the different standards that comprise the Security Roadmap for Web Services Security is shown in Figure 3.17. It illustrates that EFSOC mostly rests on three pillars: message-level protection via WS-Security, the ability to specify access control policies via WS-Policy and the ability to formulate access control requirements and issue authorizations to subjects via WS-Authorization. While only three pillars are shown in the figure, the remaining specifications still apply to EFSOC, albeit to a lesser extent.

As illustrated in the figure, EFSOC also borrows from XACML. In particular, the separation between policy administration, policy information and policy enforcement is inspired by the XACML architectural model.

3.10.2 EFSOC and its design objectives

In Section 2.6.4, we listed a number of design objectives for the EFSOC service broker. These design objectives are listed below, and their manifestation in the overall EFSOC framework is discussed.

1. *Service Autonomy*; The service autonomy principle is safeguarded in the architecture of EFSOC by allowing each subject to specify its own access control policies. Access control policies will be evaluated for all events received by a subject, and all events that are sent by a subject. This allows each subject to remain fully in control of the messages it sends and received.

The drawback of the service autonomy principle is that it is possible to specify conflicting access control policies, in which one policy explicitly allows an event to be routed to a recipient, which another rule explicitly refused it to be delivered.

EFSOC processed the policies as follows: sender's send rules for himself, sender's receive rules for EFSOC, EFSOC's send rules for itself, sender's receive rules for intended recipients, intended recipient's receive rules for sender. If at any point in time, a negative access control decision is reached, processing will stop and the message will not be delivered.

2. *Containment*; Containment addresses limiting the extent to which an intruder can affect services after one or more security mechanisms have failed. EFSOC proposes a solution in which each message transfer goes to a number of stages. At each stage, access control decisions will be taken. Access control decisions may not be cached, or calculated ahead of time, resulting in a continuous re-evaluation of a subject's permissions.

   Invalid messages can only be sent when the security controls at all phases of a message relay fail, and when this happens multiple times in succession.

   Through this mechanism, EFSOC implements active security.

3. *Separation of Duty* (SoD); Separation of duties means that services can only be provided when a number of subjects (each playing separate roles) collaborate. EFSOC allows the implementation of separation of duty by specifying per-subject access control rules, or global SoD constraints by implementing them on the EFSOC access control level.

4. *Availability, Integrity and Confidentiality*; Availability, integrity and confidentiality are beyond the scope of this research, but can be achieved by implementing EFSOC as a value-added service on top of Web Services Technology, and by deploying the appropriate WS-Security and WS-Policy technologies.

5. *Auditability*; Auditability is achieved by routing traffic through the EFSOC broker. All operations that are executed lead to one or more entries in the EFSOC audit log. Audit log entries are timestamped and fingerprinted to ensure their integrity.

6. *Message-context level access control*; Message-level access control is available in EFSOC in the form of access control conditions with include temporal and causal operators. The operators place events in sequence, or in a logically causal relationship.

7. *Authentication*; Authentication is beyond the scope of this research. However, in the chapter on future research (Section 8.6), we do share some thoughts on identity management and identities spanning multiple EFSOC instances.

3.10.3 Delegation and role hierarchies

EFSOC does not provide explicit facilities for role hierarchies. The Role-Based Access Control model expects an access control model to support role hierarchies to comply with the $RBAC_1$ requirements. $RBAC_1$ requires role hierarchies to provide automated permission inheritance, which we view as a mechanism to facilitate *implicit* delegation.

In EFSOC, delegations are not needed for a number of reasons:

1. RBAC assumes that when a role is delegated from one subject to another, all privileges that are associated with that role are delegated too. In effect, it is not possible to delegate individual permissions to subjects, while there are many cases in which that is desirable.

   The least privilege principle requires that a subject only obtains the rights that he needs to execute his task. Delegating multiple rights, when only a subset of those rights is required, therefore constitutes a violation of that principle.

   Unlike full delegation of roles, partial delegation of privileges may be acceptable. However, whereas RBAC accepts the direct assignment of privileges to roles, EFSOC does not. Instead, EFSOC computes privileges using access control policies and access control rules. As a result, permissions can not be delegated.

2. While permissions cannot be delegated, because they cannot be addressed directly, it is possible to formulate a rule of which the principal is a subject. For example, assume that a service (e.g., the Charting Service) wishes to delegate the right to update charts to Mary, it could create a specific rule for this purpose.

3. Because of the prescribed order in which security policies are evaluated, delegating a permissions or a role would be pointless. The access control policies which are evaluated are known ahead of time, and delegations are simply represented by such rules.

   This is illustrated in Figure 3.18. Assume that the owner of the *Prescribe-Medication* event body type specifies physicians to send events of that type.

   Furthermore, assume that the physician also specifies an access control rule that allows nurses to send events of this type. Note that this is a violation of the service autonomy concept, which prescribes that services must be in full control of their events.

   Whenever medication is prescribed, the prescribe medication event must be generated. According to the algorithm presented in Section 3.7.1, first the sender's own policy is checked and then the event body type's owner. If a nurse now attempts to generate the prescribe medication event, the physician's policy (which specified that she is allowed to do so) is never checked.

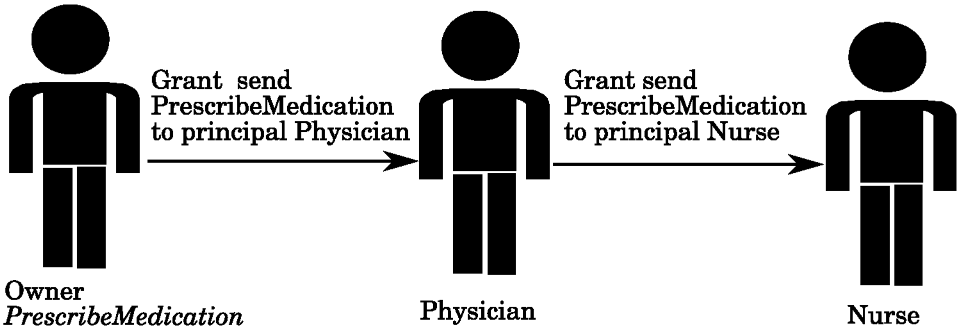


Figure 3.18: EFSOC Delegation

> The event body type's owner refuses access to anyone but physicians, which means that the nurses attempt will be blocked.

Summarizing: the way that EFSOC uses access control policies and the rules that belong to such policies supersedes the need to be able to delegate permissions by other mechanisms, such as role hierarchies.

3.11 Summary

Summarizing, EFSOC provides a discretionary role-based event-driven framework for access control in service-oriented architectures. The main components of the EFSOC model are *subject*, *event* and *role*. The relationships between subjects and roles, and the relationships between subjects and events are known as *operations*, which can be constrained by *access control policies*.

A typical service-invocation in EFSOC terminology starts when a subject generates an event, as shown in Figure 3.7. When the subject generates an event, a number of access control policies is evaluated to determine whether or not the event will be accepted. If it is accepted, EFSOC will first identify the subjects that are *subscribed* to events containing that event's body type. Next, a possibility is created to add additional business logic. How this is done, and of which elements the business logic consists is left outside the scope of this research. One the proper recipients have been determined, additional access control checks are performed to ensure that each of the recipients is allowed to receive the event. If those checks have been successfully completed, the events are delivered to each subject.

Chapter 4

EFSOC Definition and Execution Language

> The truth knocks on the door and you say, ”Go away, I’m looking for the truth,” and so it goes away. Puzzling.
>
> *Robert M. Pirsig, Zen and the Art of Motorcycle Maintenance*

Chapter 3 introduced EFSOC informally. We showed the different element of the framework and illustrated them with natural text and graphical representations in the form of UML diagrams. This chapter proposes a language that can be used to define and modify EFSOC elements, using a declarative language which implements the semantics of the EFSOC model as discussed in the previous chapter.

4.1 Extensible Markup Language

The EFSOC Definition and Constraint language (EDL) will be implemented as an XML application. XML (Yergeau et al., 2004) is a syntactic markup language that provides basic constructs to define domain-specific markup languages. Examples of such XML languages are XML Encryption (Reagle, 2002), XML Signatures (Bartel et al., 2002), XML Schema (Thompson et al., 2001), XML Namespaces (Bray et al., 1999), etc.

Because of its extensibility, the widespread availability of tools, and the fact that XML is widely accepted as a data representation format, the EFSOC DCL will be implemented in XML. To ensure interoperability with other XML languages, we introduce a special EFSOC XML Namespace.

4.2 The EDL language

The EFSOC Definition and Constraint Language (EDL) provides the constructs to define EFSOC vocabulary elements. EDL consists of two sub-languages: the EDL definition language and the EDL execution language. Each of the elements of the definition language can be encapsulated in execution language elements, which in

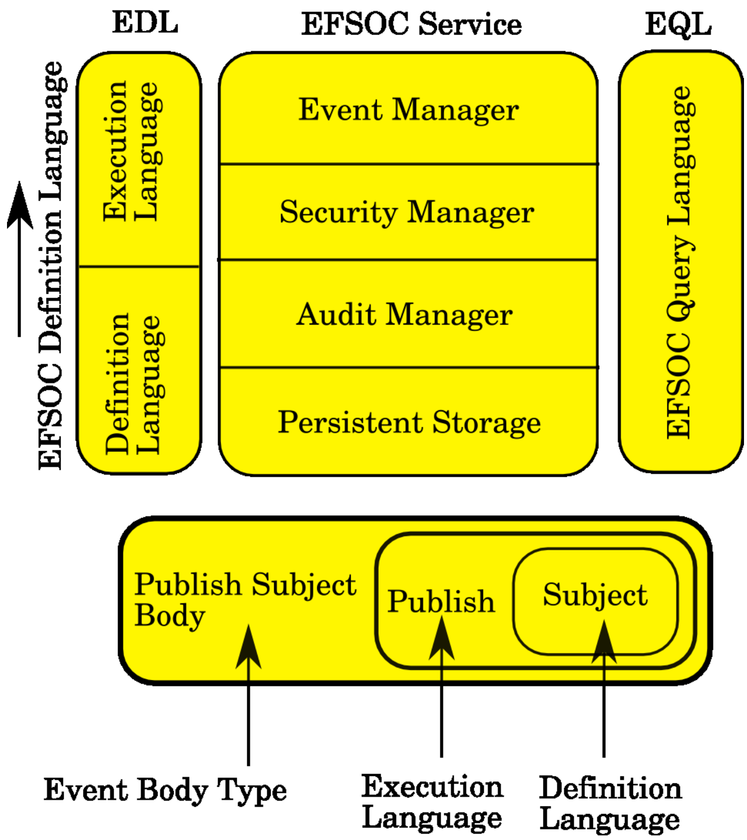

EDL
EFSOC Service
EQL
EFSOC Definition Language
Execution Language
Definition Language
Event Manager
Security Manager
Audit Manager
Persistent Storage
EFSOC Query Language
Publish Subject Body
Publish
Subject
Event Body Type
Execution Language
Definition Language

Figure 4.1: EFSOC Languages

turn form event body types. For example, to introduce a new subject to EFSOC, one would use a `subject` definition, encapsulated in a *publish* operation, which is associated with the `publishSubject` event body type. The EFSOC service will subscribe to all event body types that encapsulate execution language elements. This provide a natural way for services to interact with EFSOC.

4.3 EDL in relation to WSDL

EDL is complementary to WSDL. WSDL provides a description of service interfaces. EDL provides descriptions of meta information about those services. For example, using WSDL, it is possible to specify that the charting service has an operation *requestChart*, which in turn takes as input a patient's name, and returns the requested chart as output. Using EDL, we can describe who the patients are, who physicians are, and which patient is treated by which physician. Additionally, EDL provides a way to express subscriptions to input events and output events, access control rules, etc.

Using XML's namespacing mechanism, EDL and WSDL can be easily embedded in the same document.

4.4 Notational conventions

This specification uses several XML namespaces. The following prefixes are chosen arbitrarily and will be used:

Prefix	*Canonical name*	*Namespace URL*
xs	XML Schema	http://www.w3.org/2001/XMLSChema
efsoc	EFSOC DCL	http://infolab.uvt.nl/efsoc

4.5 Definition Language

4.5.1 Representing events

In Section 3.3, we defined an event as an atomic occurence in an organization. In Section 2.4, we discussed the work of Luckham, who distinguishes three distinct aspects of events: form, significance and relativity. We propose an event representation which captures and expresses these aspects. An event representation consists of an event envelope and an event body. The event body may contain any structured data type, which will represent the context in which the event occured. The event envelope contains a number of headers which provide meta-information about the event body, such as an event body type (which corresponds to the significance of the event), a possible time stamp, event id, and a number of additional headers which represent causal relatioships to other events.

Events are can be modeled as a composition of a single event body and a number of event headers. This representation allows us to easily map events to existing standards, such as SOAP.

Events can be represented in an XML-based representation as shown in Figure 4.2. The figure shows that an EFSOC event consists of the event headers and the event body. By grouping these two separate entities into elements, we provide easy support for adding privacy and integrity facilities later on. The figure also shows the use of XML namespaces to keep EFSOC defintions (the overall structure of an event message) separate from the body type and the data conveyed by the event.

4.6 Vocabulary Definitions

The EFSOC definition language takes the form of an XML extension language which uses XPath expressions. In the following sections, we propose a syntax for specifying EFSOC elements.

4.6.1 Subject

All subjects must be able to be referenced by a URI.

```
<efsoc:subject
    ID = xs:ID>
```

```
<?xml version=”1.0”>
<efsoc:event id=”some−id”
 xmlns:efsoc=”http://infolab.uvt.nl/efsoc”
 xmlns:loanex=”http://infolab.uvt.nl/efsoc/loanexample.xsd”>
 <efsoc:headers>
  <efsoc:timestamp>
   <efsoc:sent>12345678</efsoc:sent>
  </efsoc:timestamp>
 </efsoc:headers)
 <efsoc:body type=”patientadmitted”>
  <hospitalex:patient>
   <hospitalex:lastname>Smith</hospitalex:lastname>
   <hospitalex:insurance>Medicair</hospitalex:insurance>
  </hospitalex:patient>
  <hosptialex:admission>
   <hospitalex:date>2005−dec−04</hospitalex:date>
  </hospitalex:admisson>
 </efsoc:body>
</efsoc:event>
```

Figure 4.2: An XML event representation

```
   Content: (xs:any)
</efsoc:subject>
```

The ID attribute is a required attribute. Each subject may have additional information associated with it.

For example, the subject john is described as follows:

```
<efsoc:subject ID=”1234”>
   <efsoc:content>
       <persondata>
          <firstname>john</firstname>
          <lastname>smith</lastname>
          <born>1973−08−29</born>
       </persondata>
       <insurancedata>
          <type>medicair</type>
          <number>123456789</number>
       </insurancedata>
       <medicaldata>
          <medication>
             <name>zantac</name>
             <form>pill</form>
             <amount>40mg</amount>
             <usage>4x/day with meal</usage>
         </medication>
         <diagnonis>
```

```
                <name>acid reflux </name>
                <date >2005−08−21</date >
            <diagnosis >
          </medicaldata >
     </efsoc:content >
</efsoc:subject >
```

This description includes facts about john's person (name, date of birth), health insurance (medicair, policy number 123456789), and medical data on any medication that the patient is currently taking, and previously reached diagnoses.

4.6.2 Role

All roles must be able to be referenced by a URI.

```
<efsoc:role
    ID = xs:ID>
</efsoc:role >
```

The ID attribute is a required attribute. Each role may have additional information associated with it in the form of role attributes.

4.6.3 Role Attribute Type

All role attribute types must contain one or more role references. All role attributes must be able to be referenced by a URI

```
<efsoc:roleattributetype
    ID = xs:ID,
    roleref = xs:URI/>
```

The ID attribute is a required attribute.

4.6.4 Role Attribute Value

All role attribute values must reference a role attribute type and an role assingment. All role attribute values must be referencable by a URI.

```
<efsoc:roleattributevalue
    ID = xs:ID
    roleattributetyperef = xs:URI
    roleassingmentref = xs:URI
    value = xs:any>
</efsoc:roleattributevalue >
```

4.6.5 Event

Events consists of an event body which can contain any structured data respresentation which is part of event envelope, which also contains a number of headers.

An event is defined as follows

```
<efsoc:event
    ID = xs:ID>
    Content: ((efsoc:eventheader)*, efsoc:eventbody)
</efsoc:event>
```

4.6.6 Event Header

An event header conveys some kind of meta-information about the event body that is being sent. An event header is defined as

```
<efsoc:eventheader
    name = xs:String>
    Content: (xs:any)
</efsoc:eventheader>
```

Each event header has a required name and a structured content. Event headers may repeat, although it is likely that more expressiveness is obtained by using an adequately defined content.

The following event headers are minimally supported:

1.
   ```
   <efsoc:eventheader name="causality">
       #123
   </efsoc:eventheader>
   ```

   The causality header contains references to events that caused the current event to be sent. If the cardinality header is omitted, the current event does not have any causal relationships. If the cardinality header is repeated multiple times, the current event is sent as a result of the combination all events listed.

2.
   ```
   <efsoc:eventheader name="timestamp-sent">
     2006-02-15 14:36:45.134+02:00
   </efsoc:eventheader>
   ```

   Contains a timestamp which signifies the time that the current event was originally sent. This header must contain a single valid date/time value according to RFC3339 (Klyne and Newman, 2002).

4.6.7 Event Body

While an event body type defines the name and the structure of future event bodies, event bodies themselves contain instantiated data. An event body is defined as

```
<efsoc:eventbody
    type = xs:anyURI>
    Content: (xs:any)
</efsoc:eventbody>
```

The type attribute is a required attribute. An event body must always reference its event body type. The event body content must match the structure of the references event body type.

4.6.8 Event Body Type

Event bodys define the structure of events. An event body can take any data structure. It is defined as follows:

```
<efsoc:eventbodytype
      ID = xs:ID>
      Content: (xs:schema)
</efsoc:eventbody>
```

The ID attribute is a required attribute. Each event body must be uniqly identifyable by an ID and be referencable via a URI.

4.6.9 Access Control Policies

Access control rules function as groupings of access control rules. Access control policies are defined as follows:

```
<efsoc:accesscontrolpolicy
    ID = xs:ID
    ownerref = xs:anyURI>
    Content: (defaultpermission: PermissionList)
</efsoc:accesscontrolpolicy>
```

The ID attribute and the owner attribute are required attribute. Each access control policy must have a unique ID and must have an owner which is a URI reference to a subject.

4.7 Execution language definitions

4.7.1 Publish

Events, subjects, roles, access control policies or access control rules can be published by using the `publish` operator. The publish operator expects an event body, a role, a subject, an access control policy or an access control rule as input. Publish is defined as

```
<efsoc:publish>
    Content: (efsoc:subjectref,
                (efsoc:eventbodytype | efsoc:role | efsoc:subject |
                 efsoc:accesscontrolpolicy |
                 efsoc:accesscontrolrule
                )+
             )
</efsoc:publish>
```

The publish operator takes as parameters a single URI identifying a subject and a list of at least one event body type, role or subject that must be published. The `publish` operator must function as a single transaction. In other words, all event body types must be published successfully, or none may be published.

The definition of `eventbodytype` and an example showing how to use the `publish` operator is shown in Section 4.5.1.

For example, publication of the ‘patient admitted’ event would looks like

```
<efsoc:publish>
  <efsoc:subjectref>
     http://efsoc.infolab.uvt.nl/subject#1234
   </efsoc:subjectref>
  <efsoc:eventbodytype >....</efsoc:eventbodytype>
</efsoc:publish>
```

4.7.2 Unpublish

The `unpublish` operator works similarly to the `publish` operator. `Unpublish` is defined as follows

```
<efsoc:unpublish>
    Content: (efsoc:subjectref ,
               (efsoc:eventbodytyperef | efsoc:roleref |
                efsoc:subjectref |
                efsoc:accesscontrolpolicyref |
                efsoc:accesscontrolruleref
               )+
             )
</efsoc:unpublish>
```

All parameters are URI references to their corresponding elements. An unpublish operation must be considered as a transaction. In other words, if one unpublish operation fails, all unpublish operations must fail.

For example, unpublication of the ‘patient admitted’ event looks like

```
<efsoc:unpublish>
   <efsoc:eventbodytyperef>
       http://efsoc.infolab.uvt.nl/eventbodytype/#1234
    </efsoc:eventbodytyperef>
</efsoc:unpublish>
```

4.7.3 Send

Events can be generated (sent) by using the `send` operator. The generate operator expects an event as input. Generate is defined as

```
<efsoc:send>
    Content: (efsoc:subjectref , (efsoc:event)+)
</efsoc:send>
```

The generate operator takes as parameter a single URI identifying a subject and a list of at least one event that must be generated. The `send` operator must function as a single transaction. In other words, if multiple events are listed, all of them must be generated successfully, or none may be generated at all.

For example, to represent the fact that a ‘patient admitted’ event is generated by the subject with ID 1234, the following representation would be used

```
<efsoc:send>
  <efsoc:subjectref>
     http://efsoc.infolab.uvt.nl/subject#1234
   </efsoc:subjectref>
  <efsoc:event >..... </efsoc:event>
</efsoc:send>
```

4.7.4 Subscribe

Subjects may subscribe to events by using the subscribe operator. The subscribe operator is defined as

```
<efsoc:subscribe>
    Content: (efsoc:subjectref , (efsoc:eventbodytyperef)+)
</efsoc:subscribe>
```

Each subject must be referenced by a single unique uniform resource indicator (URI). Each event body type must be referenced by a URI. In a single subscribe operation, a subject may subscribe to one or more different event body types by listing their URIs. If a subscribe statement contains multiple event body URIs, the subscription must be considered a transaction. In other words, all subscriptions must be successful for the operation to succeed. In case of failure, the subject must not be subscribed to any additional URIs.

For example, to represent the fact that subject #1234 subscribes to the ‘patient admission event’ (event #1234), the following fragment can be used

```
<efsoc:subscribe>
  <efsoc:subjectref>
     http://efsoc.infolab.uvt.nl/subject#1234
   </efsoc:subjectref>
  <efsoc:eventbodytyperef>
      http://efsoc.infolab.uvt.nl/eventbodytype/#1234
   </efsoc:eventbodytyperef>
</efsoc:subscribe>
```

4.7.5 Unsubscribe

The `unsubscribe` operator works similarly to the `subscribe` operator. Parameters and their meaning are identical. `Unsubscribe` is defined as follows

```
<efsoc:unsubscribe>
```

```
   Content: (efsoc:subjectref , (efsoc:eventbodytyperef)+)
</efsoc:unsubscribe>
```

4.7.6 Assign

The `assign` operator is used to link subjects to roles. The assign operator is defined as

```
<efsoc:assign>
    Content: (efsoc:subjectref+, efsoc:roleref+)
</efsoc:assign>
```

The assign operator takes one or more URIs as a reference to a subject as an attribute and any number of URI references to roles. In a single assign operation, a subject may be assigned to multiple roles or multiple subjects may be assigned to multiple roles. The assign operator must function as a transaction. In other words, if one assignment fails, no assignments may take place at all.

For example, to represent that subject #1234 is assigned the 'patient admission service' role (#101) as well as the 'hospital service' role (#102), the following XML can be used.

```
<efsoc:assign>
  <efsoc:subjectref>
    http://efsoc.infolab.uvt.nl/subject#1234
  </efsoc:subjectref>
  <efsoc:roleref>
    http://efsoc.infolab.uvt.nl/role#101
  </efsoc:roleref>
  <efsoc:roleref>
    http://efsoc.infolab.uvt.nl/role#102
  </efsoc:roleref>
</efsoc:assign>
```

4.7.7 Unassign

The `unassign` operator is used to link subjects to roles. The unassign operator is defined as

```
<efsoc:unassign>
    Content: (efsoc:subjectref , (efsoc:roleref)+)
</efsoc:unassign>
```

The unassign operator takes one URI as a reference to a subject as an attribute and any number of URI references to roles. In a single unassign operation, a subject may be unassigned from multiple roles. The unassign operator must function as a transaction. In other words, if one unassignment fails, no unassignments may take place at all.

4.7.8 Activate

The `activate` operator is activate roles for subjects. The activate operator is defined as

```
<efsoc:activate>
    Content: (efsoc:subjectref, (efsoc:roleref)+)
</efsoc:activate>
```

The activate operator takes one URI as a reference to a subject as an attribute and any number of URI references to roles. In a single activate operation, a subject may activate multiple roles. The activate operator must function as a transaction. In other words, if one activation fails, no activations may take place at all.

4.7.9 Deactivate

The `deactivate` operator is used to deactivate roles for subject. The deactivate operator is defined as

```
<efsoc:deactivate>
    Content: (efsoc:subjectref, (efsoc:roleref)+)
</efsoc:deactivate>
```

The deactivate operator takes one URI as a reference to a subject as an attribute and any number of URI references to roles. In a single deactivate operation, a subject may deactivate multiple roles. The deactivate operator must function as a transaction. In other words, if one deactivation fails, no deactivations may take place at all.

4.7.10 Set

The `set` operator is used to set role attribute values. The operator is defined as

```
<efsoc:set>
   Content: (efsoc:roleref, efsoc: roleattributeref,
             efsoc:value)
</efsoc:set>
```

4.8 Access Control Rules

Access control rules take the form

```
<efsoc:accesscontrolrule
  ID = xs:ID
  priority = xs:Integer
  operation = OperationList
  permission = PermissionList
  Content: (efsoc:policyref+, efsoc:principal,
efsoc:condition)
</efsoc:accesscontrolrule>
```

ID, operation and permission are required attributes. Priority will default to 1. The policyref parameter may be repeated and refers by URI to the access control policy to which this rule belongs.

4.8.1 Principal

Principals are either subjects or roles. The principal tag takes the form

```
<efsoc:principal>
     (efsoc:roleref | efsoc:subjectref)+
</efsoc:principal>
```

When multiple principals are specified, the logical union is assumed.

4.8.2 Permission

PermissionList is defined as follows:

```
PermissionList ::=
      "permit"
    | "reject"
```

4.8.3 Operation

OperationList is defined as follows:

```
OperationList ::=
      "publish"
    | "unpublish"
    | "subscribe"
    | "unsubscribe"
    | "send"
    | "assign"
    | "unassign"
    | "activate"
    | "deactivate"
    | "set"
```

This list corresponds to the infrastructure operations that were discussed in the previous chapter.

4.8.4 Condition

In the previous chapter, conditions were also introduced as consisting of a principal and an expression. We define a condition as:

```
<efsoc:condition>
    Content: (xs:any)
</efsoc:condition>
```

Conditions which must be met can be specified on the three different layers: transport layer, message layer and message-context layer. To this end, the EFSOC access control language provides a number of expressions which may be used to express those conditions.

Conditions are expressed in the form of XPath statements which should return a reference to the event body, if they match the event that is evaluated, or nothing, if they do not.

4.8.5 Transport level expressions

Source IP address

To match the source IP address of the requester, a condition may inspect the ’origin-ip’ header.

For example, to validate that the event was sent by a subject at IP address 137.56.127.213, the following expression would be used:

```
/event/context/event[
   eventheader[@name=’sender-ip’]  =’137.56.217.213’
]
```

4.8.6 Message level expressions

Message level expressions are expressed according to XPath specifications. For example, assume that *Message* takes the format:

```
<efsoc:event>
   <efsoc:eventbody type="prescribemedication">
      <patient>sue</patient>
      <drug>Ibuprofin</drug>
      <dosage>800 mg, every 8 hours</dosage>
      <numberofdoses>15</numberofdoses>
   </efsoc:eventbody>
</efsoc:event>
```

To require that medication for Ibuprofin may only be prescribed in quantities smaller dan 10 doses, the following expression may be used.

```
/event[eventbody[@eventbodytype=’prescribemedication’]
/numberofdoses < 10]
```

Note that EFSOC will not be able to enforce any message-level access control rules if the event body is encrypted with a key that is not known to the EFSOC broker. While EFSOC is able to enforce message-level access controls, in most cases, the services involved in the message exchange will have to perform additional checking themselves.

4.8.7 Message-context level expressions

As mentioned before, access control rules may contain conditions which apply to temporal and causal relations. Causality is defined as follows: event $e2$ is said to be caused by event $e1$ when the following conditions are met:

1. Event $e1$ was received;
2. Event $e2$ is sent at a point in time *after* $e1$ was received;
3. The receiver of $e1$ and the sender of $e2$ are the same;
4. The sender of $e2$ *explicitly* states that it is sent in response to $e1$.

EFSOC allows the following message-context level expressions:

Testing for causality

Causality is captured in event header *causality*. To test for a causal relationship with other events, the following XPath expression may be used.

```
/event[eventheader[@name='causality'] = 'e1']
```

Testing for event sequence

To test for a temporal sequence in events, the ‘timestamp-sent’ header may be inspected. For example, to express the requirement that an event must be sent after a ‘prescribemedication’ event, the following expression may be used:

```
/event[eventheader[@name='timestamp-sent'] >
    /event[eventbody[@eventbodytype='prescribemedication']]
    /eventheader[@name='timestamp-sent' ]
]
```

Testing for chronological sequence

To test for chronological sequencing, i.e., if an event was sent before or after a time, or in a time interval, the timestamp-sent header can be inspected directly.

For example, to test if an event was sent after a certain time, the following XPath expression may be used.

```
/event[eventheader[@name='timestamp-sent'] >
   '2006-02-20 12:01:01+0100']
```

4.8.8 Misc expressions

1. *Testing for roles*

   To test if a

   ```
   /event[eventheader[@name='sender'] =
     /assign[subjectref=/accesscontrolrule/principal/subjectref]
     or /assign[roleref=/accesscontrolrule/principal/roleref]
   ]
   ```

4.8.9 Combining conditions

Expressions may also be combined to form more complicated requirements. For example, a condition which states that an event may only be sent on Mondays between 3pm and 4pm and after a ‘Prescribe Medication’-event has been received, the following expression may be used.

```
/event[eventheader[@name='timestamp-sent'] > 'Monday, 3pm'
    and eventheader[@name='timestamp-sent'] < 'Monday, 4pm'
    and eventheader[@name='timestamp-sent'] >
       /event[eventbody[@eventbodytype='prescribemedication']]
       /eventheader[@name='timestamp-sent']
]
```

4.9 Summary

The EFSOC definition and execution language consists of two separate sub-languages, which may be nested. The definition language provide a means to describe elements, such as subjects, roles or event body types. Definition elements may be contained in execution language elements, which allows the creating, removal or modification of definition elements. Finally, definition elements may be contained in event body types that will be pre-registered in EFSOC implementations. A full reference of EDL can be found in Appendix III.

Chapter 5

EFSOC Query Language

In the previous chapters, we introduced the EFSOC Definition and Constraint Language (DCL). The EFSOC DCL is XML-based and provides a number of contructs for defining elements using the EFSOC vocabulary.

In addition to having the ability to define EFSOC instances, it is often useful to be able to query the state of the full model.

In this chapter, we investigate querying XML-based repositories, and we introduce the EFSOC Query Language (EQL).

Throughout the chapter, we will be referring to a simple example, as illustrated in Figure 5.1. The example contains a small fragment of EFSOC DCL in which two subjects (John and Mary) and one role (Physician) are defined. John and Mary are both asssigned to the Physician role. Mary has sent an 'Prescribe Medication' event at some point in time.

5.1 Querying XML

Since the EFSOC-DCL is XML-based, it is an obvious choice to use XML-based tools to define a *query language*. The query language may be used to inspect the state of the model, for a variety of reasons. For example, using a query, a service provider who wants to deploy a new composite service will be able to check if his new service will be able to execute its operation given the current access control constraints.

5.1.1 XPath

While—strictly speaking—XPath is not a query language, most XML querying relies heavily on it. XPath is a language that can be used to locate information in XML documents. As such, it is possible to formulate XPath expressions that function as queries, however that is not where its strengths lay.

Consider the EFSOC definitions of Figure 5.1. It is fairly straightforward to extract simple information from that example. For example, using the XPath expression `/efsoc/event[eventbody[`

```
<efsoc>
  <subject ID=”mary”/>
  <subject ID=”john”/>
  <role ID=”physician”/>

  <assign>
     <subjectref>mary</subjectref>
     <roleref>physician</roleref>
  </assign>
  <assign>
     <subjectref>john</subjectref>
     <roleref>physician</roleref>
  </assign>

  <event ID=”e1”>
     <eventheader name=”sender−ip”>137.56.217.213</eventheader>
     <eventheader name=”sender”>mary</eventheader>
     <eventheader name=”timestamp−sent”>99</eventheader>
     <eventbody eventbodytype=”prescribemedication”>
        <patient>sue</patient>
        <drug>Ibuprofin</drug>
        <dosage>300 mg, 3 times per day</dosage>
        <numberofdoses>15</numberofdoses>
     </eventbody>
  </event>

  <event ID=”e2”>
     <eventheader name=”timestamp−sent”>100</eventheader>
     <eventheader name=”causality”>e1</eventheader>
  </event>

  <event ID=”e3”/>
</efsoc>
```

Figure 5.1: Sample definitions

`@eventbodytype='prescribemedication']]` will return the full event representation of all events with event body type ‘prescribe medication’.

It is also possible to crete more complicated XPath expressions, such as illustrated in the previous chapter. However, the disadvantage of such specifications are that they become very complicated quickly, and are hard to write, or understand.

Instead, we propose to implement a number of functions in the XQuery language, which will shield users from complex expressions and allow them to compose fairly complex statements easier.

For example, assume that we want to represent all subjects whole are assigned the physician role and who have sent *prescribe medication* events. In the form of XPath expressions, this would look like

1. An expression to locate all subject who are member of the physician role:

   ```
   /efsoc/subject[@ID=
        /efsoc/assign[roleref='physician']/subjectref
   ```

2. An expression to locate all subjects who sent *prescribe medication* events:

   ```
   /efsoc/event[
        eventbody[@eventbodytype='prescribemedication']
   ]/eventheader[@name='sender']
   ```

3. An expression which combines the above

   ```
   /efsoc/subject[
      @ID=/efsoc/assign[roleref='physician']/subjectref and
      @ID=/efsoc/event[
          eventbody[@eventbodytype='prescribemedication']
       ]/eventheader[@name='sender']]
   ```

As can illustrated, the expression become complicated (and prone to error) rapidly.

5.1.2 XQuery

XQuery (XQuery, 2005) is a W3C proposed standard that was designed specifically for querying XML documents. Unlike many of the X-specification, XQuery (like XPath) is not an XML language itself. XQuery does rely on XPath expressions to nagivate through document content, and both languages share the same data model.

The XQuery language is often compared with the structured query language (SQL) that is used to extract information from relational database. XQuery’s structure is simple and can be summarized by the acronym FLWOR, which standads for For-Let-Where-Order-Return.

1. The *for* statement can be used to iterate over nodes in a node set. For example, the following statement lists the identifiers of all subject elements in the file `efsoc.xml`.

```
for $x in doc(”efsoc.xml”)/efsoc/subject
return $x/@ID
```

2. The *where* statement can be used to restrict the nodes in a node set. For example, the following statement lists all subjects in the file `efsoc.xml` who sent events from a specific IP address.

   ```
   for $x in doc(”efsoc.xml”)/efsoc/event
   where $x/eventheader[@name=’sender−ip’] = ”137.56.127.213”
   return $x/eventheader[@name=’sender’]
   ```

3. The *let* statement can be used to define a variable and add a value to it. For example, the statement listed above can also be expressed as:

   ```
   for $x in doc(”efsoc.xml”)/efsoc/event
   let $ip=”137.56.127.213”
   where $x/eventheader[@name=’sender−ip’] = $ip
   return $x/eventheader[@name=’sender’]
   ```

4. The *order by* can be used to sort the output. For example, to sort the output of the previous statement chronologically, the following code can be used:

   ```
   for $x in doc(”efsoc.xml”)/efsoc/event
   let $ip=”137.56.127.213”
   where $x/eventheader[@name=’sender−ip’] = $ip
   order by $x/eventheader[@name=’timestamp−sent’]
   return $x/eventheader[@name=’sender’]
   ```

Statements listed as above can also be encapsulated in user-defined functions. For example, to define a function that produces the same output as above, but parameterizes the required IP address, the following code can be used

```
declare function sendersByIP ($ip) {
  for $x in doc(”efsoc.xml”)/efsoc/event
  where $x/eventheader[@name=’sender−ip’] = $ip
  order by $x/eventheader[@name=’timestamp−sent’]
  return $x/eventheader[@name=’sender’]
};
```

The same query shown that was shown in the previous section can also be made using XQuery, as illustrated below.

1. Define a function to retrieve subjects playing a certain role:

   ```
   declare function local:subjectsWithRole($role) {
     for $x in doc(”example.xml”)/efsoc/assign
      where $x/roleref=$role
      return $x/subjectref/text()
   };
   ```

2. Define a function to retrieve events of a particular type:

```
declare function local:EventsOfType($type) {
  for $x in doc("example.xml")/efsoc/event
  where $x/eventbody/@eventbodytype = $type
  return $x
};
```

3. Define a function that retrieves subjects by ID:

```
declare function local:subjectByName($name) {
  for $x in doc("example.xml")/efsoc/subject
  where $x/@ID = $name
  return $x
};
```

4. Define a function that retrieves the ID of the sender of a particular event:

```
declare function local:senders($event) {
  for $x in local:EventsOfType($event)
  return $x/eventheader[@name="sender"]/text()
};
```

5. An expression which combines the above

```
<out>
  {
    let $sender := local:subjectByName(
                     local:senders("prescribemedication")),
        $physicians := local:subjectByName(
                         local:subjectsWithRole("physician"))

    return $sender intersect $physicians
  }
</out>
```

As can be seen, the final expression above is easier to understand than the one using plain XPath expressions.

5.1.3 XSL

The XSL specification is an XML-language used for transforming XML documents from one structure to another. XSL is commonly used to transform XML documents into (X)HTML documents that can be shown in a user's web browser. However, XSL can also be used to filter XML content, which means that it is usuable as a query language.

In the example shown in Figure 5.1, we showed an EFSOC DCL fragment which we subsequently queried using XPath, respectively using XQuery. The same query can be done using XSL as follows.

```
<xsl:transform version=”1.0”
      xmlns:xsl = ”http://www.w3.org/1999/XSL/Transform”>

<xsl:variable
   name=”physicians”
   select=”/efsoc/assign[roleref=’physician ’]/subjectref”/>
<xsl:variable
   name=”senders”
   select=”/efsoc/event[eventbody[
              @eventbodytype=’prescribemedication ’]]/
              eventheader[@name=’sender ’]”/>

<xsl:template match=”/”>
    <xsl:for−each select=”$physicians”>
       <xsl:apply−templates select=”current()”/>
    </xsl:for−each>
</xsl:template>

<xsl:template match=”subjectref”>
   <xsl:variable name=”s”>
      <xsl:value−of select=”current()”/>
   </xsl:variable>
   <xsl:for−each select=”$senders”>
      <xsl:if test=”current()=$s”>
         <xsl:value−of select=”current()”/>
      </xsl:if>
   </xsl:for−each>
</xsl:template>

</xsl:transform>
```

The code example begins by defining a variable `physicians`, which will be assigned the value of the evaluated XPath expression listed in the `select` attribute. The same happens for the variable `senders`. Next, all physicians are checked to see if they are also in the senders. If that is the true, they become part of the output.

Like the XQuery expression, the listing shown above is preferred over a plain XPath expression. However, the readablity and level of understanding of both leaves to be desired.

5.2 EFSOC Query Language Overview

In the previous chapter, we analyzed three query language for use with XML data: XPath, XQuery and XSL. Of all three languages, XQuery provides the most expressive querying capabilities and offers methods for encapsulating functionality via user-defined functions.

For these reasons, we will define a number of XQuery functions that can be used to inspect the state of a model. These functions can be divided into two categories:

the first category provides basic queries to retrieve information defined in EDL. The second category of functions combines functions of the first category to provide answers to more complicated queries.

5.2.1 Basic queries

The following basic queries are defined. The `storage()` expression in the following XPath functions provides access to the persistent storage manager containing all EFSOC definitions.

1. `function local:subjectByID($ID)`
   Return the subject with the given ID.

```
declare function local:subjectByID($ID) {
  for $x in storage()/efsoc/subject
  where $x/@ID = $ID
  return $x
};
```

2. `function local:roleByID($ID)`
   Returns the role with the given ID.

```
declare function local:roleByID($ID) {
   for $x in storage()/efsoc/role
   where $x/@ID = $ID
   return $x
};
```

3. `function local:eventTypeByID($ID)`
   Returns the event body type with the given ID.

```
declare function local:eventTypeByID($ID) {
   for $x in storage()/efsoc/eventbodytype
   where $x/@ID = $ID
   return $x
};
```

4. `function local:eventByID($ID)`
   Returns the event with the given ID.

```
declare function local:eventByID($ID) {
   for $x in storage()/efsoc/event
   where $x/@ID = $ID
   return $x
};
```

5. `function local:eventsByType($type)`
   Returns all events of the given type that have been sent.

```
declare function local:eventsByType($type) {
  for $x in storage()/efsoc/event
  where $x/eventbody/@eventbodytype = $type
  return $x
};
```

6. `function local:subjectsByRole($role)`
   Returns all subjects who are currently *assigned* to the given role.

```
declare function local:subjectsByRole($role) {
  for $x in storage()/efsoc/assign
  where $x/roleref=$role
  return $x/subjectref/text()
};
```

7. `function local:rolesBySubject($subject)`
   Returns all roles that are currently *assigned* to the given subject. The subject is references by ID.

```
declare function local:rolesBySubject($subject) {
  for $x in storage()/efsoc/assign
  where $x/subjectref=$subject
  return $x/roleref/text()
};
```

8. `function local:subjectsBySubscription($type)`
   Returns identifiers of subjects who are subscribed to the given event body type.

```
declare function local:subjectsBySubscription($type) {
   for $x in storage()/efsoc/subscribe
   where $x/eventbodyref = $type
   return $x
};
```

9. `function local:eventsBySubscriber($subject)`
   Returns identifiers of all event body types that the given subject is subscribed to.

```
declare function local:eventsBySubscriber($subject) {
   for $x in storage()/efsoc/subscribe
   where $x/subjectref = $subject
   return $x
};
```

10. `function local:roleWithAttribute($roleattribute)`
    Returns identifiers of all roles that have a role attribute with the given name.

```
declare function local:rolesWithAttribute($attribute) {
  for $x in storage()/efsoc/roleattributetype
  where $x/@ID=$attribute
  return $x
}
```

11. `function local:activeRolesOfSubject($subject)`
    Returns the role identifiers that the specified subject currently has active.

```
declare function local:activeRolesOfSubject($subject) {
  for $x in storage()/efsoc/activate
  where $x/subjectref = $subject
  return $x
}
```

5.2.2 Second-order queries

The following second-order queries are defined. Some examples of such queries are enumerated below:

1. `function sendersOfEventType($type)`
   Returns subject ID’s of all subjects who have sent events of the given type.

```
declare function local:sendersOfEventType($type) {
  for $x in eventsByType($type)
  return $x/eventheader[@type='sender']
}
```

2. `function local:subjectWithAttributeValue($roleattribute, $value)`
   Returns identifiers of all subjects who have a role with the given role attribute and the value of that role attribute is equal to the specified value.

```
declare function
  local:subjectsWithAttributeValue($roleattribute, $value) {
  let $y := storage()/efsoc/roleattributevalue[
    @roleattributetyperef=$roleattribute and
    @value=$value]/roleassignmentref
  for $a in storage()/efsoc/assign
  for $b in $y
  where $a/@ID = $b
  return $a/subjectref
};
```

5.3 Summary

In this chapter, we evaluated three languages that can be used to query XML data. XPath is a language that can be used to locate data in XML documents. Strictly speaking, this means that it is not a query language per se, but it can be used as such. While XPath expressions tend to quickly become large and complex, they are useful for smaller lookups. In addition, XPath is used heavily in XQuery.

XQuery is a W3C proposed standard that was designed specifically for querying XML documents. Unlike many of the X-specification, XQuery (like XPath) is not an XML language. XQuery's structure is simple and can be summarized by the acronym FLWOR, which standads for For-Let-Where-Order-Return.

The third language that was used is XSL. The XSL specification is an XML-language used for transforming XML documents from one structure to another. XSL is commonly used to transform XML documents into (X)HTML documents that can be shown in a user's web browser. However, XSL can also be used to filter XML content, which means that it is usuable as a query language.

After evaluating of the three languages, we chose to implement the EFSOC Query Language (EQL) as a set of XQuery functions. The functions that are provided can easily be extended by defining additional ones, which may use pre-existing queries.

Part II

Validation

Chapter 6

Formal Foundations

> Entia non sunt multiplicanda praeter necessitatem
>
> *William of Ockham, c. 1288–1348*

This chapter aims to provide a formal analysis of the EFSOC framework. By including this analysis, we illustrate that the underlaying framework is *operationally consistent* and that it can be implemented into a working software solution. As a result, we will show that the framework does not violate its own integrity constraints and that the framework can indeed be instantiated.

In this section, the first step is to define a basic *vocabulary*, which may be used to express *facts* and *axioms*. Next, we will describe access control rules and access control policies and their relationship to the vocabulary.

6.1 Reasons for formalizing

Unfortunately, all too often, a formalism is viewed as a goal in its own right. We disagree with this view, and are convinced that a formalism is only useful when it addresses a specific goal that is known in advance. There are several good reasons to formalize an approach, a framework or a model.

1. Our everyday use of language is vague, and our everyday level of thinking is often muddled (Suppes, 1957). Logic formalisms can play a role in clarifying certain types of descriptive language, most notably concepts and relations between concepts. This is most commonly achieved by providing a small set of language constructs that have an agreed upon meaning, and ways to combine those constructs.

   Formalizing can be used as a means to explain concepts and their relationships using precise semantics.

2. Since a formalism provides a description of concepts and relationships between concepts, it is possible to express separate 'systems' using a common vocabulary. This makes it possible to compare the systems and make exact statements about properties of thoe systems.

Formalizing can be used as a means to compare different 'systems'.

3. Many formalisms offer a way to create prove. By combining facts with axioms (A self-evident and necessary truth, or a proposition whose truth is so evident as first sight that no reasoning or demonstration can make it plainer), propositions and deductive rules, it can be deduced that facts are in compliance with axioms and rules.

   Formalizing can be used as a means to proof that a set of facts is consistent with a set of axioms and deductive rules.

In this thesis, we will use the process of formalization in the first role: as a means to explain concepts and their relationships using precise semantics. In other words, we will express the EFSOC framework in logic in order to explain in a common language of which elements it consists and how elements are related. The formalized representation of the EFSOC framework presented in this chapter is intended for a human reader. As such, representations that are primarily intended for automated processing, such as the Web Ontology Language (McGuinness and van Harmelen, 2004), are not considered.

Secondly, we will use the framework defined in the formalism to show that the results of a case study, expressed in terms of the framework, are logically consistent with the rules and constraints put forward by the EFSOC approach.

6.2 Approach

In any formalistic approach, choosing the correct formalism is a very difficult problem, as well as an important decision. To be able to make the right choice, we require a formalism to have the following properties:

1. The ability to express facts;
2. The ability to express deductive rules;
3. The ability to express constraints;
4. The ability to express queries;
5. Support for complex data types;
6. The ability to represent information at multiple levels of abstraction

After an initial survey, the shortlist for formalisms that were considered consists of predicate logic, Datalog and Telos. Each of these formalisms will be discussed briefly and evaluated against the criteria listed above.

6.2.1 Predicate Logic

Predicate logic is an extension of propositional logic, which can be used to express facts and assigns truth values to combinations of facts, and extends it with the ability to express quantified statements. In particular, predicate logic introduces the existential quantifier $\exists$, which is pronounced ‘for some’ or ‘there exists at least one’ and the universal quantifier $\forall$, which is pronounced ‘for all’.

Predicate logic, or rather, propositional logic, is extremely well suited for expressing facts, and for combining facts with logic operators to arrive at sentences which may (or may not) be true. For example, we introduce the predicate $\mathsf{subject}(a)$ to represent that a is a subject and the predicate $\mathsf{role}(b)$ to represent that b a role. Furthermore, assume that the predicate $\mathsf{assigned}(a, b)$ represents the fact that a is assigned role b.

Using propositional logic, it can now be expressed that ‘john’ is a subject, ‘physician’ is a role and that John plays the role of physician:

$$(\mathsf{subject} = \{john\}, \mathsf{role} = \{physician\}, \mathsf{assigned} = \{(john, physician)\})$$

Deductive rules can be specified with the same amount of ease. To state that a relation a is true if b or c are true, one would simply write

$$a \leftarrow b \vee c$$

Like specifying deductive rules, constraints can be interpreted as rules that must always be true. To express the constraint that all subjects must be assigned a role, one would write

$$\forall s\ \mathsf{subject}(s) \rightarrow \exists r\ \mathsf{role}(r)\ \mathsf{assigned}(s, r)$$

When predicate logic is defined as full clausal logic, it is also possible to specify variables. Using variables and predicate logic, it becomes possible to specify queries. For example, to specify a query that retrieves the subject who play the role physician, one could write

$$\mathsf{physician}(X) \leftarrow \mathsf{subject}(X) \wedge \mathsf{assigned}(X, physician)$$

Note that the direction of the implication arrow is reversed from when specifying a constraint.

While we have shown that predicate logic can be used to express facts, rules, constraints and queries, predicate logic is not intuitive for expressing complex data types. For example, to express that subject john was born on the first of January, 1970, that his username is ”john” and that his password is ”secret”, we would need to introduce the following data

($\mathsf{username}$ = $\{”john”\}$, $\mathsf{password}$ = $\{”secret”\}$, $\mathsf{birthdate}$ = $\{$”1970-01-01”$\}$, $\mathsf{subjectusers}$ = $\{(john,\ ”john”)\}$, $\mathsf{subjectpassword}$ = $\{(john,\ ”secret”)\}$, $\mathsf{subjectbirthdate}$ = $\{(john, ”1970-01-01”)\}$)

While it is *possible* to express complex data using predicate logic, it is not a convenient formalism to do so.

In addition, (first order) predicate logic is not suitable for combining multiple layers of abstration in a single model.

6.2.2 Datalog

Datalog is a logic-based data model which is loosely based on the Prolog programming language. The underlaying mathematical model of data for datalog is essentially that of the relational model. Predicate symbols in datalog denote relations. Datalog supports two types of relations: existential database relations and sintensional database relations. Existential database relations (EDBs) are relations that are reflected directly in the database of facts, while intentional database relations (IDBs) are only defined by logical rules (Ullman, 1988).

The above excerpt implies that Datalog provides support for representing facts (through a database), and rules to deduce new facts from known ones.

Since Datalog is essentially built on the relational model, it is—at least partially— possible to store complex data types. The restriction in complexity is caused by the fact that the relational model does not allow tables to contain nested structures, like it is common in for example XML databases.

Datalog supports the formulation of queries via rules that derive predicates in a bottom-up fashion. For example, the query to retrieve which subjects play the role of physician can be represented as follows:

```
physician(X) :- subject(X), assigned(X, physician)
```

Datalog allows for the specification of constraints using the same rule system. By specifying the consequent of the rule as ‘inconsistent’, the antecedent must remain false. For example, to specify that the physician role and the patient role cannot be assigned to the same subject, the following representation can be used.

```
inconsistent :- subject(X), assigned(X, physician),
assigned(X, patient)
```

Unfortunately, like predicate logic, Datalog is not able to natively specify information at different levels of abstraction.

6.2.3 Telos

Telos provides facilities for constructing, querying and updating structured knowledge bases. It includes a first order assertion sublanguage as means of specifying integrity constraints and deductive rules (Mylopoulos et al., 1990).

The Telos knowledge base may contain structured objects called *propositions* that can either be *individuals* or *attributes*. An individual is a concrete entity, such as a subject, or `john`, while an attribute represents a binary relationship between entities or other relationships.

Telos offers the additional benefit that it is able to express a model that is layered; in other words, Telos understands the concepts of instances, models and meta models. A proposition that has no instances represents concrete entities in the domain of discourse and is called a *token*. Simple classes are tokens that only have tokens as instances, meta-classes are tokens that only have simple classes as instances, etc.

Every attribute consists of a source, a label and a destination and can be represented by a three-tuple $[$ source, label, to $]$. Consider the previous example. To express that John is a subject, physician a role and that john plays the role of physician, the following Telos representation would be used:

```
Token physician in Role end
Token john in Subject with
   uses use1: physician
end
```

We can also define subjects to be complex data types containing the attributes password and birthdate.

```
Class Subject in SimpleClass with
attribute
   birthdate: Date;
   password: String;
   assigned: Role;
   uses: Role
end
```

While Telos will translate the above complex data types to a structure that is similar to the one that was explained in Section 6.2.1, we can abstract from this and do not need to worry about it.

The definition of Subject can be extended with the constraint that a role can only be activated on a subject if it has been assigned to him. The definition of Subject will then be

```
Class Subject in SimpleClass with
attribute
   birthdate: Date;
   password: String;
   assigned: Role;
   uses: Role
constraint
   c: $ forall r/Role (this uses r)
         ==> (r in this.assigned)
      $
end
```

In other words, the Telos language meets all requirements that were outlined in Section 6.2.

6.3 Definition and Constraint Language

The basic concepts that comprise the EFSOC framework are: Role, Role Attribute, Subject, Event, Event Type. For each of these concepts, a corresponding Telos SimpleClass is defined. Additionally, roles and subjects can be related via relationships publish, assigned and uses. Subjects can be related to events via the sent relationships and subject can be related to event body types via the subscribe and publish relationships.

This leads to the following vocabulary for the definition and constraint language.

1. Subjects are defined as

```
Class Subject in SimpleClass with
attribute
   assigned: Role;
   uses: Role
   subscribedto: EventBody
constraint
   c: $ forall r/Role (this uses r)
         ==> (this assigned r)
      $
end
```

Axiom 6.1 *A role can only be used by a subject if the role is also assigned to that subject.*

2. Roles are defined as

```
Class Role in SimpleClass isA Principal with
attribute
    publishedby: Subject;
    roleattribute: RoleAttributeType
constraint
    c: $ exists s/Subject (this publishedby s) and
         forall s1/Subject s2/Subject
         (this publishedby s1) and (this publishedby s2)
         ==> (s1 == s2) $
end
```

Axiom 6.2 *All roles must be published by one subject.*

3. Role Attribute Types are defined as

```
Class RoleAttributeType in SimpleClass end
```

4. Role Attribute (values) are defined as

```
Class RoleAttribute in SimpleClass with
attribute
   roleAssignment: Subject!uses;
   roleAttributeType: RoleAttributeType;
   value: String
end
```

5. Events are defined as

```
Class EventClass in MetaClass with
attribute
   header: EventHeader;
   body: EventBody;
end
```

   An event class must be instantiated before it can be used as an event.

6. Event Bodies are defined as

```
Class EventBody in MetaClass with
attribute
   publishedby: Subject
constraint
   c : $ exists s/Subject (this publishedby s) and
         forall s1,s2/Subject (this publishedby s1) and
         (this publishedby s2) ==> (s1 == s2)$
end
```

   Axiom 6.3 *All event bodies must be published by one subject.*

7. Event Headers are defined as

```
Class EventHeader in MetaClass end
```

   Note that the Event Header class is defined as a meta class. As a result, it must be instantiated before it can be used in an event.

The minimal set of headers that an event must have is

1. sender; representing the sender of the event.

```
Class EventHeader_sender in SimpleClass,EventHeader
end
```

It is important that the only one who is able to set headers or modify header values is the EFSOC system itself. The reason for this is that event headers contain data that are important to achieve proper security. For example, if subjects were able to set their own sender header, identity spoofing would easily be achieved.

Assume the situation outlined in the previous example. The knowledge base will be populated as follows:

```
Token physician in Role with
publishedby
    p : efsoc
end

Token john in Subject with
assigned
   a1 : physician
uses
   u1 : physician
end
```

Further assume that john is able to prescribe medication to patients by generating prescribeMedicationEvents.

```
Class PrescribeMedicationBody in EventBody, SimpleClass
with attribute
   patient: Subject;
   drugname: String;
   dosage: String;
   numberofdoses: Integer
end

Class PrescribeMedicationEvent in EventClass, SimpleClass
with body
   b: PrescribeMedicationBody
end
```

As a result, an event instance which represents a prescription for Sue concerning 800 mg Ibuprofin, to be administered 3 times a day for a week would look like

```
Token prescription1bdy in PrescribeMedicationBody with
   patient s: sue
   drugname drug: "Ibuprofin"
   dosage dose: "800 mg, 3 times per day"
   numberofdoses: 21
end

Token prescription1event in PrescribeMedicationEvent
with
   body b:prescription1bdy
end
```

The full implementation of the EFSOC definition and execution language is included for reference in Appendix III.

6.4 Query Language

Having laid down the basic EFSOC vocabulary in the previous section, the section can now be used to specify queries. The EFSOC query language consists of a number of predefined queries that can be used to inspect the state of the model. For practical purposes, the query language can be divided in queries that directly query classes for instances, and for in queries that are more complex and add relationships.

For example, to find out which subjects are subscribed to a particular event body type, the following query can be used:

```
Token SubjectsBySubscription in GenericQueryClass
   isA Subject with
parameter
    event : EventBody
constraint
    c : $ (~this subscribedto ~event)
    $
end
```

A slightly more complicated query retrieves all subjects who have ever sent an event of a specific type. The query uses the fact that Telos considers all objects as individuals or as attributes.

```
Token SendersOfEventType in GenericQueryClass
   isA Subject with
parameter
   ebt: EventClass
constraint
   c: $ exists e/Individual s/EventHeader_Sender
        (e in ~ebt) and
        (e sender s) and
        (s subject ~this)
     $
end
```

Another example of a query which appears as a complicated one is illustrated by Figure 6.1. Assume that we wish to query the model to find all subjects who have a specific value for a given role attribute. For example, this query would be used to find out who are treating a particular patient.

Expressed in ConceptBase notation, this query looks as follows:

```
Token SubjectWithRoleAttributeValue in GenericQueryClass
   isA Subject with
parameter
   attr: RoleAttributeType;
   value: String
constraint
   c: $ exists ra/RoleAttribute u/Subject!uses
        (ra roleAttributeType ~attr) and
        (ra value ~value) and
        (ra roleAssignment u) and
        Ai(~this, uses, u)
     $
end
```

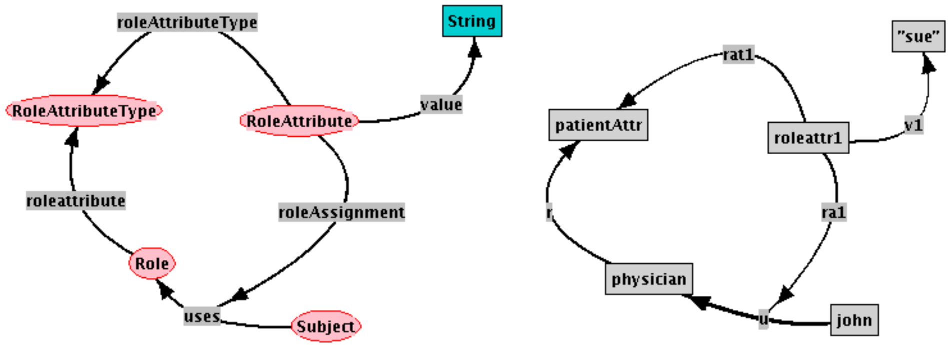

roleAttributeType
String
RoleAttributeType
RoleAttribute
value
roleattribute
roleAssignment
Role
uses
Subject
"sue"
rat1
patientAttr
roleattr1
v1
r
ra1
physician
u
john

Figure 6.1: Role Attribute Example

The query works by identifying the role attribute values that match on type and value, and, using the role assignment associated with the value, identify the appropriate subjects.

The full query language has been implemented as ConceptBase GenericQueryClasses and is included for reference in Appendix III.

6.5 Expressing Security Policies and Security Rules

As outlined previously Security Policies may contain access control rules, which regulate permission to execute certain operations, and membership rules, which regulate membership access to a policy. This section discusses how policies and rules are expressed, and how they can be used to achieve separation of duty, as explained in Section 2.6.2.

First, we establish the concept of a security policy:

```
Class AccessControlPolicy in SimpleClass with
    attribute
        owner: Subject
end
```

This provides us with a simple owned class that can be referenced by access control rules at a later point. As expressed in a previous chapter, access control rules themselves consist of an operation, a principal and a condition, and belong to a specific policy and may have a priority. This is expressed as follows:

```
Class AccessControlRule in SimpleClass with
    attribute
        operation: Operation;
        condition: Proposition;
        permission: Permission;
        priority: Integer;
        policy: AccessControlPolicy;
    principal: Principal
end
```

Enforcing these access control policies and access control rules is kept out-of-scope. The goal of this formalization is to explain the EFSOC concepts and their relationships using precise semantics. For that reason, we will provide a way to specify access control constructs, but we will not implement the algorithm in ConceptBase.

6.5.1 Security Rules: Queries or Constraints

Security rules can be viewed as constraints which preserve the state of a database in a logically consistent fashion, or as queries which can be executed by a reference monitor who uses them to decide whether or not access will be granted. In EFSOC, we choose view security rules as queries.

6.6 Implementing Separation of Duty

Separation of duty comes in two different forms: static separation of duty prevents two roles or more to be *assigned* simultaneously to a subject, while dynamic separation of duty allows roles to be assigned simultaneous, but not to be *activated* at the name time.

Encforcing separation of duty constraints can be divided in two parts: maintaining records of which combinations of roles will be considered as a separation of duty (either static or dynamic), and enforcing the separation of duty.

A static separation of duty can be described as:

$$\begin{aligned}\forall s \in \mathsf{Subject}\ r1, r2 \in \mathsf{Role}\ (s\ \mathsf{assigned}\ r1)\ \wedge\ (s\ \mathsf{assigned}\ r2)\ \wedge\ \neg(r1\ ==\ r2)\\ \Rightarrow\\ \neg\exists f \in \mathsf{SSDConstraint}(f\ \mathsf{role}\ r1) \wedge (f\ \mathsf{role}\ r2)\end{aligned}$$

Smilarly, a dynamic separation of duty can be described as:

$$\begin{aligned}\forall s \in \mathsf{Subject}\ r1, r2 \in \mathsf{Role}\ (s\ \mathsf{uses}\ r1)\ \wedge\ (s\ \mathsf{uses}\ r2)\ \wedge\ \neg(r1\ ==\ r2)\\ \Rightarrow\\ \neg\exists f \in \mathsf{DSDConstraint}(f\ \mathsf{role}\ r1) \wedge (f\ \mathsf{role}\ r2)\end{aligned}$$

The formula specifies that if a single subject is assigned two (or more) distinct roles, there may not be a static separation of duty constraint that restricts the two roles from being used as the same time.

The separations of duty are enforced by two classes which only contain a simple consstraint. These classes are called `SSDEnforcer` and `DSDEncforcer`. To specify a dynamic separation between physicians and patients, and between nurses and patients, and between physicians and nurses, the following constraints can be specified:

```
Token dsd1 in DSDConstraint with
role
    r1: physician;
    r2: patient
end

Token dsd2 in DSDConstraint with
role
   r1: nurse;
   r2: patient
end

Token dsd3 in DSDConstraint with
role
   r1: physician;
   r2: nurse
end
```

6.7 Relationship with EDL

All of the Telos classes listed in the previous sections are direct representations of the EDL specification language elements, which were introduced in Chapter 4. In addition to these representation, this formalization include model constraints to ensure the consistency of the model. These model constraints provide a concrete and unambiguous representation of assumptions that were left implicit until know. The assumptions are:

1. A role can only be used by a subject if the role is assigned to that subject.
2. All roles must be published by exactly one subject.
3. All event body types must be published by exactly one subject.
4. Roles for which a dynamic separation of duty has been defined may not be *used* by a subject at the same time.
5. Roles for which a static separation of duty has been defined may not be *assigned* by a subject at the same time.
6. Event Body Types may not be used in *send* operations, until they have been published.
7. When an event body type is unpublished, all subscriptions to that event body type are unsubscribed automatically.

8. When a role is unpublished, all assignments to that role are unassigned.
9. When a role is unassigned, all activations of that role are deactivated.

Telos allows constraints to be specified and enforced by specifying them as part of any class. Constraints are defined on a global level, which means that their restrictive power is not limited to the class in which they are defined.

6.8 Discussion

In this chapter, we expressed the EFSOC definition and constraint language and of the EFSOC query language using the Telos language. To validate that the implementation is correct, we implemented the full model in ConceptBase. The associated specifications are available in Appendix III.

Chapter 7

Prototype Implementation

> The World is a book, and those who do not travel read only a page.
> *St. Augustine, 354–430*

7.1 Introduction

Having defined the EFSOC concepts and architectures in Chapter 3, the EFSOC definition and constraint language in Chapter 4, and finally the EFSOC query language in Chapter 5, this chapter applies the concepts and language elements in the form of a prototype implementation and a laboratory experiment using that prototype.

First, we discuss the laboratory experiment, which is based on the running example of Section 3.2. Next, we introduce the architecture of the main prototype and the technologies that were used to implement it. We also discuss a number of smaller proofs of concept that were implemented during the research phase of this project.

Having introduced the architecture and the technologies, we then proceed by using them to implement the laboratory experiment. Those results are discussed in Sections 7.4 and 7.5. Finally, we discuss the lessons learned and we conclude the chapter in Section 7.6.

7.2 Laboratory Experiment

To test the prototype, we defined a laboratory experiment based on the running example of Section 3.2. In the experiment, we defined four subjects named *John*, *Mark*, *Mary* and *Sue* and three roles *Physician*, *Nurse* and *Patient*. The physician role will have one role attribute, *patient*, and the nurse role will have one role attribute, *floor*.

John and Mark will be members of the physician role, Mary will be a member of the Nurse role and Sue will be a member of the Patient role. Furthermore, Sue will be listed as one of John's patients and Mary will work on the Oncology floor, on which Sue is a patient.

In addition to these human subjects, we define the following services: *Billing Service*, *Charting Service*, *Laboratory Service*, *Radiology Service*, *Pharmacy Service* and the *Insurance Service*.

We also define the following events: *UpdateChart*, *InspectChart*, *Chart*, *OrderLaboratoryTest*, *RetrieveTestResults*, *LaboratoryTest*, *PrescribeMedication*, *DispenseMedication*.

In the laboratory experiment, the following access control policies are defined:

1. *Charting Service Policy*
   The charting service policy dictates that physicians and nurses may access any patient's records. Charts may be updated by the physicians who are treating a patient, or by a nurse if the patient is on her floor.

2. *Laboratory Service Policy and Radiology Service Policy*
   The laboratory service policy dictates that the charting service, nurses and physicians may access *all* test results as they become available. Only physicians may order tests to take place.

3. *Pharmacy Service Policy*
   The pharmacy service policy dictates that only physicians may prescribe medications to patients. Medication may be dispensed to patients, only if a prescription for that medication and patient has been received and is not yet filled.

7.3 Architecture and Technology

For the creation of the prototype, a two-track approach was followed. The main track consists of a thorough implementation, using as many ‘production-grade’ components as possible. The secondary track consists of smaller proofs of concept, built from scratch, using either ConceptBase or small scripts written in the PHP scripting language. The two tracks complement each other, because the smaller proofs of concept are easier to adapt to changing viewpoints and provide a ideal proofing ground for early ideas.

7.3.1 Main prototype

The EFSOC model itself does not impose any requirements for *bootstrapping* the model. In the main prototype, we have addressed this by having one hard-coded subject called ’efsoc’. The policies associated with this user play the role of *EFSOC global policy* and can be used to constrain publication of new subjects, event body types or roles.

The main prototype was implemented as a set of Java server pages that are deployed using Apache Tomcat[1]. The Tomcat engine is bundled with the JBoss application server[2], which is J2EE 1.4 certified. The prototype architecture is graphically depicted in Figure 7.1.

---

[1] `http://tomcat.apache.org/`

[2] `http://www.jboss.com`

The main prototype consists of the five main components outlined in the main EFSOC block:

1. *Event Director*
   The Event Director manages event subscriptions and provides the interface by which services exchange messages. The event router implements subscription-based event routing.

   All interaction between EFSOC and external services are channeled through the event director. The event director uses Apache's Axis[3] to provide support for WSDL messages.

2. *Security Director*
   The Security Director implements EFSOC's access control model. It provides access control policy selection, evaluation and enforcement.

3. *Persistent Storage*
   The Persistent Storage module is built using existing technologies. The underlying database is MySQL[4], which may be accessed through Hibernate[5]. Hibernate enables a developer to develop applications according to the object-oriented paradigm and maps objects to relational databases.

4. *Audit Trail Director*
   The Audit Trail Director is responsible for capturing, storing and protecting all state changes of the EFSOC model. The Audit Trail Director also provides facilities for querying the model.

5. *Java Messaging System*
   All EFSOC modules interact with each other via the Java Message System which is provided by JBoss. The Java Message System may be replaced by Enterprise Service Bus implementations.

In addition to the components described above, a management interface which directly access the database has been realized. That interface will allow for easier inspection and debugging of the prototype implementation.

7.3.2 Proofs-of-concept

In addition to the main prototype implementation, a number of smaller proofs-of-concept (PoC) have been realized. Each proof-of-concept address specific and highly specialized issues, such as validating EDL or testing the EFSOC query language.

---

[3] `http://axis.apache.org`

[4] `http://www.mysql.com`

[5] `http://www.hibernate.org`

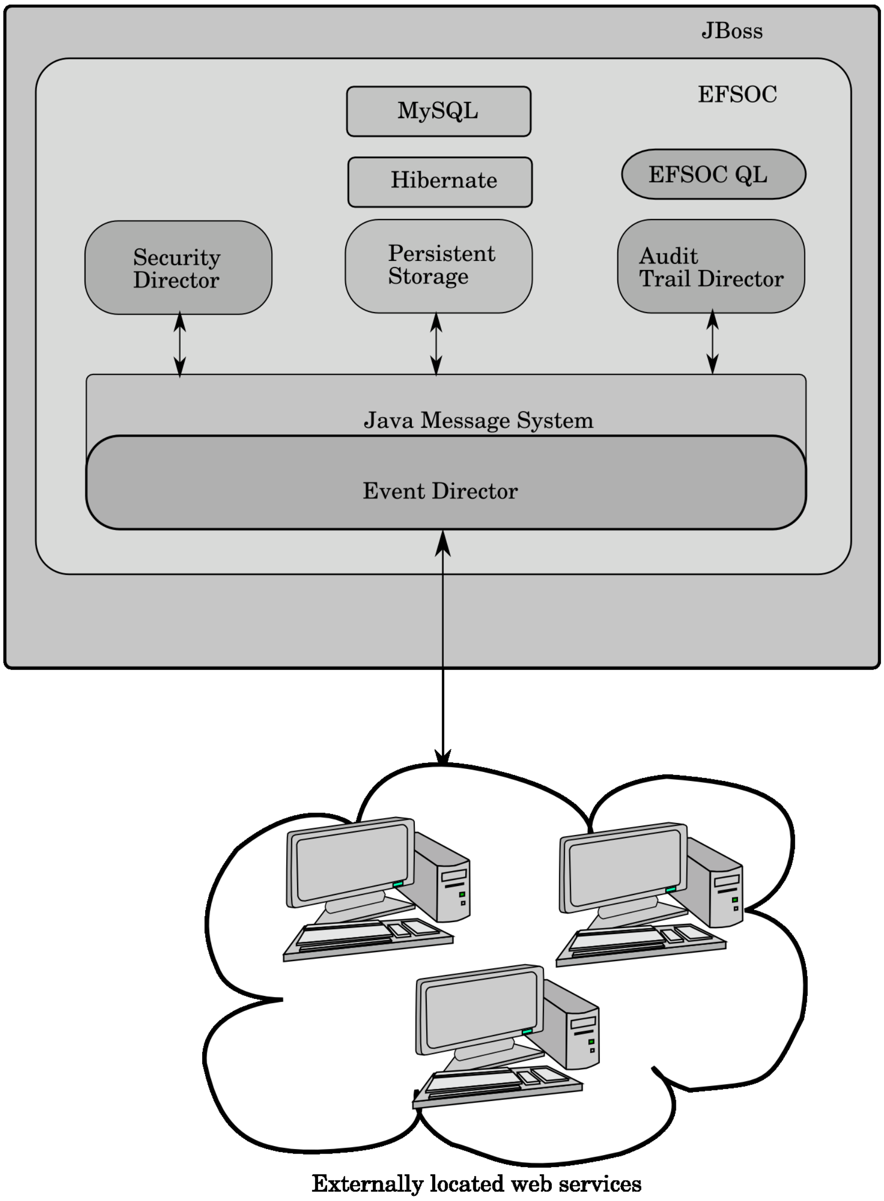

JBoss
EFSOC
MySQL
Hibernate
EFSOC QL
Security Director
Persistent Storage
Audit Trail Director
Java Message System
Event Director
Externally located web services

Figure 7.1: Prototype reference architecture

The PoC's are generally not written using the same technology as above, but rely on scripting languages such as the PHP language[6].

The reason for using the script languages is that they support a much faster development cycle because they eliminate the need to compile and deploy sources via the application server environment that is used for the main prototype.

Definition Language Validator

The language validator is fully written in the PHP5 scripting language and relies heavily on PHP's abilities to handle DOM trees and evaluate XPath expressions. The objectives of the prototype can be summarized as follows:

1. Validate case study specifications (listed in Section III) against EFSOC XML Schema presented in the same section.

2. Ability to parse access control rules against a number of pre-defined events.

In addition to the functionality specified above, the language validator will also resolve role assignments to the corresponding users. A partial screen-shot is shown in Figure 7.2.

As shown, the charting service's access control policies contains a rule which permits physicians and nurses to send events, under a certain condition. The condition is expressed as an XPath query and can be executed by clicking on the corresponding like. The user interface includes an expansion of the roles physician and nurse to the corresponding subjects.

Execution of the second access control rule leads to the screen as shown in Figure 7.3. The screen-shot includes the XPath query, and the results of the evaluation of the query on the full database. In this case, the event shown in the figure will be permitted to be sent.

Query Language Validator

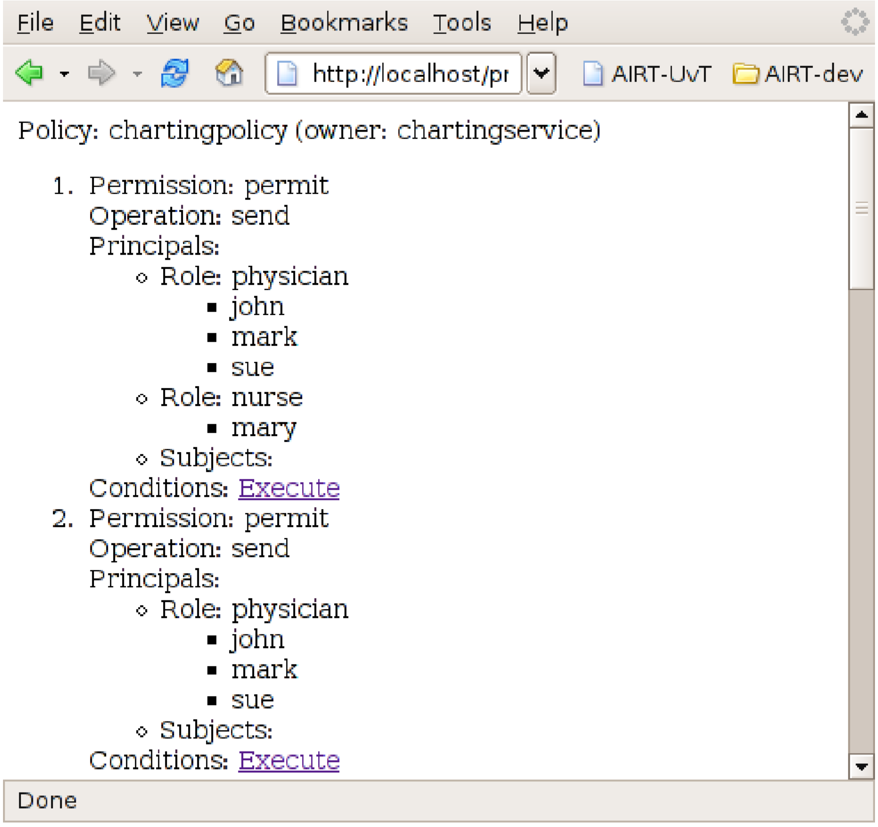
The EFSOC query language was implemented as a collection of XQuery functions. To come to this decision, we evaluated three possible technologies:

1. *XPath*
   To evaluate XPath queries, we used a tool called 'XPath Explorer'[7]. The tool provides a graphical user interface which loads an XML fragment, parses it into a DOM tree and graphically presents it to the user. Using a simple input field, XPath queries can be formulated and they are applied directly to the DOM tree in the GUI. This allows for rapid evaluation of XPath queries and proved to be an invaluable tool.

---

[6] `http://www.php.net`

[7] `http://xpe.sourceforge.net`


File
Edit
View
Go
Bookmarks
Tools
Help
http://localhost/pr
AIRT-UvT
AIRT-dev
Done

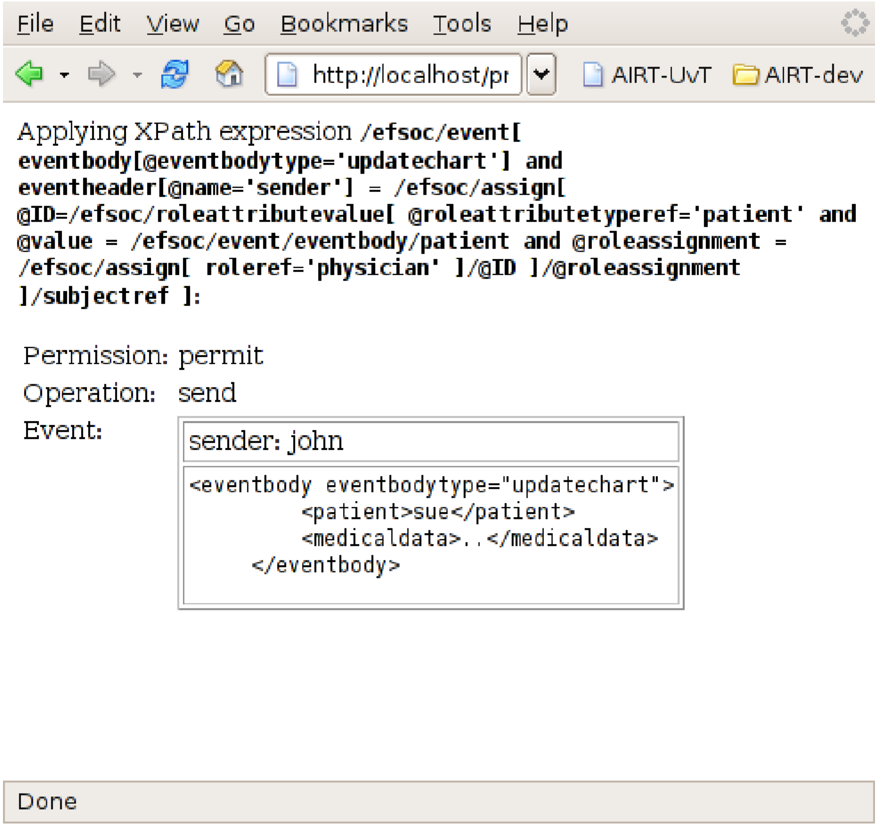
Policy: chartingpolicy (owner: chartingservice)

1. Permission: permit
   Operation: send
   Principals:
   - Role: physician
     - john
     - mark
     - sue
   - Role: nurse
     - mary
   - Subjects:

   Conditions: Execute
2. Permission: permit
   Operation: send
   Principals:
   - Role: physician
     - john
     - mark
     - sue
   - Subjects:

   Conditions: Execute

Figure 7.2: Language validator: Access control policy


File Edit View Go Bookmarks Tools Help
http://localhost/pr
AIRT-UvT
AIRT-dev
Done

Applying XPath expression `/efsoc/event[eventbody[@eventbodytype='updatechart'] and eventheader[@name='sender'] = /efsoc/assign[@ID=/efsoc/roleattributevalue[@roleattributetyperef='patient' and @value = /efsoc/event/eventbody/patient and @roleassignment = /efsoc/assign[roleref='physician']/@ID]/@roleassignment]/subjectref]`:

Permission: permit
Operation: send
Event:

sender: john

```
<eventbody eventbodytype="updatechart">
        <patient>sue</patient>
        <medicaldata>..</medicaldata>
    </eventbody>
```

Figure 7.3: Language validator: Access control rule evaluation

2. *XSL Transformations*
   To evaluate XSL Transformations, we had to develop our own code. Like in the case of the validator for the definition and constraint language, we used the PHP scripting language, and used its interface to the libxslt programming library[8].

   The tool that we developed only provided a rudimentary interface with allowed us to load two files: one XML file containing EFSOC definitions, and one file containing the XSL style-sheet. The prototype subsequently applied the style-sheet to the XML document and presented the output in a web browser.

3. *XQuery*
   While XQuery is often heralded as the language of choice for querying XML data, surprisingly few implementations turned out to be available that support the whole XQuery candidate recommendations. Eventually, we settled on using two existing pieces of software. The first is called ipsi-xq, which is developed by the German Fraunhofer institute. Ipsi-XQ[9] provides full support for XQuery, including the ability to specify functions. It also comes with a graphical user interface, as well as a command-line query processor. The second tool is called Qexo[10]. Qexo is distributed as Free Software under the GNU General Public License and is integrated in the Kawa Language Framework[11].

   Both Qexo and Ipsi-XQ provide interactive ways of specifying and executing XQuery statements, and did not need any additional development to meet our purposes.

7.4 Element definitions

Having determined the playing field and the scope of the laboratory experiment, we proceeded by defining the subject, roles and role-attributes, and we assigned roles to users and role attribute values to role assignments.

The corresponding definitions can be found in Appendix III. The same definitions were also implemented in the ConceptBase implementation of the model and are included in Appendix III. For example, the following fragments shows the Telos and XML definitions of the subject ‘John’ playing the role ‘Physician’ and having role attribute ‘Patient’ set to the value ‘”sue”’.

```
Token john in Subject with
   assigned a: physician
   uses u: physician
end
```

---

[8] `http://xmlsoft.org/XSLT`

[9] `http://www.ipsi.fraunhofer.de/`

[10] `http://www.gnu.org/software/qexo/`

[11] `http://www.gnu.org/software/kawa`

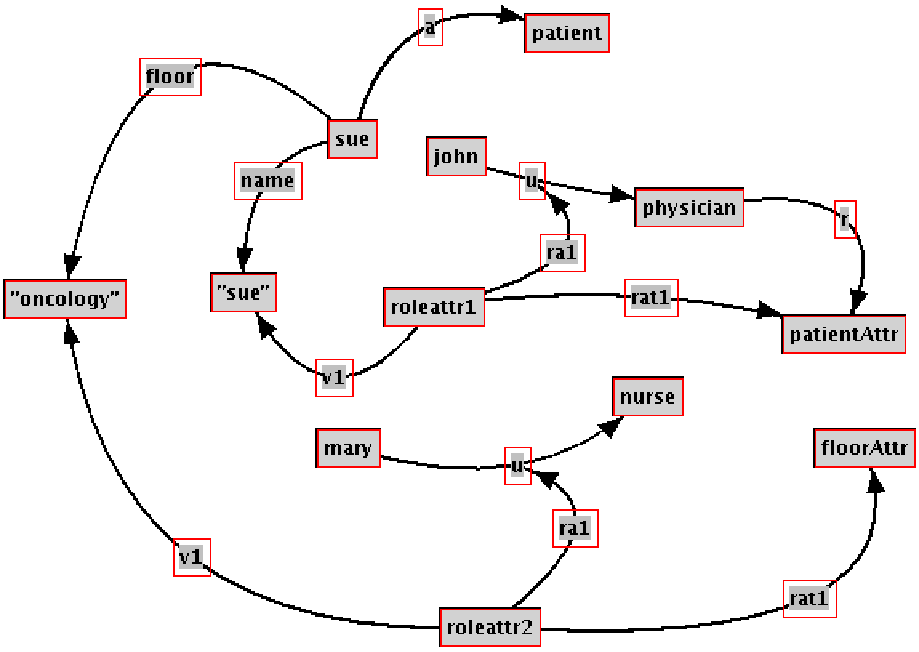

a
patient
floor
sue
john
name
u
physician
r
ra1
"oncology"
"sue"
roleattr1
rat1
patientAttr
v1
nurse
mary
u
floorAttr
ra1
v1
rat1
roleattr2

Figure 7.4: Instance level definitions in laboratory experiment

```
Token roleattr1 in RoleAttribute with
    roleAssignment ra1: john!u
    roleAttributeType rat1: patientAttr
    value v1: "sue"
end
```

The XML representation of the same definitions is included below:

```
<subject ID="john"/>
<assign ID="ra1">
     <subjectref>#john</subjectref>
     <roleref>#physician</roleref>
</assign>

<roleattributetype ID="patientAttr"/>
<roleattributevalue ID="roleattr1"
    roleattributetyperef = "#patientAttr"
    roleassignmentref = "ra1"/>"sue"</roleattributevalue>
</roleattributevalue>
```

A graphical representation of the full set of role definitions, subject definitions, role assignment, role attribute types and role attribute value definitions is show in Figure 7.4.

Starting near the top-right of the figure, the object labeled ‘john’ can be found.

John uses the ‘physician’ role, which has a role attributed associated with it called ‘patientAttr’. The object with label ‘roleattr1’ is a role attribute with role attribute type ‘patientAttr’ and is linked to John’s role as physician. The value of the role attribute is “Sue”, which is the name of the object labeled ‘sue’.

Sue is ‘patient’ who is admitted to the ‘oncology’ floor.

Mary, who is a nurse, works on the oncology floor. This is represented by the role attribute ‘floorAttr’, which used by the object with label ‘roleattr2’ to denote the fact.

Using the graph, relationships between John, Mary and Sue are illustrated. John is physician who treats Sue, Sue is a patient on the Oncology ward, and Mary works there.

Similar definitions were also made to represent the different event body types, as illustrated by the Telos definitions of the event body type for ‘PrescribeMedication’ event body.

```
Token PrescribeMedicationBody in EventBody with
attribute
    patient: String;
    drugname: String;
    dosage: String;
    numberofDoses: Integer
end
```

The definition specifies that the event body of a ‘PrescribeMedication’ event must contain four elements: patient, drugname, dosage and numberofDoses.

An actual event body of this type then looks like

```
Token prescribeMedicationBodySue in PrescribeMedicationBody
with
    patient   p: ”sue”
    drugname drug: ”ibuprofin”
    dosage    d: ”600mg 3 times per day after meal”
    numberofDoses num: 15
end
```

Such an event body can then be used to create an event.

7.5 Defining access control rules

The next step in the approach is to define a set of access control rules. The requirements for the rules were introduced in Section 7.2.

In Appendix III we provide the fully specified access control policy for the Northside laboratory experiment using the EDL format. In this section, we will take one rule and discuss it in more detail. For this, we have chosen the second rule in the charting service’s access control policy. That rule states that medical charts may be updated by physicians who are treating the patient.

In EFSOC terms, this means that the principal of the rule is the role ‘physician’, the permission is ‘permit’ for a ‘send operation’.

This is represented by the first part of the access control rule.

```
<accesscontrolrule id=”chartingrule2”
                   policyref=”#chartingpolicy”
                   operation=”send”
                   permission=”permit”>
   <principal>
      <roleref>#physician</roleref>
   </principal>
```

The `<accesscontrolrule>` element contains a number of attributes which specify a unique identifier, a reference to the policy of which the rule is part, and references to operations and permissions. The first element in the body of the tag is the `<principal>` tag, which specifies to who the access control rule applies. In this example, rule applies to all subjects playing the role ‘physician’, but references to multiple roles and references to multiple subjects may be listed here.

The remainder of the access control rule states under which conditions physicians are permitted to send events.

```
1     <condition>
2  (/efsoc/event/eventheader[@name=”sender”] =
3  /efsoc/assign[roleref=”physician”]/subjectref)
4  AND
5  (/efsoc/event/eventbody[@eventbodytype=”updatechart”]
6  /patient =
7  /efsoc/roleattributevalue[
8            @roleattributetyperef=”#patient”
9     AND  @roleassignment=/efsoc/assign[
10          subjectref=/efsoc/subject[
11             @ID=/efsoc/event/eventheader[@name=”sender”]
12          ]/@ID
13     ]/@ID
14 ])
15    </condition>
16 </accesscontrolrule>
```

The `<condition>` element takes an XPath(Berglund et al., 2005) expression as its body. Complex XPath expressions may lead to complex rules, as shown above. However, we feel that this is not a problem since much of the rule creation can be automated and hidden from the end-users.

This particular expression states in lines 2–3 that the sender of the event must be a member of the role ‘physician’. This is tested by inspecting the ‘sender’ header of the event envelope, which is set by EFSOC. Next, a test is performed to see if the role ‘physician’ is indeed assigned to this subject.

Lines 5–14 validate that the type of the event body is ‘updatechart’, and that the ‘patient’ element of the event body is listed in the subject’s role attribute values of the role attribute type ‘patient’.

When all these conditions are met, the rule will ‘permit’ the event to be ‘sent’.

7.6 Observations and Conclusions

The proofs-of-concept for the definition and constraint language and for the query language illustrated in a very convincing way that our initial thoughts, which were to use XPath exclusively, would not be a good choice. While it would be possible to use XPath, the expressions would be excessively complicated and hard to write or understand.

Implementing the laboratory experiment made it obvious that the EFSOC model requires parameterized roles. Not doing so would have resulted in the definition of a large number of roles, each of which would only have a very limited number of members. This in itself is not undesirable per se, however when analyzing the situation, it became clear that in the model without such role parameters, each patient would need to have a role designed specifically for them. This will invalidate the main benefit of role-based access control (i.e., the reduced management overhead), and must therefor be addressed differently.

After introducing parameterized roles into the EFSOC conceptual framework, we were able successfully use the framework to express all definitions and access control policies, and implement them in the prototype.

Chapter 8

Conclusions, Discussion and Future Research

8.1 Summary

If one cannot effectively manage the growing volume of security events flooding the enterprise, one cannot secure one's business. New technologies and the continued expansion of the enterprise environment only means that this security overload will get worse (Kelley and Moritz, 2006).

Service-Oriented Computing (SOC) is the computing paradigm that utilizes services as fundamental elements for developing applications (Papazoglou and Georgakapoulos, 2003). The paradigm of service-orientation is a relatively recent enrichment to the discipline of designing distributed applications. Its vision encompasses a future in which application development is not constrained by organizational or technological boundaries.

Services may be perceived as self-describing, open components that support rapid, low-cost composition of distributed applications.

The need for addressing security in web services has been acknowledged in an early stage by the IBM Corporation and the Microsoft Corporation. In April, 2002, the two software giants published a joint white-paper titled *Security in a Web Services World: A Proposed Architecture and Roadmap* (Web Services Security Roadmap, 2002).

The architecture that is presented in the article proposes a solution which places the entire security stack in the realm of SOAP headers.

Unfortunately, despite many years of work, the security road-map is currently in an early stage of development; only the WS-Security and the WS-Policy standards have reached a stage of adolescence.

In addition, the security road-map is strongly tied to the web services SOA in that all security operations are implemented in basic SOAP headers. Finally, there is no common vocabulary or shared processing model for the components that make up the joint road-map.

To address the issue of security, and, more specifically, *access control* in service-oriented computing, we developed the EFSOC framework. EFSOC conceptualizes service-oriented computing and access control and provides an architecture and suggested implementation for a secure service middleware layer.

EFSOC lets go of the premise that services are required to invoke each other directly. Instead, we assume that services publish their interfaces to a *service broker*.

Adopting this approach facilitates the service broker (or the service brokers, in case of a distributed solution) to establish an certifiable *audit trail* in a single location, which can be used to settle differences of opinion which may arise in the future.

Furthermore, by routing the service interactions through a broker, it becomes possible to specify and enforce access control policies unambiguously and at a single point. An event-driven approach with distributed service brokers provides a flexible mechanism for ensuring increased availability of services, the ability to implement competing services and provides a reliable infrastructure for service interactions.

This chapter summarized and discusses the research results that were presented in the previous chapters, and the research questions that were formulated in Section 1.4 are answered.

8.2 Research Results

In Chapter 1, we explained the research methodology for this thesis. The methodology consisted of a problem definition phase, in which the research goal was formulated and the strategic research questions were formulated. The literature research was used to refine the strategic research questions and find answers to those questions. The solution design phase continued finding answers to questions, by proposing an architecture, definition language and query language. The validation phase consisted of the creation of the Telos model which was used to analyze the framework, a case study to establish usability and a prototype to illustrate the fact that the architecture can be implemented.

In this section, we provide answers to the strategic research questions.

1. **Question**: What is the state-of-the-art in access control?

   Answer: The current state-of-the-art in access control is captured by Role-Based Access Control, and by the specifications in XACML and SAML. Additionally, the WS-* security specifications apply these standards to web services technology.

2. **Question**: Is the current state of the art of access control adequate for use in service-oriented computing?

   Answer: The current state-of-the-art in access control is comprised of the Role-Based Access Control model (RBAC), and of technologies such as XACML and the WS-* security specification.

   Role-based access control is not sufficient for use in service-oriented computing for the following reasons (see also Section 2.2.8):

   (a) RBAC does not define the meaning of ’permissions’.
   (b) RBAC assumes a central administration of access control.

   The WS-* security specifications are currently not developed enough to be ready for large scale use. Of the total set of specification, the WS-Authorization standard addresses the issue of access control and authorization. Unfortunately, WS-Authorization only suggests a generic representation format, and does not include a common methodological framework or reference architecture.

   A reference framework for access control in distributed environments is provided by XACML. However, XACML also does not provide a common methodological framework.

The questions that arose as a result of the framework design are answered as follows:

3. **Question**: In what way should a definition language that supports the framework be formed?

   Answer: A support language that supports the framework should be formed as a declarative XML-based language. The dominant reason for this choice is that XML provides an easily extendible infrastructure, which is widely accepted in the software development industry. As a result software tools are readily available.

   A second reason is that by choosing an XML-based approach, interoperability with other XML-based specifications becomes trivial.

4. **Question**: In what way should a query language that supports the framework be formed?

   Answer: A query language should be formed as a set of XML Queries. XML Query is a query language specifically designed for querying XML documents, and has reached an early stage of maturity. By defining the query language as XML Queries, they are portable and easily implemented in a variety of XML-based infrastructures.

5. **Question**: What does an architecture for implementing the framework look like?

   Answer: The architecture that is presented in Chapter 7 uses existing of-the-self technology. By using an XML-based specification language, and an XQuery-based query language, the proposed architecture uses very few components that needed to be developed fully from scratch.

   We proposed a solution which is based on commonly available Java components, such as JMS. The application server that was chosen was JBoss. A detailed description of the architecture can be found in Chapter 7.

8.3 Case study results

The case study that was performed at Northside Hospital provided us with valuable feedback about our approach. In a number of telephone interviews that were conducted with the hospital's information technology specialists, it became clear that a service-oriented approach will offer many benefits for the organization when they move to fully computerized medical charts.

One of the most important observations that we made was that the number of roles was almost proportional to the number of patients and/or physicians. This observation seriously undermined one of the main reasons to adopt a role-based approach. Fortunately, by introducing the concept that roles can be parameterized by use of *role attributes*, we were able to overcome this problem.

After adding role attributes to the model, the case study showed that we were indeed able to express a realistic scenario using EFSOC concepts. A detailed description of the definitions that were used can be found in Appendix III.

In addition to the lessons learned from conducting the case study, the implementation of the case study in the laboratory experiment confirmed that the technology choices that were made for the prototype were correct. Using as much existing off-the-shelf technology, we were able to relatively quickly deploy the Northside scenario in the prototype.

8.4 Contributions

Current approaches to access control in distributed environments tend to focus on centrally administered access control policies, which are deployed to decentralized points of enforcement. The benefit of this approach is that security administrators only need to keep one central policy, which allows them to maintain large access control systems.

In service-oriented architectures, this assumption is not valid. While services that belong to one single organization may be administered centrally, cooperating services may be offered by different organizations, each of which prescribes their own security policies.

We assume that all services should be in full control of their own security policies. In other words, we advocate decentralized access control policy administration. In this thesis, this principle is known as the *service autonomy* principle, at the solution that we propose respects this principle.

Contribution 1 *We introduced an access control model for service-oriented architectures that is decentrally administered, yet centrally enforced.*

Of all the access control models, Role-Based Access Control is often heralded as the most modern approach, and as an approach that closely aligns with the way that enterprises organize their processes. Role-Based Access Control (RBAC) is based on the abstraction of users into roles, and of the assignment of permissions to roles. An underlying assumption is that the mapping of users to roles is more volatile than the mapping of roles to permissions, but that the mapping of roles to permissions is larger. By abstracting users from their roles, the latter administrative load reduces, and the system becomes more scalable.

EFSOC adopts a parameterized role-based approach, but acknowledges the fact that some permissions must not be assigned to roles, but should be assigned to individuals within a role. The direct assignment of users to permissions is often called discretionary access control.

Contribution 2 *EFSOC adopts a discretionary parameterized role-based access control model.*

Service-oriented architectures do not implement a common access control model. A specification known as WS-Authorization attempts to fill this gap, however, it does not provide a common methodological framework for access control, and it does not provide a reference architecture. Instead, WS-Authorization restricts itself to providing a generic representation format for access control data.

Contribution 3 *EFSOC provides a common methodological framework and provides a reference architecture for decentrally managed, yet centrally enforced, discretionary parameterized role-based access control.*

The benefits of our approach are:

1. Services are able to individually specify their own access control policies, but can rely on EFSOC for enforcement of those policies. As a consequence, EFSOC provides service autonomy.
2. EFSOC's messaging model (which is an event-based model), conveniently maps to the Enterprise Service Bus. As a consequence, EFSOC can easily be implemented in existing enterprises that have adopted the ESB.
3. By using a common definition of roles and event types, EFSOC provides a natural approach to facilitate services that span organizational boundaries.
4. EFSOC provides a common framework for access control in Service-Oriented Architectures, which captures the roles that subjects play in the processes that require or provide services. That information can subsequently be used to provide access control policies.

8.5 Benefits and Limitations

Adopting the EFSOC approach enables organizations that adopt a service-based approach to decentrally specify access control policies, without the need to also implement access control functions in each service. Such a reduction in code size increases the speed by which new services can be created and published, and reduces the complexity of services.

A potential area which may slow down the adoption of EFSOC may be that all service providers who participate must be aware of the semantics of roles and event body types. However, using standard classification schemes to achieve interoperability is an area that is currently receiving much attention by the ontology research community.

EFSOC's messaging model provides a convenient way to distribute events over multiple services, which can be used to implement multicast messaging. In many cases, a single event may need to be delivered to many subscribed parties and EFSOC's approach allows this to take place naturally.

Another area of interest is not specific to EFSOC, but applies to *all* web-based solutions. By channeling traffic over HTTP or HTTPS, traditional firewalling approaches become much less effective. A traditional network firewall operates based on the network addresses and ports of the sender and the of receiver of data *packets*. Most firewalls are capable of providing some additional analysis, which allows sessions to be established and maintained. Even more advanced firewalls are capable of inspecting the protocol that is transferred over such ports (e.g., HTTP). With web-services technology increasingly being adopted, firewalls will need to be come more and more intelligent to prevent data from leaving or entering the organization without proper authorization.

8.6 Future Research

After having conducted the research presented in this thesis, many new questions arise that need further attention. These questions include, but are not limited to:

1. *Methodology for Access Control in Service-Oriented Architectures*
   After having developed a policy-neutral framework for access control in service-oriented architectures, we came to the conclusion that the well-known phrase "security transcends technology" applies fully in the SOA domain. Whereas most current efforts are technology driven, EFSOC provided an architecture-driven solution.

   However, while the architecture-driven solution is an critical tool for the implementation of access control requirements, it does not address issues such as identification of critical assets, risk management procedures, continuity planning, etc.

   Since the focus of this research was on access control, these elements have been left out-of-scope justifiably. Now that this part of the research has completed, it is time to step back and look at security in the bigger picture and consider those elements once more. How are critical assets identified in service-oriented architectures? What are the specific risks associated with those assets, and how can those risks be assessed and mitigated?

2. *Service clusters*
   Identity management is an important consideration in any business application. Service-oriented applications rely on short term collaboration which are

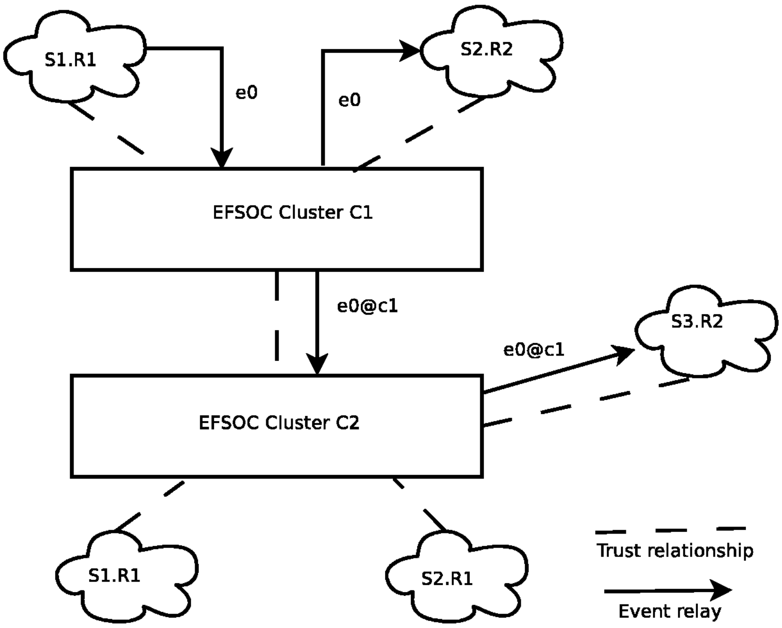


Figure 8.1: Trust relationships spanning service clusters

often created and used in an ad-hoc fashion. Such collaborating services may not belong to the same EFSOC service cluster, yet it is desirable that identities that have been established in one cluster can be ported to others.

To this effect, services are allowed to authenticate themselves at one EFSOC service, and re-use that identity in other clusters. While it is possible to regulate the use of such ‘deep identities’ in access control rule, the assumption is once more that the establishment of the trust-relationships per sé takes places out-of-band.

To illustrate this concept, consider Figure 8.1. It shows two EFSOC service clusters, each with a number of services participating in it. Both clusters have agreed on two things: role r_2 in both clusters have the same semantics and both clusters will acknowledge each other’s authentications of subjects in those roles.

We introduce the following notations:

(a) $s1.r1$ represents a subject $s1$ playing role $r1$.

(b) $@c1$ represents that an identity is limited to a certain EFSOC service cluster.

(c) $e1(e0)$ represents the fact that $e1$ is causally linked to event $e0$. In other words, $e1$ is a direct result of $e0$.

For example, $s1@c1$ represents subject $s1$ at cluster $c1$, $r2@c1$ represents role $r2$ at cluster $c1$, $s1.r2@c2$ represents subject $s1$ playing role $r2$ at cluster $c2$, etc.

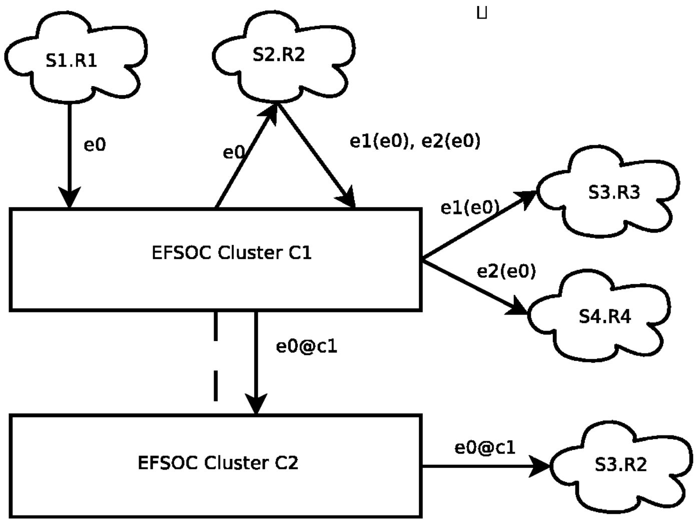


Figure 8.2: Trust relationships spanning service clusters combined with service aggregation

Given Figure 8.1, it is now possible to express access control rules such as 1) “event $e0$ may only sent by subjects playing role $r1$”, and 2) “event $e0$ may be sent to all subjects playing role $r2$”, or 3) “event $e0$ may only be sent to subjects in cluster $c1$ who are are playing role $r2$, The difference between rules 2) and 3) is that rule 2) only mentions that recipients must be playing role r2, which means that trust relationships will be respected and subject $s3@c2$ is also a valid recipient. Rule 3) explicitly states that the event may only be received by subjects in cluster $c1$. In that case, explicit restrictions supersede derived trust relationships.

This scenario can be extended as shown in Figure 8.2, which assumes that $s2@c1$ is actually a composite service which relies on two additional services $s3@c1$ and $s4@c1$.

For example, assume a hospital scenario in which a patient is committed for a routine surgery. The admission of the patient is represented by event $e0$. As a result of the patient getting admitted, patient records are sent to the hospital’s administrative department for proper billing ($e1$), and to the planning department so that an operating room can be scheduled ($e2$). In such a scenario, an access control rule that can be specified is that patient records may only be transferred to the planning department after the patient has been committed. In other words, a causal dependency exists between $e0$ and $e2$.

3. *Workflow support*
The EFSOC reference architecture provides the ability to deploy a workflow monitor, which may influence the way that messages are transferred. Using the workflow manager provides a way to bridge the gap between access control rules and business rules, and allows EFSOC-based services to interact

seamlessly with, for example, BPEL systems.

More research needs to be devoted to questions that address problems such as required primitives for expressing inter-service workflows, consequences of service-based workflows that cross organizational boundaries, handling of security-related exceptions in workflows, etc.

4. *Intrusion Detection Systems*
   No matter how well designed an architecture, and how well engineered the products that instantiate the architecture, there will always be attack vectors that can be exploited. While EFSOC attempts to provide a secure framework that provides strong access control and has containment features, it must be assumed that attacks against EFSOC-based systems will be attempted. This is true not only for EFSOC-based Service-Oriented Architectures, but for all SOAs.

   A future research project could resolve around the question of what types of attack vectors are specific to service-based solutions, how to detect attacks (successful and unsuccessful), and how to mitigate the effects of a successful attack.

Part III

Appendices

EFSOC XML Definitions

This appendix contains the XML Schema definitions of the EFSOC definition language (EDL) and the detailed definitions used for the laboratory experiment.

Definition and Constraint Language Defintions

```
<?xml version=”1.0”?>

<xs:schema
    targetNamespace = ”http://infolab.uvt.nl/efsoc.xsd”
    xmlns:xs = ”http://www.w3.org/2001/XMLSchema”>

  <xs:element name=”efsoc” type=”efsocType”/>

  <xs:complexType name=”efsocType”>
     <xs:sequence>
        <xs:element name=”operation” type=”efsocOperationType”
          minOccurs=”0” maxOccurs=”unbounded”/>
     </xs:sequence>
  </xs:complexType>

  <xs:complexType name=”efsocOperationType”>
    <xs:sequence>
      <xs:element name=”publish” type=”publishType”
        minOccurs=”0” maxOccurs=”unbounded”/>
      <xs:element name=”unpublish” type=”unpublishType”
        minOccurs=”0” maxOccurs=”unbounded”/>
      <xs:element name=”subscribe” type=”subscribeType”
        minOccurs=”0” maxOccurs=”unbounded”/>
      <xs:element name=”unsubscribe” type=”unsubscribeType”
        minOccurs=”0” maxOccurs=”unbounded”/>
      <xs:element name=”generate” type=”generateType”
        minOccurs=”0” maxOccurs=”unbounded”/>
      <xs:element name=”assign” type=”assignType”
        minOccurs=”0” maxOccurs=”unbounded”/>
      <xs:element name=”unassign” type=”unassignType”
        minOccurs=”0” maxOccurs=”unbounded”/>
      <xs:element name=”activate” type=”activateType”
```

```
        minOccurs=”0” maxOccurs=”unbounded”/>
     <xs:element name=”deactivate” type=”deactivateType”
        minOccurs=”0” maxOccurs=”unbounded”/>
   </xs:sequence>
</xs:complexType>

<xs:complexType name=”publishType”>
    <xs:sequence>
        <xs:element name=”subjectref” type=”xs:anyURI”
            minOccurs=”1” maxOccurs=”1”/>
        <xs:element name=”eventbody” type=”EventBodyTypeType”
            minOccurs=”0” maxOccurs=”unbounded”/>
        <xs:element name=”subject” type=”SubjectType”
            minOccurs=”0” maxOccurs=”unbounded”/>
        <xs:element name=”role” type=”RoleType”
            minOccurs=”0” maxOccurs=”unbounded”/>
        <xs:element name=”accesscontrolpolicy”
            type=”accesscontrolpolicyType”
            minOccurs=”0” maxOccurs=”unbounded”/>
        <xs:element name=”accesscontrolrule”
            type=”accesscontrolpolicyRuleType”
            minOccurs=”0” maxOccurs=”unbounded”/>
    </xs:sequence>
</xs:complexType>

<xs:complexType name=”unpublishType”>
    <xs:sequence>
        <xs:element name=”subjectref” type=”xs:anyURI”
            minOccurs=”1” maxOccurs=”1”/>
        <xs:element name=”eventbodyref” type=”xs:anyURI”
            minOccurs=”0” maxOccurs=”unbounded”/>
        <xs:element name=”subjectref” type=”xs:anyURI”
            minOccurs=”0” maxOccurs=”unbounded”/>
        <xs:element name=”roleref” type=”xs:anyURI”
            minOccurs=”0” maxOccurs=”unbounded”/>
        <xs:element name=”accesscontrolpolicyref” type=”xs:anyURI”
            minOccurs=”0” maxOccurs=”unbounded”/>
        <xs:element name=”accesscontrolpolicyref” type=”xs:anyURI”
            minOccurs=”0” maxOccurs=”unbounded”/>
    </xs:sequence>
</xs:complexType>

<xs:complexType name=”generateType”>
    <xs:sequence>
        <xs:element name=”subjectref” type=”xs:anyURI”
            minOccurs=”1” maxOccurs=”1”/>
        <xs:element name=”event” type=”eventType”
            minOccurs=”1” maxOccurs=”unbounded”/>
   </xs:sequence>
</xs:complexType>

<xs:complexType name=”subscribeType”>
   <xs:sequence>
      <xs:element name=”subjectref” type=”xs:anyURI”
         minOccurs=”1” maxOccurs=”1”/>
      <xs:element name=”eventbodytyperef” type=”xs:anyURI”
         minOccurs=”1” maxOccurs=”unbounded”/>
   </xs:sequence>
</xs:complexType>

<xs:complexType name=”unsubscribeType”>
   <xs:sequence>
      <xs:element name=”subjectref” type=”xs:anyURI”
         minOccurs=”1” maxOccurs=”1”/>
      <xs:element name=”eventbodytyperef” type=”xs:anyURI”
         minOccurs=”1” maxOccurs=”unbounded”/>
   </xs:sequence>
</xs:complexType>

<xs:complexType name=”eventType”>
   <xs:sequence>
      <xs:element name=”headergroup” type=”eventHeaderGroupType”
         minOccurs=”1” maxOccurs=”unbounded”/>
      <xs:element name=”body” type=”eventBodyType”
         minOccurs=”1” maxOccurs=”1”/>
   </xs:sequence>
</xs:complexType>

<xs:complexType name=”eventHeaderGroupType”>
   <xs:sequence>
      <xs:element name=”header” type=”eventHeaderType”
         minOccurs=”0” maxOccurs=”unbounded”/>
   </xs:sequence>
   <xs:attribute name=”owner” type=”xs:anyURI”/>
</xs:complexType>

<xs:complexType name=”eventHeaderType”>
   <xs:attribute name=”name” type=”xs:string”/>
</xs:complexType>

<xs:complexType name=”eventBodyType”>
   <xs:attribute name=”type” type=”xs:anyURI”/>
</xs:complexType>

<xs:complexType name=”eventBodyTypeType”>
   <xs:attribute name=”name” type=”xs:string”/>
```

```
</xs:complexType>

<xs:complexType name=”subjectType”>
   <xs:attribute name=”id” type=”xs:ID”/>
</xs:complexType>

<xs:complexType name=”roleType”>
   <xs:attribute name=”id” type=”xs:ID”/>
</xs:complexType>

<xs:complexType name=”assignType”>
   <xs:sequence>
      <xs:element name=”roleref” type=”xs:anyURI”
          minOccurs=”0” maxOccurs=”unbounded”/>
    </xs:sequence>
   <xs:attribute name=”subjectref” type=”xs:anyURI”/>
</xs:complexType>

<xs:complexType name=”unassignType”>
   <xs:sequence>
      <xs:element name=”roleref” type=”xs:anyURI”
          minOccurs=”0” maxOccurs=”unbounded”/>
    </xs:sequence>
   <xs:attribute name=”subjectref” type=”xs:anyURI”/>
</xs:complexType>

<xs:complexType name=”accesscontrolpolicyType”>
   <xs:sequence>
      <xs:element name=”label” type=”labelType”/>
    </xs:sequence>
   <xs:attribute name=”ownerref” type=”xs:anyURI”/>
   <xs:attribute name=”ID” type=”xs:ID”/>
</xs:complexType>

<xs:simpleType name=”labelType”>
  <xs:restriction base=”xs:String”/>
</xs:simpleType>

<xs:complexType name=”accesscontrolruleType”>
  <xs:sequence>
     <xs:element name=”policyref” type=”xs:anyURI”
        minOccurs=”1” maxOccurs=”unbounded”/>
     <xs:element name=”condition” type=”conditionType”
        minOccurs=”1” maxOccurs=”1”/>
   </xs:sequence>
  <xs:attribute name=”operation” type=”xs:operationList”
     use=”required”/>
  <xs:attribute name=”permission” type=”xs:permissionList”
```

```
       use=”required”/>
  </xs:complexType>

  <xs:complexType name=”conditionType”>
     <xs:sequence>
        <xs:element name=”roleref” type=”xs:anyURI”
            minOccurs=”0” maxOccurs=”unbounded”/>
        <xs:element name=”subjectref” type=”xs:anyURI”
            minOccurs=”0” maxOccurs=”unbounded”/>
        <xs:element name=”expression” type=”expressionType”
            minOccurs=”1” maxOccurs=”1”/>
     </xs:sequence>
   </xs:complexType>

   <xs:simpleType name=”operationList”>
     <xs:restriction base=”xs:string”>
        <xs:enumeration value=”publish”/>
        <xs:enumeration value=”unpublish”/>
        <xs:enumeration value=”subscribe”/>
        <xs:enumeration value=”unsubscribe”/>
        <xs:enumeration value=”generate”/>
        <xs:enumeration value=”assign”/>
        <xs:enumeration value=”unassign”/>
        <xs:enumeration value=”activate”/>
        <xs:enumeration value=”deactivate”/>
     </xs:restriction>
  </xs:simpleType>

  <xs:simpleType name=”permissionList”>
     <xs:restriction base=”xs:string”>
        <xs:enumeration value=”publish”/>
        <xs:enumeration value=”unpublish”/>
        <xs:enumeration value=”subscribe”/>
        <xs:enumeration value=”unsubscribe”/>
        <xs:enumeration value=”generate”/>
        <xs:enumeration value=”assign”/>
        <xs:enumeration value=”unassign”/>
        <xs:enumeration value=”activate”/>
        <xs:enumeration value=”deactivate”/>
     </xs:restriction>
  </xs:simpleType>

  <xs:complexType name=”expressionType”/>

</xs:schema>
```

Laboratory experiment definitions

```
<efsoc xmlns:xs = ”http://www.w3.org/2001/XMLSchema”>

  <!-- subject definitions -->
  <subject ID=”john”><name>John</name></subject>
  <subject ID=”mark”><name>Mark</name></subject>
  <subject ID=”mary”><name>Mary</name></subject>
  <subject ID=”sue”><name>Sue</name>
     <floor>oncology</floor>
  </subject>

  <subject ID=”billingservice”>
    <wsdl>http://localhost/ws/billingservice</wsdl>
  </subject>

  <subject ID=”chartingservice”>
    <wsdl>http://localhost/ws/chartingservice</wsdl>
  </subject>

  <subject ID=”labservice”>
    <wsdl>http://localhost/ws/labservice</wsdl>
  </subject>

  <subject ID=”radioservice”>
    <wsdl>http://localhost/ws/radioservice</wsdl>
  </subject>

  <subject ID=”pharmaservice”>
    <wsdl>http://localhost/ws/pharmaservice</wsdl>
  </subject>

  <subject ID=”insuranceservice”>
    <wsdl>http://localhost/ws/insuranceservice</wsdl>
  </subject>
  <!-- role and role attribute definitions -->
  <role ID=”physician”/>
  <role ID=”nurse”/>
  <role ID=”patient”/>

  <roleattributetype ID=”patient” roleref=”physician”/>
  <roleattributetype ID=”floor” roleref=”nurse”/>

  <!-- event body types -->

  <eventbodytype ID=”updatechart”>
    <xs:element name=”patient” type=”xs:string”/>
    <xs:element name=”medicaldata” type=”xs:string”/>
  </eventbodytype>
```

```
<eventbodytype ID=”inspectchart”>
  <xs:element name=”patient” type=”xs:cname”/>
</eventbodytype>

<eventbodytype ID=”chart”>
  <xs:element name=”patient” type=”xs:cname”/>
  <xs:element name=”medicaldata” type=”xs:string”/>
</eventbodytype>

<eventbodytype ID=”orderlaboratorytest”>
  <xs:element name=”test” type=”xs:string”/>
  <xs:element name=”patient” type=”xs:string”/>
  <xs:element name=”reason” type=”xs:string”
              minOccurs=”0”/>
</eventbodytype>

<eventbodytype ID=”retrievetestresults”>
  <xs:element name=”test” type=”xs:string”/>
  <xs:element name=”patient” type=”xs:string”/>
</eventbodytype>

<eventbodytype ID=”laboratorytest”>
  <xs:element name=”test” type=”xs:string”/>
  <xs:element name=”patient” type=”xs:string”/>
  <xs:element name=”results” type=”xs:string”/>
</eventbodytype>

<eventbodytype ID=”prescribemedication”>
  <xs:element name=”patient” type=”xs:string”/>
  <xs:element name=”drugname” type=”xs:string”/>
  <xs:element name=”dosage” type=”xs:string”/>
  <xs:element name=”numberofdoses” type=”xs:integer”/>
</eventbodytype>

<eventbodytype ID=”dispensemedication”>
  <xs:element name=”patient” type=”xs:string”/>
  <xs:element name=”drugname” type=”xs:string”/>
  <xs:element name=”dosage” type=”xs:string”/>
  <xs:element name=”numberofdoses” type=”xs:integer”/>
</eventbodytype>

<!−− role assignments and role attribute assignments −−>
<assign ID=”ra1”>
   <subjectref>john</subjectref>
   <roleref>physician</roleref>
</assign>

<assign ID=”ra2”>
```

```
     <subjectref>mark</subjectref>
     <roleref>physician </roleref>
  </assign>

  <roleattributevalue ID=”rav1”
     roleattributetyperef=”patient”
     roleassignment = ”ra1”
     value=”sue”/>

<assign ID=”ra3”>
     <subjectref>mary</subjectref>
     <roleref>nurse </roleref>
  </assign>

  <roleattributevalue ID=”rav2”
     roleattributetyperef=”floor”
     roleassignment=”ra3”
     value=”oncology”/>

  <assign ID=”ra4”>
     <subjectref>sue</subjectref>
     <roleref>patient </roleref>
  </assign>

   <!−− Charting service access control policy −−>
   <accesscontrolpolicy ID=”chartingpolicy”
                        ownerref=”chartingservice”/>

   <accesscontrolrule ID=”chartingrule1”
                      policyref=”chartingpolicy”
                      operation=”send”
                      permission=”permit”>
      <principal>
         <roleref>physician </roleref>
         <roleref>nurse </roleref>
      </principal>
      <condition>
/efsoc/event[eventbody[@eventbodytype=”inspectchart”]]
      </condition>
   </accesscontrolrule>

   <accesscontrolrule ID=”chartingrule2”
                      policyref=”chartingpolicy”
                      operation=”send”
                      permission=”permit”>
      <principal>
         <roleref>physician </roleref>
      </principal>
      <condition>
/efsoc/event[
   eventbody[@eventbodytype='updatechart'] and
   eventheader[@name='sender'] = /efsoc/assign[
      @ID=/efsoc/roleattributevalue[
         @roleattributetyperef='patient' and
         @value = /efsoc/event/eventbody/patient and
         @roleassignment = /efsoc/assign[
            roleref='physician'
         ]/@ID
      ]/@roleassignment
   ]/subjectref
]</condition>
   </accesscontrolrule>

   <accesscontrolrule ID="chartingrule3"
                      policyref="chartingpolicy"
                      operation="send"
                      permission="permit">
      <principal>
         <roleref>nurse</roleref>
      </principal>
      <condition>
/efsoc/event[
   eventbody[@eventbodytype='updatechart'] and
   eventheader[@name='sender'] = /efsoc/assign[
         @ID=/efsoc/roleattributevalue[
            @roleattributetyperef='floor' and
            @value=/efsoc/subject[
               @ID=/efsoc/event/eventbody/patient
            ]/floor and
            @roleassignment=/efsoc/assign[
               roleref='nurse'
            ]/@ID
         ]/@roleassignment
      ]/subjectref]
      </condition>
   </accesscontrolrule>

   <!-- Laboratory service access control policy -->
   <accesscontrolpolicy ID="laboratorypolicy"
                        ownerref="labservice"/>

   <accesscontrolrule ID="laboratoryrule1"
                      policyref="laboratorypolicy"
                      operation="send"
                      permission="permit">
      <principal>
```

```
            <subjectref>chartingservice</subjectref>
            <roleref>physician</roleref>
            <roleref>nurse</roleref>
        </principal>
        <condition>
/efsoc/event/eventbody[@eventbodytype=”retrievetestresults”]
        </condition>
    </accesscontrolrule>

    <accesscontrolrule ID=”laboratoryrule2”
                       policyref=”laboratorypolicy”
                       operation=”send”
                       permission=”permit”>
        <principal>
            <roleref>physician</roleref>
        </principal>
        <condition>
/efsoc/eventbody[@eventbodytype=”orderlaboratorytest”]
        </condition>
    </accesscontrolrule>

    <accesscontrolrule ID=”laboratoryrule3”
                       policyref=”laboratorypolicy”
                       operation=”receive”
                       permission=”permit”>
        <principal>
            <roleref>physician</roleref>
            <roleref>nurse</roleref>
            <subjectref>chartingservice</subjectref>
        </principal>
        <condition>
/efsoc/eventbody[@eventbodytype=”laboratorytest”]
        </condition>
    </accesscontrolrule>

    <!-- pharamacy policy -->
    <accesscontrolpolicy ID=”pharmapolicy”
                         ownerref=”pharmaservice”/>

    <accesscontrolrule ID=”pharmacyrule1”
                   policyref=”pharmapolicy”
                   operation=”send”
                   permission=”permit”>
        <principal>
            <roleref>physician</roleref>
        </principal>
        <condition>
/efsoc/event/eventbody[@eventbodytype=”prescribemedication”]
```

```
      </condition>
    </accesscontrolrule>

    <accesscontrolrule ID=”pharmacyrule2”
                       policyref=”pharmapolicy”
                       operation=”send”
                       permission=”permit”>
      <principal>
        <subjectref>pharmaservice</subjectref>
      </principal>
      <condition>
/efsoc/event/eventbody[@eventbodytype=”dispensemedication”]
  and
/efsoc/event[@ID=/efsoc/eventheader[@name=”causality”]]/body
[@eventbodytype=”prescribemedication”]
  and
/efsoc/event[@ID=/efsoc/eventheader[@name=”causality”]]/body
/patient = /efsoc/event/eventbody/patient
  and
/efsoc/event[@ID=/efsoc/eventheader[@name=”causality”]]/body
/drugname = /efsoc/event/eventbody/drugname
  and
/efsoc/event[@ID=/efsoc/eventheader[@name=”causality”]]/body
/dosage = /efsoc/event/eventbody/dosage
  and
/efsoc/event[@ID=/efsoc/eventheader[@name=”causality”]]/body
/numberofdoses = /efsoc/event/eventbody/numberofdoses
      </condition>
    </accesscontrolrule>

    <!−− Example events that should be accepted −−>
    <event ID=”e1”>
      <eventheader name=”sender”>john</eventheader>
      <eventbody eventbodytype=”updatechart”>
        <patient>sue</patient>
        <medicaldata>..</medicaldata>
     </eventbody>
    </event>

    <event ID=”e2”>
      <eventheader name=”sender”>mary</eventheader>
      <eventbody eventbodytype=”updatechart”>
        <patient>sue</patient>
        <medicaldata>..</medicaldata>
      </eventbody>
    </event>

    <event ID=”e3”>
```

```
      <eventheader name=”sender”>john</eventheader>
      <eventbody eventbodytype=”orderlaboratorytest”>
      ...
      </eventbody>
   </event>

   <event ID=”e4”>
      <eventheader name=”sender”>mary</eventheader>
      <eventbody eventbodytype=”retrievetestresults”>
      ...
      </eventbody>
   </event>

   <event ID=”e5”>
      <eventheader name=”sender”>john</eventheader>
      <eventbody eventbodytype=”prescribemedication”>
         <patient>sue</patient>
         <drugname>ibuprofen</drugname>
         <dosage>600 mg, 3x per day</dosage>
         <numberofdoses>15</numberofdoses>
      </eventbody>
   </event>

   <event ID=”e6”>
      <eventheader name=”sender”>
        pharmacyservice
      </eventheader>
      <eventheader name=”causality”>e1</eventheader>
      <eventbody eventbodytype=”dispensemedication”>
         <patient>sue</patient>
         <drugname>ibuprofen</drugname>
         <dosage>600 mg, 3x per day</dosage>
         <numberofdoses>15</numberofdoses>
      </eventbody>
   </event>
</efsoc>
```

EFSOC Conceptbase definitions

This appendix contains the formal representation of the EFSOC Definition Language (EDL) in Telos. The definitions also include model constraints to ensure the integrity of the model. In addition to the EDL definitions, this chapter also includes the EFSOC Query Language (EQL) and the detailed definitions for the laboratory experiment.

Model definitions

```
{* $Id: efsoc.sml,v 1.4 2005/10/26 13:54:18 kees Exp $ *}
{* $Source: /home/cvs/phd-kees/thesis/draft/src/efsoc.sml,v $ *}

{* Meta class definitions *}
Class Principal in MetaClass end

Class Role in SimpleClass isA Principal with
attribute
    publishedby: Subject;
    roleattribute: RoleAttributeType
constraint
    c1: $ exists s/Subject (this publishedby s) and
          forall s1/Subject s2/Subject
          (this publishedby s1) and (this publishedby s2)
          ==> (s1 == s2) $
end

Class Subject in SimpleClass isA Principal with
attribute
   assigned: Role;
   uses: Role;
   subscribedto: EventBody
constraint
   c1: $ forall r/Role (this uses r) ==> (this assigned r) $
end

Class RoleAttributeType in SimpleClass end
Class RoleAttribute in SimpleClass with
attribute
   roleAssignment: Subject!uses;
   roleAttributeType: RoleAttributeType;
   value: String
end

Class EventClass in MetaClass with
attribute
   header: EventHeader;
   body: EventBody
end

Class EventHeader in MetaClass end

Class EventBody in MetaClass with
attribute
   publishedby: Subject
constraint
   c : $ exists s/Subject (this publishedby s) and
         forall s1,s2/Subject (this publishedby s1) and
         (this publishedby s2) ==> (s1 == s2)$
end

Class EventHeader_Sender in SimpleClass, EventHeader with
attribute
   subject: Subject
end


Class Operation in SimpleClass isA Subject end

Class MessageOperation in SimpleClass isA Operation end

Class RoleOperation in SimpleClass isA Operation end

Token publish in Operation end

Token unpublish in Operation end

Token subscribe in MessageOperation end

Token unsubscribe in MessageOperation end

Token send in MessageOperation end

Token assign in RoleOperation end
Token revoke in RoleOperation end

Token activate in RoleOperation end

Token deactivate in RoleOperation end

Class Permission in SimpleClass end

Token permit in Permission end

Token refuse in Permission end

Class AccessControlPolicy in SimpleClass
with
attribute
   owner: Subject
end

Class AccessControlRule in SimpleClass with
attribute
   operation: Operation;
   condit: Proposition;
   permission: Permission;
   priority: Integer;
   policy: AccessControlPolicy;
   principal: Principal
end

SDConstraint in SimpleClass with
attribute
   role: Role
end

DSDConstraint isA SDConstraint end
SSDConstraint isA SDConstraint end

DSDEnforcer in Class with
constraint c:
   $ forall s/Subject r1,r2/Role ( s uses r1) and (s uses r2) and
      not (r1 == r2)
      ==>
      not exists f/DSDConstraint (f role r1) and (f role r2)
   $
end

SSDEnforcer in Class with
constraint c:
   $ forall s/Subject r1,r2/Role ( s assigned r1) and (s assigned r2) and
```

```
      not (r1 == r2)
      ==>
      not exists f/SSDConstraint (f role r1) and (f role r2)
   $
end
```

Query language

```
{* $Id$ *}

{**** FIRST ORDER QUERIES ****}
Token EventsByType in GenericQueryClass isA Proposition with
parameter
    eventType : EventBody
constraint
    c: $(~this in ~eventType)$
end

Token SubjectsByRole in GenericQueryClass isA Subject with
parameter
    role : Role
constraint
    c: $(~this assigned ~role)$
end

Token RolesBySubject in GenericQueryClass isA Role with
parameter
   subject: Subject
constraint
   c: $(~subject assigned ~this)$
end

Token SubjectsBySubscription in GenericQueryClass
   isA Subject with
parameter
    event : EventBody
constraint
    c : $ (~this subscribedto ~event)
    $
end

Token EventsBySubscriber in GenericQueryClass
   isA EventBody with
parameter
    subject : Subject
constraint
    c : $ (~subject subscribedto ~this)
    $
```

```
end

Token RoleWithAttribute in GenericQueryClass
   isA Role with
parameter
   attribute : RoleAttributeType
constraint
   c : $ (~this roleattribute ~attribute) $
end

Token ActiveRolesOfSubject in GenericQueryClass
   isA Role with
parameter
   subject: Subject
constraint
   c: $(~subject uses ~this)$
end

{***** SECOND ORDER *****}

Token SubjectWithRoleAttributeValue in GenericQueryClass
   isA Subject with
parameter
   attr: RoleAttributeType;
   value: String
constraint
   c: $ exists ra/RoleAttribute u/Subject!uses
        (ra roleAttributeType ~attr) and
        (ra value ~value) and
        (ra roleAssignment u) and
        Ai(~this, uses, u)
      $
end

Token SendersOfEventType in GenericQueryClass
   isA Subject with
parameter
   ebt: EventClass
constraint
   c: $ exists e/Individual s/EventHeader_Sender
        (e in ~ebt) and
        (e sender s) and
        (s subject ~this)
      $
end
```

Laboratory experiment definitions

```
Token john in Subject end
Token mark in Subject end
Token sue in Subject end
Token mary in Subject end

Token admin in Subject end

Token physician in Role with
   publishedby p: admin
   roleattribute r: patientAttr
end

Token nurse in Role with
   publishedby p: admin
end

Token patient in Role with
   publishedby p: admin
end

Token john in Subject with
   assigned a: physician
   uses u: physician
end

Token mary in Subject with
   assigned a: nurse
   uses u: nurse
end

Token sue in Subject with
   assigned a: patient
   attribute name: ”sue”;
             floor: ”oncology”

end

Token mark in Subject with
   assigned a: physician
end

Token BillingService in Subject end

Token ChartingService in Subject with
subscribedto
    s1: InspectChartBody
end
```

```
Token LaboratoryService in Subject end

Token RadiologyService in Subject end

Token PharamacyService in Subject end

Token InsuranceSerice in Subject end

Token patientAttr in RoleAttributeType end
Token floorAttr in RoleAttributeType end

Token roleattr1 in RoleAttribute with
   roleAssignment ra1: john!u
   roleAttributeType rat1: patientAttr
   value v1: ”sue”
end

Token roleattr2 in RoleAttribute with
   roleAssignment ra1: mary!u
   roleAttributeType rat1: floorAttr
   value v1: ”oncology”
end

Token UpdateChartBody in EventBody with
attribute
   patient: String;
   medicaldata:String
publishedby
   p: admin
end

Token InspectChartBody in EventBody with
attribute
   patient: String
publishedby
   p: admin
end

Token ChartBody in EventBody with
attribute
   patient: String;
   medicaldata: String
publishedby
   p: admin
end

Token OrderLaboratoryTestBody in EventBody with
attribute
```

```
    test: String;
    patient: String;
    reason: String
publishedby
   p: admin
end

Token RetrieveTestResultBody in EventBody with
attribute
    test: String;
    patient: String
publishedby
   p: admin
end

Token LaboratoryTestBody in EventBody with
attribute
     test: String;
    patient: String;
    results: String
publishedby
   p: admin
end

Token PrescribeMedicationBody in EventBody with
attribute
    patient: String;
   drugname: String;
   dosage: String;
   numberofDoses: Integer
publishedby
   p: admin
end

Token DispenseMedicationBody in EventBody with
attribute
    patient: String;
   drugname: String;
   dosage: String;
   numberofDoses: Integer
publishedby
   p: admin
end

Token ChartingPolicy in AccessControlPolicy with
   owner o: ChartingService
end
```

```
Token ChartingRule1 in AccessControlRule with
   policy      pol: ChartingPolicy
   operation   op1: send
   permission perm1: permit
   principal   princ1: physician;
               princ2: nurse
end

Class InspectChartEvent in EventClass with
   header sender: EventHeader_Sender
   body bdy: InspectChartBody

end

Token inspectChartBodySue in InspectChartBody with
   patient p: "sue"
end

Token inspectChartSenderHeaderSue in EventHeader_Sender
with
  subject s: sue
end

Token e1 in InspectChartEvent with
   sender s: inspectChartSenderHeaderSue
   bdy b: inspectChartBodySue
end

Token prescribeMedicationBodySue in PrescribeMedicationBody
with
   patient  p: "sue"
   drugname drug: "ibuprofin"
   dosage   d: "600mg 3 times per day after meal"
   numberofDoses num: 15
end

{* EOF *}
```

Samenvatting

Service-Oriented Computing (SOC) is een relatief nieuwe stroming binnen het ontwerpen en integreren van informatiesystemen. De stroming manifesteert zich in oplossingen die gebaseerd zijn op de *servicegerichte architectuur* (Engels: *Service-Oriented Architecture*, SOA) en die *web services* worden genoemd.

In de SOA is het beveiligingsaspect en met name het controleren van toegang tot services, onderbelicht gebleven. Het onderzoek dat gepresenteerd wordt in dit proefschrift heeft dan ook als centraal thema: toegangscontrole en servicegerichte architecturen.

De gevolgde onderzoeksmethodologie omvat een aantal stappen dat elkaar beïnvloedt. Allereerst is begonnen met de precieze formulering van het probleem dat in deze studie wordt onderzocht. Deze formulering is:

> Ontwerp, ontwikkel en valideer een referentiekader voor toegangscontrole voor servicegerichte architecturen. Het referentiekader dient toepasbaar te zijn bij architecturen die zich binnen één enkele organisatie bevinden, maar ook op architecturen die organisatiegrenzen overschrijden.

Om dit doel te bereiken wordt een aantal stappen uitgevoerd, zoals aangegeven in figuur 1.3. Deze stappen omvatten het uitvoeren van een (doorlopende) literatuurstudie, gevolgd door het creëren van het ontwerp van een referentiekader. Het ontwerp is gebaseerd op de bevindingen van deze literatuurstudie.

Het referentiekader wordt vervolgens empirisch gevalideerd met behulp van het uitvoeren van een casestudie en door de uitvoering van een gecontroleerd laboratoriumexperiment. In deze studie, die plaats vond in het Amerikaanse Northside ziekenhuis, zijn informatiestromen in kaart gebracht rond de opname en behandeling van een chirurgisch patiënt.

De resultaten van de studie zijn vervolgens gebruikt om de *volledigheid* en de *nauwkeurigheid* van het voorgestelde raamwerk te toetsen. Op basis van de concepten die in het referentiekader zijn geïntroduceerd is vervolgens een technisch prototype gemaakt. Dit prototype is gepopuleerd met voorbeelden die zijn afgeleid uit de studie in Northside. Bij het populeren van het prototype werd met name gezocht naar een antwoord op de vraag of gewenste situaties werden geaccepteerd door het prototype en ongewenste situaties werden geweigerd. De interpretatie van de studie en de beoordeling van situaties als gewenst of ongewenst is door ons zelf uitgevoerd. Met het populeren van het prototype werd aangetoond dat het prototype, alsmede het referentiekader waarop het prototype is gebaseerd, *operationeel consistent* is.

De belangrijkste onderzoeksvragen worden als volgt beantwoord:

1. **Vraag**: Wat is de state-of-the-art van toegangscontrole?
   Antwoord: De huidige state-of-the-art van toegangscontrole omvat rolgebaseerde toegangscontrole en de specificaties van de XACML- en SAML-standaarden. Verder spelen ook de specificaties van de WS-* beveiligingsstandaarden een rol.

2. **Vraag**: Is de huidige state-of-the-art van toegangscontrole adequaat bruikbaar voor servicegerichte architecturen?
   Antwoord: De huidige state-of-the-art van toegangscontrole is onvoldoende. De overwegingen om tot dit oordeel te komen zijn dat 1) RBAC de betekenis van ‘toestemming’ niet definieert en 2) dat RBAC gebaseerd is op de aanname van centraal beheerde toegangscontrole.

   De WS-* specificaties zijn niet voldoende ontwikkeld voor grootschalig gebruik. XACML biedt weliswaar een referentiekader, maar omvat geen conceptueel methodologisch raamwerk.

Op basis van het referentiekader dat in dit proefschrift wordt geïntroduceerd kunnen onze bijdragen aan toegangscontrole voor servicegerichte architecturen als volgt worden samengevat:

1. In het onderzoek introduceren wij een toegangscontrolemodel voor servicegerichte architecturen dat decentraal wordt beheerd, maar centraal wordt afgedwongen.

2. Het toegangscontrolemodel van EFSOC kan worden gecategoriseerd als discretionair en rolgebaseerd.

3. Het toegangscontrolemodel van EFSOC voorziet in een gemeenschappelijk methodologisch referentiekader en voorziet in een referentiearchitectuur voor decentraal beheerde, maar centraal uitgevoerde, discretionaire geparameteriseerde rolgebaseerde toegangscontrole.

Part IV

Reference

SIKS Dissertation Series

Other books that have appeared in the SIKS Dissertation Series:

1998-1 Johan van den Akker (CWI) *DEGAS - An Active, Temporal Database of Autonomous Objects*

1998-2 Floris Wiesman (UM) *Information Retrieval by Graphically Browsing Meta-Information*

1998-3 Ans Steuten (TUD) *A Contribution to the Linguistic Analysis of Business Conversations within the Language/Action Perspective*

1998-4 Dennis Breuker (UM) *Memory versus Search in Games*

1998-5 E.W.Oskamp (RUL) *Computerondersteuning bij Straftoemeting*

1999-1 Mark Sloof (VU) *Physiology of Quality Change Modelling; Automated modelling of Quality Change of Agricultural Products*

1999-2 Rob Potharst (EUR) *Classification using decision trees and neural nets*

1999-3 Don Beal (UM) *The Nature of Minimax Search*

1999-4 Jacques Penders (UM) *The practical Art of Moving Physical Objects*

1999-5 Aldo de Moor (KUB) *Empowering Communities: A Method for the Legitimate User-Driven Specification of Network Information Systems*

1999-6 Niek J.E. Wijngaards (VU) *Re-design of compositional systems*

1999-7 David Spelt (UT) *Verification support for object database design*

1999-8 Jacques H.J. Lenting (UM) *Informed Gambling: Conception and Analysis of a Multi-Agent Mechanism for Discrete Reallocation.*

2000-1 Frank Niessink (VU) *Perspectives on Improving Software Maintenance*

2000-2 Koen Holtman (TUE) *Prototyping of CMS Storage Management*

2000-3 Carolien M.T. Metselaar (UVA) *Sociaal-organisatorische gevolgen van kennistechnologie; een procesbenadering en actorperspectief.*

2000-4 Geert de Haan (VU) *ETAG, A Formal Model of Competence Knowledge for User Interface Design*

2000-5 Ruud van der Pol (UM) *Knowledge-based Query Formulation in Information Retrieval.*

2000-6 Rogier van Eijk (UU) *Programming Languages for Agent Communication*

2000-7 Niels Peek (UU) *Decision-theoretic Planning of Clinical Patient Management*

2000-8 Veerle Coup’e (EUR) *Sensitivity Analyis of Decision-Theoretic Networks*

2000-9 Florian Waas (CWI) *Principles of Probabilistic Query Optimization*

2000-10 Niels Nes (CWI) *Image Database Management System Design Considerations, Algorithms and Architecture*

2000-11 Jonas Karlsson (CWI) *Scalable Distributed Data Structures for Database Management*

2001-1 Silja Renooij (UU) *Qualitative Approaches to Quantifying Probabilistic Networks*

2001-2 Koen Hindriks (UU) *Agent Programming Languages: Programming with Mental Models*

2001-3 Maarten van Someren (UvA) *Learning as problem solving*

2001-4 Evgueni Smirnov (UM) *Conjunctive and Disjunctive Version Spaces with Instance-Based Boundary Sets*

2001-5 Jacco van Ossenbruggen (VU) *Processing Structured Hypermedia: A Matter of Style*

2001-6 Martijn van Welie (VU) *Task-based User Interface Design*

2001-7 Bastiaan Schonhage (VU) *Diva: Architectural Perspectives on Information Visualization*

2001-8 Pascal van Eck (VU) *A Compositional Semantic Structure for Multi-Agent Systems Dynamics.*

2001-9 Pieter Jan ´t Hoen (RUL) *Towards Distributed Development of Large Object-Oriented Models, Views of Packages as Classes*

2001-10 Maarten Sierhuis (UvA) *Modeling and Simulating Work Practice, BRAHMS: a multiagent modeling and simulation language for work practice analysis and design*

2001-11 Tom M. van Engers (VUA) *Knowledge Management: The Role of Mental Models in Business Systems Design*

2002-01 Nico Lassing (VU) *Architecture-Level Modifiability Analysis*

2002-02 Roelof van Zwol (UT) *Modelling and searching web-based document collections*

2002-03 Henk Ernst Blok (UT) *Database Optimization Aspects for Information Retrieval*

2002-04 Juan Roberto Castelo Valdueza (UU) *The Discrete Acyclic Digraph Markov Model in Data Mining*

2002-05 Radu Serban (VU) *The Private Cyberspace Modeling Electronic Environments inhabited by Privacy-concerned Agents*

2002-06 Laurens Mommers (UL) *Applied legal epistemology; Building a knowledge-based ontology of the legal domain*

2002-07 Peter Boncz (CWI) *Monet: A Next-Generation DBMS Kernel For Query-Intensive Applications*

2002-08 Jaap Gordijn (VU) *Value Based Requirements Engineering: Exploring Innovative E-Commerce Ideas*

2002-09 Willem-Jan van den Heuvel(KUB) *Integrating Modern Business Applications with Objectified Legacy Systems*

2002-10 Brian Sheppard (UM) *Towards Perfect Play of Scrabble*

2002-11 Wouter C.A. Wijngaards (VU) *Agent Based Modelling of Dynamics: Biological and Organisational Applications*

2002-12 Albrecht Schmidt (UvA) *Processing XML in Database Systems*

2002-13 Hongjing Wu (TUE) *A Reference Architecture for Adaptive Hypermedia Applications*

2002-14 Wieke de Vries (UU) *Agent Interaction: Abstract Approaches to Modelling, Programming and Verifying Multi-Agent Systems*

2002-15 Rik Eshuis (UT) *Semantics and Verification of UML Activity Diagrams for Workflow Modelling*

2002-16 Pieter van Langen (VU) *The Anatomy of Design: Foundations, Models and Applications*

2002-17 Stefan Manegold (UVA) *Understanding, Modeling, and Improving Main-Memory Database Performance*

2003-01 Heiner Stuckenschmidt (VU) *Ontology-Based Information Sharing in Weakly Structured Environments*

2003-02 Jan Broersen (VU) *Modal Action Logics for Reasoning About Reactive Systems*

2003-03 Martijn Schuemie (TUD) *Human-Computer Interaction and Presence in Virtual Reality Exposure Therapy*

2003-04 Milan Petkovic (UT) *Content-Based Video Retrieval Supported by Database Technology*

2003-05 Jos Lehmann (UVA) *Causation in Artificial Intelligence and Law - A modelling approach*

2003-06 Boris van Schooten (UT) *Development and specification of virtual environments*

2003-07 Machiel Jansen (UvA) *Formal Explorations of Knowledge Intensive Tasks*

2003-08 Yongping Ran (UM) *Repair Based Scheduling*

2003-09 Rens Kortmann (UM) *The resolution of visually guided behaviour*

2003-10 Andreas Lincke (UvT) *Electronic Business Negotiation: Some experimental studies on the interaction between medium, innovation context and culture*

2003-11 Simon Keizer (UT) *Reasoning under Uncertainty in Natural Language Dialogue using Bayesian Networks*

2003-12 Roeland Ordelman (UT) *Dutch speech recognition in multimedia information retrieval*

2003-13 Jeroen Donkers (UM) *Nosce Hostem - Searching with Opponent Models*

2003-14 Stijn Hoppenbrouwers (KUN) *Freezing Language: Conceptualisation Processes across ICT-Supported Organisations*

2003-15 Mathijs de Weerdt (TUD) *Plan Merging in Multi-Agent Systems*

2003-16 Menzo Windhouwer (CWI) *Feature Grammar Systems - Incremental Maintenance of Indexes to Digital Media Warehouses*

2003-17 David Jansen (UT) *Extensions of Statecharts with Probability, Time, and Stochastic Timing*

2003-18 Levente Kocsis (UM) *Learning Search Decisions*

2004-01 Virginia Dignum (UU) *A Model for Organizational Interaction: Based on Agents, Founded in Logic*

2004-02 Lai Xu (UvT) *Monitoring Multi-party Contracts for E-business*

2004-03 Perry Groot (VU) *A Theoretical and Empirical Analysis of Approximation in Symbolic Problem Solving*

2004-04 Chris van Aart (UVA) *Organizational Principles for Multi-Agent Architectures*

2004-05 Viara Popova (EUR) *Knowledge discovery and monotonicity*

2004-06 Bart-Jan Hommes (TUD) *The Evaluation of Business Process Modeling Techniques*

2004-07 Elise Boltjes (UM) *Voorbeeldig onderwijs; voorbeeldgestuurd onderwijs, een opstap naar abstract denken, vooral voor meisjes*

2004-08 Joop Verbeek(UM) *Politie en de Nieuwe Internationale Informatiemarkt, Grensregionale politionele gegevensuitwisseling en digitale expertise*

2004-09 Martin Caminada (VU) *For the Sake of the Argument; explorations into argument-based reasoning*

2004-10 Suzanne Kabel (UVA) *Knowledge-rich indexing of learning-objects*

2004-11 Michel Klein (VU) *Change Management for Distributed Ontologies*

2004-12 The Duy Bui (UT) *Creating emotions and facial expressions for embodied agents*

2004-13 Wojciech Jamroga (UT) *Using Multiple Models of Reality: On Agents who Know how to Play*

2004-14 Paul Harrenstein (UU) *Logic in Conflict. Logical Explorations in Strategic Equilibrium*

2004-15 Arno Knobbe (UU) *Multi-Relational Data Mining*

2004-16 Federico Divina (VU) *Hybrid Genetic Relational Search for Inductive Learning*

2004-17 Mark Winands (UM) *Informed Search in Complex Games*

2004-18 Vania Bessa Machado (UvA) *Supporting the Construction of Qualitative Knowledge Models*

2004-19 Thijs Westerveld (UT) *Using generative probabilistic models for multimedia retrieval*

2004-20 Madelon Evers (Nyenrode) *Learning from Design: facilitating multidisciplinary design teams*

2005-01 Floor Verdenius (UVA) *Methodological Aspects of Designing Induction-Based Applications*

2005-02 Erik van der Werf (UM)) *AI techniques for the game of Go*

2005-03 Franc Grootjen (RUN) *A Pragmatic Approach to the Conceptualisation of Language*

2005-04 Nirvana Meratnia (UT) *Towards Database Support for Moving Object data*

2005-05 Gabriel Infante-Lopez (UVA) *Two-Level Probabilistic Grammars for Natural Language Parsing*

2005-06 Pieter Spronck (UM) *Adaptive Game AI*

2005-07 Flavius Frasincar (TUE) *Hypermedia Presentation Generation for Semantic Web Information Systems*

2005-08 Richard Vdovjak (TUE) *A Model-driven Approach for Building Distributed Ontology-based Web Applications*

2005-09 Jeen Broekstra (VU) *Storage, Querying and Inferencing for Semantic Web Languages*

2005-10 Anders Bouwer (UVA) *Explaining Behaviour: Using Qualitative Simulation in Interactive Learning Environments*

2005-11 Elth Ogston (VU) *Agent Based Matchmaking and Clustering - A Decentralized Approach to Search*

2005-12 Csaba Boer (EUR) *Distributed Simulation in Industry*

2005-13 Fred Hamburg (UL) *Een Computermodel voor het Ondersteunen van Euthanasiebeslissingen*

2005-14 Borys Omelayenko (VU) *Web-Service configuration on the Semantic Web; Exploring how semantics meets pragmatics*

2005-15 Tibor Bosse (VU) *Analysis of the Dynamics of Cognitive Processes*

2005-16 Joris Graaumans (UU) *Usability of XML Query Languages*

2005-17 Boris Shishkov (TUD) *Software Specification Based on Re-usable Business Components*

2005-18 Danielle Sent (UU) *Test-selection strategies for probabilistic networks*

2005-19 Michel van Dartel (UM) *Situated Representation*

2005-20 Cristina Coteanu (UL) *Cyber Consumer Law, State of the Art and Perspectives*

2005-21 Wijnand Derks (UT) *Improving Concurrency and Recovery in Database Systems by Exploiting Application Semantics*

2006-01 Samuil Angelov (TUE) *Foundations of B2B Electronic Contracting*

2006-02 Cristina Chisalita (VU) *Contextual issues in the design and use of information technology in organizations*

2006-03 Noor Christoph (UVA) *The role of metacognitive skills in learning to solve problems*

2006-04 Marta Sabou (VU) *Building Web Service Ontologies*

2006-05 Cees Pierik (UU) *Validation Techniques for Object-Oriented Proof Outlines*

2006-06 Ziv Baida (VU) *Software-aided Service Bundling - Intelligent Methods & Tools for Graphical Service Modeling*

2006-07 Marko Smiljanic (UT) *XML schema matching – balancing efficiency and effectiveness by means of clustering*

2006-08 Eelco Herder (UT) *Forward, Back and Home Again - Analyzing User Behavior on the Web*

2006-09 Mohamed Wahdan (UM) *Automatic Formulation of the Auditor's Opinion*

2006-10 Ronny Siebes (VU) *Semantic Routing in Peer-to-Peer Systems*

2006-11 Joeri van Ruth (UT) *Flattening Queries over Nested Data Types*

2006-12 Bert Bongers (VU) *Interactivation - Towards an e-cology of people, our technological environment, and the arts*

2006-13 Henk-Jan Lebbink (UU) *Dialogue and Decision Games for Information Exchanging Agents*

2006-14 Johan Hoorn (VU) *Software Requirements: Update, Upgrade, Redesign - towards a Theory of Requirements Change*

2006-15 Rainer Malik (UU) *CONAN: Text Mining in the Biomedical Domain*

2006-16 Carsten Riggelsen (UU) *Approximation Methods for Efficient Learning of Bayesian Networks*

2006-17 Stacey Nagata (UU) *User Assistance for Multitasking with Interruptions on a Mobile Device*

2006-18 Valentin Zhizhkun (UVA) *Graph transformation for Natural Language Processing*

2006-19 Birna van Riemsdijk (UU) *Cognitive Agent Programming: A Semantic Approach*

2006-20 Marina Velikova (UvT) *Monotone models for prediction in data mining*

2006-21 Bas van Gils (RUN) *Aptness on the Web*

2006-22 Paul de Vrieze (RUN) *Fundaments of Adaptive Personalisation*

2006-23 Ion Juvina (UU) *Development of Cognitive Model for Navigating on the Web*

2006-24 Laura Hollink (VU) *Semantic Annotation for Retrieval of Visual Resources*

2006-25 Madalina Drugan (UU) *Conditional log-likelihood MDL and Evolutionary MCMC*

2006-26 Vojkan Mihajlovic (UT) *Score Region Algebra: A Flexible Framework for Structured Information Retrieval*

2006-27 Stefano Bocconi (CWI) *Vox Populi: generating video documentaries from semantically annotated media repositories*

2006-28 Borkur Sigurbjornsson (UVA) *Focused Information Access using XML Element Retrieval*

List of Figures

1.1 EFSOC overview . . . 3

1.2 Positioning of research . . . 5

1.3 Research methodology . . . 7

1.4 Thesis outline . . . 11

2.1 Public Key Infrastructure CA Hierarchy . . . 19

2.2 Web of Trust . . . 21

2.3 Role-Based Access Control (Sandhu et al., 1996) . . . 25

2.4 Publishing, Discovering and Invoking Services . . . 29

2.5 The Extended SOA (Papazoglou and Georgakapoulos, 2003) . . . 29

2.6 The IBM and Microsoft Security Specification for Web Services . . . 33

2.7 XACML Dataflow Diagram (XACML, 2005) . . . 37

2.8 Brokered publish-subscribe interaction . . . 43

3.1 Case study elicitation process . . . 56

3.2 Northside Hospital Activity diagram . . . 59

3.3 Running example . . . 63

3.4 The EFSOC Conceptual Model . . . 65

3.5 EFSOC Event Model . . . 66

3.6 Subscribe to event activity diagram . . . 67

3.7 Sending events . . . 68

3.8 EFSOC Role Model 71
3.9 Access Control Model 72
3.10 Taking access control decisions 73
3.11 Evaluating access control policies 77
3.12 Evaluating access control rules 78
3.13 EFSOC Service Architecture 79
3.14 EFSOC Event Processing Overview 81
3.15 Pharmacy's access control policy 83
3.16 Charting Service Access Control Policy 84
3.17 EFSOC in relation to the WS Security Roadmap 86
3.18 EFSOC Delegation 89

4.1 EFSOC Languages 92
4.2 An XML event representation 94

5.1 Sample definitions 108

6.1 Role Attribute Example 128

7.1 Prototype reference architecture 136
7.2 Language validator: Access control policy 138
7.3 Language validator: Access control rule evaluation 139
7.4 Instance level definitions in laboratory experiment 141

8.1 Trust relationships spanning service clusters 151
8.2 Trust relationships spanning service clusters combined with service aggregation 152

Bibliography

Bacon, J., Moody, K., and Yao, W. (2002). A Model of OASIS Role-Based Access Control and Its Support for Active Security. *ACM Transactions on Information and System Security*, 5(4):492 – 540.

Bajaj, S., Della-Libera, G., Dixon, B., Dusche, M., Hondo, M., Hur, M., Kaler, C., Lockhart, H., Maruyama, H., Nadalin, A., Nagaratnam, N., Nash, A., Prafullchandra, H., and Shewchuk, J. (2003). Web Services Federation Language (WS-Federation). Technical report.

Bajaj, S. and et al, D. B. (2004). Web Services Policy Framework (WS-Policy). Technical report, BEA Systems, IBM, Microsoft, SAP AG, Sonic Software, Verisign, Inc.

Banavar, G., Chandra, T., Strom, R., and Sturman, D. (1999). A case for message oriented middleware. *Lecture Notes in Computer Science*, 1693:1–18.

Barka, E. S. (2002). *Framework for Role-Based Delegation Models*. PhD thesis, George Mason University.

Bartel, M., Boyer, J., Fox, B., LaMacchia, B., Simon, E., Eastlake, D., Reagle, J., and Solo, D. (2002). XML-Signature Syntax and Processing. W3C Recommendation, W3C. http://www.w3.org/TR/xmldsig-core/.

Bell, D. E. and LaPadula, L. J. (1973). Secure computer systems: Mathematical foundations. Technical report, The MITRE Corporation. MITRE Technical Report 2547, Volume I.

Berglund, A., Boag, S., Chamberlin, D., Fernández, M. F., Kay, M., Robie, J., and Siméon), J. (2005). XML Path Language (XPath) 2.0. W3C Working Draft, W3C. http://www.w3.org/TR/xpath20/.

Biba, K. (1977). Integrity constraints for secure computer systems. Technical report, USAF Electronic System Division, Bedford, Massachusetts. Technical Report ESD-TR76-372.

Botha, R. and Eloff, J. (2001). Separation of duties for access control enforcement in workflow environments. *IBM Systems Journal*, 40(3):666–682.

Brambilla, M., Ceri, S., Passamani, M., and Riccio, A. (2004). Managing Asynchronous Web Services Interactions. In *Proceedings of the IEEE International Conference on Web Services (ICWS'04)*. IEEE.

Bray, T., Hollander, D., and Layman, A. (1999). Namespaces in XML. Technical report, W3C.

Burrows, M., Abadi, M., and Needham, R. (1990). A Logic of Authentication. *ACM Transactions on Computer Systems*, 8(1):18–36.

Ceri, S., Fraternali, P., and Navathe, S. B. (1997). *Designing Database Applications with Objects and Rules*. Series on Database Systems and Applications. Addison-Wesley.

Chappell, D. A. (2004). *Enterprise Service Bus*. O'Reilly Media, Inc. ISBN 0-596-00675-6.

Christensen, E., Curbera, F., Meredith, G., and Weerawarana, S. (2001). Web Services Description Language (WSDL) 1.1. W3C Note, W3C. http://www.w3.org/TR/2001/NOTE-wsdl-20010315.

Clark, D. and Wilson, D. (1987). A comparison of commercial and military computer security policies. In *Proc. of the IEEE Symposium on Security and Privacy*.

Della-Libera, G. and et al, B. D. (2002). Web Services Secure Conversation Language (WS-SecureConversation). Technical report.

deTroyer, O. and Leune, K. (1998). WSDM: A User-Centered Design Method for Web Sites. In *Computer Networks and ISDN Systems, Proceedings of the 7th WWW conference*, pages 85–94. Elsevier.

Eugster, P. T., Felber, P. A., Guerraoui, R., and Kermarrec, A.-M. (2003). The many faces of publish/subscribe. *ACM Computing Surveys*, 35(2):114–131.

Farell, S., Reid, I., Lockhart, H., Orchard, D., Sankar, K., Adams, C., Moses, T., Edwards, N., Pato, J., Blakley, B., Erdos, M., Cantor, S., Morgan, R. B., Chanliau, M., McLaren, C., Knouse, C., Godik, S., Platt, D., Moreh, J., Hodges, J., and Hallam-Baker, P. (2003). Assertions and Protocol for the OASIS Security Assertion Markup Language (SAML) V1.1. Committee specification, OASIS. http://www.oasis-open.org/committees/documents.php?wg_abbr ev=security.

Ferraiolo, D. F., Barkey, J. F., and Kuhn, D. R. (1999). A role-based access control model and reference implementation within a corporate intranet. *ACM Transactions on Information and System Security*, 2(1):34–64.

Gasser, M., Goldstein, A., Kaufman, C., and Lampson, B. (1989). The digital distributed system security architecture. In *Proc. 12th NIST-NCSC National Computer Security Conference*, pages 305–319.

Gavrilla, S. I. and Barkley, J. F. (1998). Formal Specification for Role Based Access Control User/Role and Role/Role Relationship Management. In *Proceedings of the 3rd ACM Workshop on Role-Based Access Control*, pages 81–90.

Giuri, L. and Iglio, P. (1997). Role Templates for Content-Based Access Control. In *Proceedings of RBAC'97*.

Gollmann, D. (2006). *Computer Security*. John Wiley & Sons. ISBN 0-470-86293-9.

Gudgin, M. and et al, A. N. (2001–2005). Web Services Trust Language (WS-Trust). Technical report, Actional Corporation, BEA Systems, Inc., Computer Associates International, Inc., International Business Machines Corporation, Layer 7 Technologies, Microsoft Corporation, Oblix Inc., OpenNetwork Technologies Inc., Ping Identity Corporation, Reactivity Inc., RSA Security Inc., and VeriSign Inc.

Haas, H. and Brown, A. (2004). Web services glossary. Technical report, W3C.

Hamada, T. (1998). Role-Based Access Control in Telecommunication Service Management. In *Proceedings of the 3rd ACM Workshop on Role-Based Access Control*, pages 105–113.

Hansche, S., Berti, J., and Hare, C. (2004). *Official (ISC)^2 Guide to the CISSP Exam*. Auerbach Publications.

Hansen, S. and Fossum, T. (2004). Events Not Equal To GUIs. In *Proceedings of SIGCSE'04*, pages 378–381. ACM.

Harrison, M. A., Ruzzo, W. L., and Ullman, J. D. (1976). Protection in Operation Systems. *Communications of the ACM*, pages 461–471.

Hartman, B., Flinn, D. J., Beznosov, K., and Kawamoto, S. (2003). *Mastering Web Services Security*. Wiley Publishing, Inc. ISBN 0-471-26716-3.

Hayton, R., Bacon, J., and Moody, K. (1998). Access control in an open distributed environment. In *Proceedings of the 1998 IEEE Symposium on Security and Privacy*, pages 3–14.

van den Heuvel, W.-J. (2002). *Integrating Modern Business Applications with Objectified Legacy Systems*. PhD thesis, CentER for Economic Research.

van den Heuvel, W.-J., Leune, K., and Papazoglou, M. P. (2005). EFSOC: A Layered Framework for Developing Secure Interactions between Web-Services. *Distributed and Parallel Databases*, 18:115–145.

HIPAA Privacy Rule (2003). Summary of the HIPAA Privacy Rule. Technical report.

Hsiao, T.-Y., Perng, N.-C., Lo, W., Chang, Y.-S., and Yuan, S.-M. (2003). A new development environment for an event-based distributed system. *Computer Standards & Interfaces*, 25:345–355.

Kelley, D. and Moritz, R. (2006). Best Practice for Building a Security Operations Center. *Information Systems Security*.

Klyne, G. and Newman, C. (2002). Date and Time on the Internet: Timestamps. Technical report, IETF Network Working Group. RFC 3339.

Leune, K., Papazoglou, M., and van den Heuvel, W.-J. (2004a). Specification and Querying Security Constraints in the EFSOC Framework. In *ICSOC '04: Proceedings of the 2nd international conference on Service oriented computing*. ACM Press.

Leune, K., van den Heuvel, W.-J., and Papazoglou, M. (2004b). Exploring a multi-faceted framework for soc: How to develop secure web-service interactions? In *Proceedings of the 14th International Workshop on Research Issues on Data Engineering: Web Services for E-Commerce and E-Government applications*, pages 56 – 61.

Li, N., Mitchell, J. C., and Winsborough, W. H. (2004). Design of a role-based trust-management framework. In *Proceedings of the IEEE Symposium on Security and Privacy*.

Luckham, D. (2002). *The Power of Events. An Introduction to Complex Event Processing in Distributed Enterprise Systems*. Addison-Wesley Press.

Lupu, E. and Sloman, M. (1997). Reconciling Role Based Management and Role Based Access Control. In *Proceedings of RBAC'97*.

M. Krause, H. T. (1999). *Handbook of Information Security Management*. Auerbach Publications, 4th edition edition.

Maheshwari, P., Tang, H., and Liang, R. (2004). Enhancing web services with message-oriented middleware. In *Proceedings of the IEEE International Conference on Web Services (ICWS'04)*. IEEE.

McGuinness, D. L. and van Harmelen, F. (2004). OWL Web Ontology Language Overveiw. W3C Recommendation, W3C. http://www.w3.org/TR/2004/REC-owl-features-20040210/.

Morgan, T. (2002). *Business Rules and Information Systems*. Addison-Wesley. ISBN 0-201-74391-4.

Mühl, G. (2002). *Large-Scale Content-Based Publish/Subscribe Systems*. PhD thesis, Tecnischen Universität Darmstadt.

Mylopoulos, J., Borgida, A., and Koubarakis, M. (1990). Telos: Representing knowledge about information systems. *ACM Transactions on Information Systems*, 8(4).

Neon Orange Book (1987). A guide to understanding discretionary access control in trusted systems. Technical Report Library No. S-228,576, National Computer Security Center.

Orange Book (1985). Trusted computer system evaluation criteria. Technical Report Library No. S225,711, Department of Defense.

Paolucci, M. and Sycara, K. (2003). Autonomous semantic web services. *IEEE Internet Computing*, pages 34–41.

Papazoglou, M. and Georgakapoulos, G. (2003). Introduction to the Special Issue about Service-Oriented Computing. *Communications of the ACM*, 46(10):24–29.

Papazoglou, M. P. and van den Heuvel, W.-J. (2006). Service oriented architectures. *VLDB Journal*, To be published.

Reagle, J. (2002). XML Encryption Requirements. W3C Note, W3C. http://www.w3.org/TR/xml-encryption-req.

Saltzer, J. and Schroeder, M. (1975). The protection of information in computer systems. *Proceedings of the IEEE*, 63(9):1278 – 1308.

Sandhu, R., Coyne, E., Feinstein, H., and Youman, C. (1996). Role-Based Access Control Models. *IEEE Computer*.

Sandhu, R., Ferraiolo, D., and Kuhn, R. (2000). The nist model for role based access control: Towards a unified standard. In *Proceedings, 5th ACM Workshop on Role-Based Access Control*.

Sandhu, R. and Samarati, P. (1996). Authentication, Access Control, and Audit. *ACM Computing Surveys*, 28(1).

Schmidt, M.-T., Hutchison, B., Lambros, P., and Phippen, R. (2005). The Enterprise Service Bus: Making service-oriented architecture real. *IBM Systems Journal*, 44(4):781–797.

Schumacher, M., Fernandez-Buglioni, E., Hybertson, D., Buschmann, F., and Sommerlad, P. (2006). *Security Patterns*. John Wiley & Sons, Ltd.

Suppes, P. (1957). *Introduction to Logic*. Dover Publications.

Tai, S., Mikalsen, T., Rouvellou, I., and Jr., S. M. S. (2002). Conditional messaging: Extending reliable messaging with application conditions. In *Proceedings of the 22nd International Conference on Distributed Computing Systems (ICDCS'02)*. IEEE.

Thompson, H., Beech, D., Maloney, M., and Mendelsohn, N. (2001). Xml schema part 1: Structures. Technical report, W3C. http://www.w3.org/TR/xmlschema-1/.

Ullman, J. D. (1988). *Principles of database and knowledge systems.* Computer Science Press. ISBN 0-7167-8158-1.

UML 2.0 (2005). Unified Modeling Language (UML). Technical report, OMG.

Vinoski, S. (2002). Where is the middleware? *IEEE Internet Computing*, pages 83–85.

Web Services Security Roadmap (2002). Security in a Web Services World: A Proposed Architecture and Roadmap. Technical report, IBM Corporation and Microsoft Corporation.

Welke, R. (1981). IS/DSS: DBMS Support for information systems development. Technical Report ISRAM WP-8105-1.0, McMaster University, Hamilton.

Wolter, R. (2001). Xml web services basics. Technical report, Microsoft Corporation. http://msdn.microsoft.com/webservices/understanding/webserv icebasics/default.aspx.

XACML (2005). eXtensible Access Control Markup Language (XACML). OASIS Standard, OASIS.

XQuery (2005). XQuery1.0: An XML Query Language. Technical report, W3C. Candidate Recommendation.

Yergeau, F., Bray, T., Paoli, J., Sperberg-McQueen, C., and Maler, E. (2004). Extensible Markup Language (XML) 1.0. Technical report, W3C.

Author Index

Abadi, Martin 16
Adams, Carlisle 34, 35
Altunay, M. 51

Bacon, Jean 49
Bacon, J.M. 22
Bajaj, Siddharth 34
Banavar, G. 38
Barka, Ezedin S. 47
Barkey, John F. 24, 69
Barkley, John F. 69
Bartel, Mark 34, 91
Beech, D. 91
Bell, D. Elliott 23
Berglund, Anders 143
Berti, John 49
Bertino, Elisa 52
Beznosov, Konstantin 35
Biba, K.J. 23
Blakley, Bob 34, 35
Boag, Scott 143
Borgida, A. 122
Botha, R.A. 22, 24
Boyer, John 34, 91
Brambilla, Marco 39
Bray, Tim 91
Brown, Allen 31
Brown, D. 51
Burrows, Michael 16
Buschmann, Frank 21
Byrd, G. 51

Cantor, Scot 34, 35
Ceri, Stefano 39
Chamberlin, Don 143
Chandra, T. 38
Chang, Yue-Shan 38
Chanliau, Marc 34, 35
Chappell, David A. 38, 41
Christensen, E. 28

Clark, D. 24

Coyne, E.J. 25, 71, 191

Curbera, F. 28

Damiani, Maria Luisa 52

Dean, R. 51

Della-Libera, G. 34

Della-Libera, Giovanni 34

deTroyer, Olga 205

Deubler, Martin 52

Dixon, Brendan 34

Dusche, Mike 34

Eastlake, Donald 34, 91

Edwards, Nigel 34, 35

Eloff, J.H.P. 22, 24

Erdos, Marlena 34, 35

et al, Anthony Nadalin 34

et al, B. Dixon 34

et al, Don Box 34

Eugster, Patrick Th. 42

Farell, Stephen 34, 35

Feinstein, H.I. 25, 71, 191

Felber, Pascal A. 42

Fernandez-Buglioni, Eduardo 21

Fernández, Mary F. 143

Ferraiolo, D. 24, 69

Ferraiolo, David F. 24, 69

Flinn, Donald J. 35

Fossum, Timothy 38, 39

Fox, Barb 34, 91

Fraternali, Piero 39

Gasser, M. 71

Gavrilla, Serban I. 69

Geihs, K. 52

Georgakapoulos, G. 28–30, 32, 145, 191

Giuri, Luigi 70

Godik, Simon 34, 35

Goldstein, A. 71

Gollmann, Dieter 73

Grode, A. 52

Grünbauer, Johannes 52

Gudgin, Martin 34

Guerraoui, Rachid 42

Haas, Hugo 31

Hallam-Baker, Phillip 34, 35

Hamada, Takeo 24, 69

Hansche, Susan 49

Hansen, Stuart 38, 39

Hare, Chris 49

Harrison, Michael A. 22

Hartman, Bret 35

Hayton, R.J. 22

Hodges, Jeff 34, 35

Hollander, Dave 91

Hondo, Maryann 34

Hsiao, Tsun-Yu 38

Hur, Matt 34

Hutchison, B. 69

Hybertson, Duane 21

Iglio, Pietro 70

Jr., Stanley M. Sutton 38

Jürjens, Jan 52

Kalcklösch, R. 52

Kaler, Chris 34

Kaufman, C. 71

Kawamoto, Shirley 35

Kay, Michael 143

Kelley, Diana 145

Kermarrec, Anne-Marie 42

Klyne, G. 96

Knouse, Charles 34, 35

Koubarakis, M. 122

Kuhn, D. Richard 24, 69

Kuhn, R. 24, 69

LaMacchia, Brian 34, 91

Lambros, P. 69

Lampson, B. 71

LaPadula, Leonard J. 23

Layman, Andrew 91

Leune, Kees 2, 26, 38, 205

Li, Ninghui 70

Liang, Roger 38

Lo, Winston 38

Lockhart, Hal 34, 35

Luckham, D. 39

Lupu, Emil 70

M. Krause, H.F. Tipton 22

Maheshwari, Piyush 38

Maler, Eve 91

Maloney, M. 91

Maruyama, Hiroshi 34

McGuinness, Deborah L. 120

McLaren, Chris 34, 35

Mendelsohn, N. 91

Meredith, G. 28

Mikalsen, Thomas 38

Mitchell, John C. 70

Momini, Davide 52

Moody, K. 22

Moody, Ken 49

Moreh, Jahan 34, 35

Morgan, RL ”Bob” 34, 35

Morgan, Tony 39

Moritz, Rom 145

Moses, Tim 34, 35, 37, 191

Mühl, Gero 45

Mylopoulos, J. 122

Nadalin, Anthony 34

Nagaratnam, Nataraj 34

Nash, Andrew 34

Navathe, Shamkant B. 39

Needham, Roger 16

Newman, C. 96

Orchard, David 34, 35

Paoli, Jean 91

Paolucci, Massimo 48

Papazoglou, Mike 2, 26

Papazoglou, Mike P. 38, 41

Papazoglou, M.P. 28–30, 32, 145, 191

Passamani, Mario 39

Pato, Joe 34, 35

Perng, Nei-Chiung 38

Phippen, R. 69

Platt, Darren 34, 35

Prafullchandra, Hemma 34

Reagle, Joseph 34, 91

Reid, Irving 34, 35

Riccio, Alberto 39

Robie, Jonathan 143

Rouvellou, Isabelle 38

Ruzzo, Walter L. 22

Saltzer, J. 21, 71

Samarati, Pierangela 16

Sandhu, R. 24, 69

Sandhu, Ravi 16

Sandhu, R.S. 25, 71, 191

Sankar, Krishna 34, 35

Schmidt, M.-T. 69

Schroeder, M. 21, 71

Schumacher, Markus 21

Shewchuk, John 34

Siméon), Jérôme 143

Simon, Ed 34, 91

Sloman, Morris 70

Solo, David 34, 91

Sommerlad, Peter 21

Sperberg-McQueen, C.M. 91

Steele, Robert 51

Stoll, Clifford xiii

Strom, R. 38

Sturman, D. 38

Suppes, Patrick 119

Sycara, Katia 48

Tai, Stefan 38

Tang, Hua 38

Tao, Will 51

Thompson, H.S. 91

Ullman, Jeffrey D. 22, 122

van den Heuvel, Willem-Jan 2, 6, 26, 38, 41

van Harmelen, Frank 120

Vinoski, Steve 38

Weerawarana, Sanjiva 28

Welke, R. 6

Wilson, D. 24

Wimmel, Guido 52

Winsborough, William H. 70

Wolter, Roger 32

Yao, Walt 49

Yergeau, François 91

Youman, C.E. 25, 71, 191

Yuan, Shyan-Ming 38

Index

*-Property, 23

A-I-C triad, 49
Access control, 1, 21
Access control policies, 72
access control policy, centrally administered, 9
Access control policy, decentralized administration, 50
Access control policy, evaluating, 73
Access control rule, condition, 73
Access control rule, operation, 72
Access control rule, permission, 72
Access control rule, principal, 73
Access control rule, priority, 77
Access control rules, 72
Access control rules, evaluating, 78
Access control system, 64
Access control, message-context level, 50
Access matrix, 22
Active security, 49, 50
Apache Tomcat, 134
Asynchronous, 40
Asynchronous communication, 38
Audit trail, 16, 26, 50, 55
Auditability, 49
Auditability, and EFSOC, 87
Auditing, 16, 26
Authentication, 16, 50
Authentication, and EFSOC, 87
Authentication, by being, 17
Authentication, by knowledge, 17
Authentication, by possession, 17
Authorization, 16
Availability, 49
Availability, and EFSOC, 87

Bell-LaPadula model, 23
Biba model, 23
BLP, 23
Business rule, 80

Case study, 8
Causal relationship, 40
Causal relationship of events, 68
Causality header, EDL, 96
Certificate, 18
Certificate Authority, 18
Certificate path, 18
Certificate revocation, 18
Chronological relationship, 40
Chronological relationship of events, 68
Clark-Wilson model, 24
Communication, asynchronous, 38
Communication, event-driven, 38
Composition layer, 28
ConceptBase, 134
Confidentiality, 16, 17, 23, 49
Confidentiality, and EFSOC, 87
Constraints, 120
Constraints, and Datalog, 122
Constraints, and predicate logic, 121
Containment, 48
Containment, and EFSOC, 87
Credentials, shared, 16

DAC, 22
Data types, complex, 120
Datalog, 122
Datalog, and Constraints, 122
Datalog, and queries, 122
Deductive rules, and predicate logic, 121
Definition and Constraint Language, 91
Delegation, 46
Delegation of roles, characteristics, 47
Delegation, in EFSOC, 88
Digital signing, 17
Discretionary access control, 22
Distributed systems, loosely coupled, 30

EDL, 91
EDL, `accesscontrolpolicy`, 97
EDL, `accesscontrolrule`, 101
EDL, `activate`, 101
EDL, `assign`, 100
EDL, `condition`, 102
EDL, `deactivate`, 101

EDL, `eventbodytype`, 97
EDL, `eventbody`, 96
EDL, `eventheader`, 96
EDL, `event`, 95
EDL, `operation`, 102
EDL, `permission`, 102
EDL, `principal`, 102
EDL, `publish`, 97
EDL, `roleattributetype`, 95
EDL, `roleattributevalue`, 95
EDL, `role`, 95
EDL, `send`, 98
EDL, `set`, 101
EDL, `subject`, 93
EDL, `subscribe`, 99
EDL, `unassign`, 100
EDL, `unpublish`, 98
EDL, `unsubscribe`, 99
EDL, definition language, 91
EDL, event header `causality`, 96
EDL, event header `timestamp-sent`, 96
EDL, execution language, 91
EDL, in relation to WSDL, 92
EDL, vocabulary, 91
EFSOC query language, 107
EFSOC, and Auditability, 87
EFSOC, and Authentication, 87
EFSOC, and Availability, 87
EFSOC, and Confidentiality, 87
EFSOC, and Containment, 87
EFSOC, and Integrity, 87
EFSOC, and message-context level access control, 87
EFSOC, and Security Roadmap, 86
EFSOC, and separation of duty, 87
EFSOC, and service autonomy, 86
EFSOC, and web services, 85
EFSOC, Architecture, 78
EFSOC, Event routing, 80
EFSOC, Message-level access control, 80
EFSOC, querying, 107
EFSOC, Service mapper, 81
EFSOC, Transport-level access control, 78
EFSOC, Workflow monitor, 80
Encryption, 18
Encryption, asymmetric, 19
End-entity, 18
Enterprise Service Bus, 41
EQL, 107
EQL, basic queries, 113
EQL, second order queries, 115
ESB, 41
ESOA, 28
ESOA, Composition layer, 28
ESOA, Composition layer, Conformance, 30
ESOA, Composition layer, Coordination, 28
ESOA, Composition layer, Monitoring, 30
ESOA, Composition layer, QoS composition, 30
ESOA, Service management layer, 30
Event, 42, 64
Event cloud, 39
Event operation, publish, 65
Event operation, receive, 65
Event operation, send, 65
Event operation, subscribe, 65
Event operation, unpublish, 66
Event operation, unsubscribe, 66
Event operations, 65
Event processing, cardinality property, 41
Event processing, receiver cardinality property, 40
Event processing, relativity property, 40
Event processing, synchronicity property, 40
Event processing, temporal property, 41
Event, aspects of, 39
Event, causal relationship, 68
Event, characteristics of relationship between sender and receiver, 39
Event, chronological relationship, 68
Event, form aspect, 39
Event, relativity aspect, 39
Event, sequential relationship, 68
Event, significance aspect, 39
Event-driven communication, 38
Events, publishing, 66
Events, receiving, 67
Events, sending, 67
Extended SOA, 28

Fire-and-forget, 43
FLWOR, 109, 116
Formalism, required properties, 120
Formalizing, reasons for, 119

Grid computing, 27

Hibernate, 135
HIPAA, 57

Identity, 16, 18

identity, 16
Information security, 15
Inheritance, roles, 70
Integrity, 16, 17, 23, 49
Integrity *-property, 23
Integrity, and EFSOC, 87
Interaction patterns, fire-and-forget, 43
Interaction patterns, polling, 43
Interaction patterns, publish-subscribe, 42
Inverted semantics, 39

J2EE, 134
Java server pages, 134
JBoss, 134

Laboratory Experiment, 9
Laboratory experiment, 133
Late binding, 30
Least privilege, 21, 49
Literature research, 7
Loosely coupled, 30

MAC, 23
Mandatory access control, 23
Member, of role, 71
Message-context level access control in EFSOC, 87
Message-level protection, 17
Message-oriented middleware, 38
methodology, 6
MOM, 38
Multicasting, 39
Multilevel access control, 23
Multiplexing, 39
MySQL, 135

No read up, 23
No write down, 23
No write up, 23
Non-repudiation, 16, 49
Northside Hospital, 56
Notification, 42
Notification filtering, 43
Notification filtering, content-based, 45
Notification filtering, subject based, 44
Notification filtering, topic-based, 44

Operation, 64
Operations, 3

Parameterized roles, 70
Permission, 24
PHP, 134
PKI, 18
PKI, CA, 18
PKI, CA Hierarchy, 18
PKI, Certificate administration, 18
Policy, access control, 72
Polling, 43
Predicate logic, and constraints, 121
Predicate logic, and deductive rules, 121
Principal, 16, 64, 71
Principal, access control rule, 73
Privacy Rule, 57
Private key, 19
Problem definition, 7
Prolog, 122
Proof-of-concept, 135
Protection, message-level, 17
Prototype, 9, 133, 134
Prototype, architecture, 134
Prototype, audit trail director, 135
Prototype, event director, 135
Prototype, Java messaging system, 135
Prototype, persistent storage, 135
Prototype, security director, 135
Public key, 19
Public key infrastructure, 18
Publish-subscribe, 42
Publishing events, 66

Queries, 120
Queries, and Datalog, 122

RBAC, 24
Receiving events, 67
Research goal, 5
Research methodology, 6
Research objectives, 6
Results, assessment, 9
Role, 24, 64, 70
Role assignment, 71
Role attributes, 70
Role hierarchies, in EFSOC, 88
Role operation, activate role, 69
Role operation, assign attribute, 69
Role operation, assign role, 69
Role operation, deactivate role, 69
Role operation, revoke role, 69
Role operations, 69
Role session, 71
Role, inheritance, 70
Role, member of, 71
Role-based access control, 24, 70
Roles, parameterized, 70
Rule, business, 80
Rule, security, 80
Rules, access control, 72
Rules, deductive, 120

SAML, 35
SAML Authority, 36
SAML, Attribute assertion, 36
SAML, Authentication assertion, 36
SAML, Authorization assertion, 36
Sarbanes-Oxley Act, 26
Secure message layer, 51
Secure transport layer, 51
Security Assertion Markup Language, 35
Security Roadmap, 32
Security roadmap, and EFSOC, 86
Security rule, 80
Semantics, inverted, 39
Sending events, 67
Separation of Duty, 22
Separation of duty, 49
Separation of Duty, and EFSOC, 87
Separation of duty, dynamic, 22, 49
Separation of duty, static, 22, 49
Sequential relationship of events, 68
Service, 64
Service aggregation, 31
Service autonomy, 48
Service autonomy, and EFSOC, 86
Service broker, 55
Service composition, 30
Service management layer, 30
Service-oriented Architecture, 32
Service-oriented architecture, 28
Service-oriented computing, 28
Services, 3
Shared credentials, 16
Signing, digitally, 17
Simple integrity property, 23
Simple security property, 23
SOA, 28, 32
SOA, Design objects for a secure, 48
SOC, 28
Software bus, 42
Solution design, 7
SOX, 26
Subject, 16, 24, 64
Synchronous, 40

Telos, 122
Telos, attribute, 122
Telos, individual, 122
Telos, meta class, 123
Telos, proposition, 122
Telos, simple class, 123
Telos, token, 123
Trust, 45
Trusted third parties, 18
Trusted third party, 19

Validation, 8

web of trust, 19
Web service, 31
Web services, and EFSOC, 85
WS-Authorization, 35
WS-Encryption, 51
WS-Federation, 34
WS-Policy, 34
WS-Privacy, 34
WS-SecureConversation, 34
WS-Security, 32
WS-Signatures, 51
WS-Trust, 34
WSDL, 28
WSDL, in relation to EDL, 92

XACML, 36
XACML, PAP, 36
XACML, PDP, 36
XACML, PEP, 36
XACML, PIP, 36
XACML, Policy Administration Point, 36
XACML, Policy Decision Point, 36
XACML, Policy Enforcement Point, 36
XACML, Policy Information Point, 36
XML, 91
XML Web services, 32
XML, querying, 107
XPath, 107
XQuery, 109
XQuery, FLWOR, 109, 116
XSL, 111

Curriculum Vitae

Cornelis Jan (Kees) Leune was born in Breda, on 29 August 1973. After finishing high school (Atheneum) at the Orduynen College in ’s-Hertogenbosch, he enrolled as a full-time student in the Information Systems and Technology program at Tilburg University. During his studies, Kees was a board member of the student association SBIT, and teaching assistant for four years for a variety of courses.

His graduation project was titled ‘Wisdom on the Web’, and attempted to find a formal design method for designing Web sites and Web-enabled application. The results of his research work were published at the seventh International World Wide Web conference in Brisbane, Australia (deTroyer and Leune, 1998).

After graduating, Kees took a position as a full-time researcher at CentER Applied Research. At that time, CentER Applied Research was an independent research institute. His work focused mainly on data modeling, information discovery and on electronic commerce.

In 2003, Kees switched back from CentER Applied Research to his Alma Mater, where the department of Information Systems and Management facilitated him to pursue his PhD research. This quickly led to several academic publication, such as ‘Exploring a Multi-Faceted Framework for SOC’ (RIDE ’04) and ’EFSOC: A Layered Framework for Developing Secure Interactions Between Web Services’ (Distributed and Parallel Databases, September 2005, Vol. 18, No. 2). In addition to his publications, Kees also contributed to the scientific world by peer reviewing articles and by being the local chair of the 2005 International Conference on Service-Oriented Computing.

Kees was also involved in teaching activities. His principal activities centered around an introductory course on Database Systems, but he also gave many guest lectures for courses such as Telematics, Software Engineering, Computer Infrastructures, etc.

Finally, Kees was a member of Tilburg University’s Computer Security Incident Response Team, member of the computer and network security task force of the University, and a member of the Dutch o-IRT-o.

As of October 2006, Kees works as an Information Security specialist at Northwave.