University of the Federal Armed Forces, Munich
Computer Science Department
Institute of Information Systems

Doctoral Dissertation

Service Oriented Security Architecture
applied to Spatial Data Infrastructures

PLANISPHÆRIVM Sive ORBIVM MVNDI PTOLEMA NO DI
PTOLEMAICVM, Machina EX HYPOTHESI ICA IN PLA SPOSITA.
SEPTEM
PLANETARVM
ORBES

by

Cristian Aurel OPINCARU

Munich, 2008

Map on the Cover Page

Planisphaerium Ptolemaicum siue machina orbium mundi ex hypothesi Ptolemaica in plano disposita, Author: Andreas Cellarius, published in *Harmonia Macrocosmica* (Amsterdam, 1661), Dimensions (original): 46 x 64 cm. Courtesy of the Norman B. Leventhal Map Center at the Boston Public Library.
The map shows the view of Claudius Ptolemy (2nd century A.D.) on the universe. At the center of this diagram there is a small map of the Earth's northern hemisphere. Revolving around the Earth in separate orbits are the Moon, Mercury, Venus, the Sun, Mars, Jupiter, and Saturn. The outer circle was reserved for the stars, represented as the constellations of the zodiac [44].
The map is intended as a metaphor. It illustrates a complex system, which is composed from several independent entities that take part in a choreography (the planets orbit around the Earth).

UNIVERSITÄT DER BUNDESWEHR MÜNCHEN
FAKULTÄT FÜR INFORMATIK

Service Oriented Security Architecture
applied to Spatial Data Infrastructures

Dipl. Ing. Cristian Aurel OPINCARU

Von der Fakultät für Informatik der Universität der Bundeswehr München zur Erlangung des akademischen Grades eines

Doktors der Naturwissenschaften (Dr. rer. nat.)

genehmigten Dissertation.

Datum des Kolloquiums: 11. Januar 2008

Abstract

Web services have been a major technology trend in the IT industry for almost a decade now. They have been promoted as a means of reducing costs, increasing reuse, simplifying integration and creating more agile infrastructures. For these reasons the industry has witnessed a pervasive shift in the use of technology, as Web services and Service Oriented Architecture (SOA) replace other methods and technologies used in design, development, deployment and integration, and management services. In this context, security represents a special concern, because without a solid foundation for security, the Web services stack would be next to useless.

In this dissertation the topic of security in the context of Web services is addressed and an architecture for security systems is proposed. The main idea behind this architecture is that security functions such as authentication, authorization, audit, etc. are implemented externally to the service provider, and not embedded in this one or implemented at the message middleware layer. Furthermore, the different security functions are realized as independent services and designed according to the principle of separation of concerns. This leads to a collection of *security services* forming a security infrastructure: these services can be used to protect several service providers within a network. Enterprise Application Integration (EAI) techniques are used for combining the security services together and binding them to service providers through the definition of choreographies. Because of its service-oriented approach to security, the architecture is named the **S**ervice **O**riented **S**ecurity **A**rchitecture or ***SOSA*** .

In the dissertation, the existing work in this direction is first evaluated. Then, the main components, connectors and information elements of this architecture are specified by means of the ISO Reference Model for Open Distributed Processing. Two prototype implementations are showed, and practical experiences using these implementations to protect Web services serving geographical content are described. The architecture is analyzed both from a design point of view as well as from the point of view of the impact that it has on the performance of service providers. This work closes with a conclusion and outlook.

Kurzfassung

Die Technologie der Web Services hat sich in den letzten Jahren als ein wichtiger Trend für die IT Industrie entwickelt. Web Services werden besonders deswegen als wesentlich erachtet, weil sie das Potential zur Reduzierung der Kosten, zur Erhöhung der Wiederverwendung haben und zur Verweinfachung der Integration beizutragen und somit die Schaffung einer agileren verteilten Infrastruktur ermöglichen. Aus diesem Grund findet ein Technologiewechsel zu Web Services und Service Oriented Architecture (SOA) statt, der allerdings ein Umdenken bzgl. Design, Entwicklung und Verteilung der Dienste sowie deren Integration und Management erfordert. In diesem Zusammenhang richtet sich besonderes Augemerk auf verschiedene Sicherheitsaspekte, die sich u.a. durch die zugrunde liegende Service Oriented Architecture ergeben und ohne deren Umsetzung eine Nutzung von Web Services undenkbar wäre.
Diese Dissertation stellt eine Sicherheitsarchitektur im Kontext von Web Services vor mit der die Umsetzung von typischen Sicherheitsaspekten flexibel möglich ist. Die Kernidee der Architektur ist es, dass Sicherheitsfunktionalitäten wie Authentifizierung, Autorisierung und Überwachung quasi unabhängig von der eigentlichen Anwendung implementiert und betrieben werden. Insbesondere sollen die Sicherheitsfunktionalitäten als autark verfügbare Web Services nach dem Prinzip der Gewaltenteilung entwickelt und betrieben werden. Dies führt zu einer Infrastruktur von orchestrierten Sicherheitsdiensten, die es auf der Grundlage von Enterprise Application Integration (EAI) erlauben, verschiedene - evtl. voneinander unabhängige - Anwendungen zu schützen. Wegen dieses Vorgehens heißt die hier vorgestellte Architektur **S**ervice **O**riented **S**ecurity **A**rchitecture, oder kurz ***SOSA*** .
Zuerst werden in dieser Dissertation die bereits existierenden Arbeiten in diesem Forschungsgebiet untersucht und ausgewertet. Anschließend werden erforderliche Komponenten, Schnittstellen und Informationselemente für SOSA gemäß dem ISO Reference Model for Open Distributed Processing vorgestellt. Es werden zwei prototypische Implementierungen als Proof-Of-Concept für den Schutz von geo-spezifischen Web Services vorgestellt und hierbei gewonnene Erfahrungen diskutiert. Abschließend wird die vorgestellte Architektur vom Gesichtspunkt des Designs bewertet. Ebenso wird analysiert, welche Auswirkungen die Performance auf die Nutzbarkeit der Architektur zum Schutz von verteilten Anwendungen hat. Diese Arbeit schließt mit einer Zusammenfassung und einem Ausblick, in dem auch auf die Auswirkungen für die aktuelle sowie weitergehende Forschung eingegangen wird.

Foreword

Context

This dissertation has been carried out while I worked as a research associate at the Institute of Information Systems within the Computer Science Department of the University of the German Armed Forces in Munich. It is based on the knowledge and experience that I gained between 2003 and 2007 while being involved in various research and industry projects. The projects most relevant to this dissertation are the following:

- The *OGC Web Services, Phase 3*[1] test bed (2005). I was involved in the GeoDRM thread, where click-through licensing extensions for the WMS and WFS service interfaces were defined.
- *Disclaimer-Enablement for the WMS service* (2006) commissioned for the Bavarian Mapping Agency. In this research project the results from OWS-3 were further developed.
- The *OGC Web Services, Phase 4*[2] test bed (2006). I was involved in the GeoDRM thread, where different components and specifications for the licensing of content served via WMS and WFS services were drafted.

Further experience and knowledge related to security topics I gained while working together with the other members of the Open Geospatial Consortium. I was involved in the activities of the *GeoDRM WG* and the *Security WG* and participated in the creation of several documents [149, 105, 79, 65, 68].
The involvement in all these geospatial-related activities offered me a good opportunity to experiment with ***SOSA*** in real-life scenarios. The practical work of this dissertation is in the field of geospatial Web services.

Acknowledgments

First of all I would like to thank Prof. Dr. Gunnar Teege for supervising my work, for giving me the freedom in choosing my topic, for supporting me throughout the past four years, for always being able to find funding for my research (I know this hasn't been always easy) and for making me learn the German language. Further thanks go to Prof. Dr. Johann Schlichter for proof-reading this thesis. Next, I would like to thank all my colleagues at the university for good discussions, in particular the people I shared the office with - Andreas Matheus (special thanks) and Frank Eyermann.
Next I would like to thank the current members of the GeoDRM working group within the OGC. I met extraordinary people here and during the almost three years that I've been involved in their

[1]http://www.opengeospatial.org/projects/initiatives/ows-3
[2]http://www.opengeospatial.org/projects/initiatives/ows-4

activities I gained a lot of knowledge and had many fruitful discussions. It was fun and I hope I will be able to remain in contact with this group in the future.
Further thanks go to the Round Table GIS for supporting my research and allowing me to participate in research projects that provided me a good insight in the field of GIS and geospatial information. I thank Prof. Dr. Matthäus Schilcher and Dr. Andreas Donaubauer. Further experience in the field of geospatial information I gained during my part-time work at Intergraph GmbH; for this I want to thank my supervisor there, Mr. Lothar Urban.
I also thank the students that chose me as their supervisor for their diploma thesis or their student assignments. Many parts of the ***SOS!e*** framework have been developed in this way. Special thanks are addressed to: Andreas Hänel, Ole Pophal, Przemyslaw Stefaniak, Ofelia Tindea, Gabriela Gheorghe and Sebastian Zimmer. Further thanks go to Lidia Gietler for her help with the OGC performance testing.
Finally, I thank all my friends that supported me: those in Munich, those in Bucharest and those scattered across the globe. The most special thanks and the deepest appreciation are addressed to my family who has missed me for the past four years while I've been away from Bucharest.

Munich, August 2007

Contents

Chapter 1

Introduction

This chapter frames the scope of this dissertation and provides motivation for its scope. In the end, an outline of this work is presented.

1.1 Motivation

1.1.1 SOA with Web Services

Web services have been a major technology trend in the IT industry for almost a decade now. Whether they are implemented based on the SOAP protocol stack or follow the REST paradigm, Web services provide the ability to build distributed systems that are interoperable, although they run on different hardware architectures, have been developed using different programming languages and they rely on different Application Programmable Interfaces (APIs). Furthermore, as remarked in [93], Web services promise a future where applications can be created by combining multiple services in a single workflow. This results in more flexible applications because services can be added, removed or replaced from a workflow, either statically (in the development phase) or dynamically (at run-time). Due to the standardization efforts of numerous organizations, most notably W3C and OASIS, the vision of Service Oriented Architectures (SOAs) has become today a reality.

Web services have been promoted as a means of reducing costs, increasing reuse, simplifying integration and creating more agile infrastructures. A Gartner investigation in the market of Web services from 2005 [55] has showed that the IT professional services market involving well-defined Web services is forecast to reach $261 billion by 2008 (up from an estimate of $3.8 billion in 2003). According to the same source, this represents a pervasive shift in the use of technology rather than an incremental market opportunity: Web services and SOA replace other methods and technologies used in design, development, deployment and integration, and management services.

This shift in technology can be seen not only in the mainstream IT industry, but also in the related domains. In the field of geospatial[1] information, the Open Geospatial Consortium[2] (the most important standardization organization in this domain) published its first service specification in year 2000 - the Web Map Service (WMS). Over the past seven years, several other service specifi-

[1] According to [59] *geospatial* is more precise than *geographic* in many contexts, because geospatial information is often used in ways that do not involve a graphic representation (such as maps) of the information.

[2] http://www.opengeospatial.org

cations and data encoding formats have been approved by the OGC membership and adopted by national agencies as implementation standards for spatial information distribution. A survey by Skylab Mobilesystems Ltd. from 2006 [106] shows that 994 WMS services, hosting 339,254 layers are publicly available.

1.1.2 Security in the Context of SOA

Quite naturally, everybody deploying applications over the Internet expect their Web services to provide a solid foundation for security. As the author of "Web Service Security" [71] points out *it should be obvious that the prospect of different companies communicating together, while powerful, is fraught with security concerns. In fact, without a conceiving security model, the Web services framework we've outlined would be next to useless.*
Security is a key requirement in the success of any SOA project and although standards like SOAP and WSDL have been in place for almost a decade, security is still a present topic. A survey of the Gartner group from 2005 where 1,300 CIOs where interviewed [97] showed that in the top 10 technology priorities indicated by the CIOs, security occupied the first place.
In an attempt to make Web services a solid ground, OASIS has standardized a number of extensions to SOAP messaging which address different security issues related to Web services. These extensions are WS-Security [35], WS-Trust [40], WS-Federation [33], WS-SecureConversation [28] and WS-Policy[3]. In addition to the SOAP extensions, other security specifications can be used in combination with Web services - XACML [18] , SAML [22] or the newer Digital Signatures Services [38] are some examples. These specifications are very useful and have been embraced by the industry. However, they only deal with distinct aspects of security and no comprehensive architectural guideline is provided by the standards organizations.
Similar to the mainstream IT industry, security is a present topic also in the geospatial information field. Two working groups within the OGC address this issue (the GeoDRM WG and the Security WG) and there have been numerous approaches both from the industry [105, 68, 149, 37] as well as from the academia [107, 118, 152] to enhancing OGC service interfaces with security features.

1.1.3 External Security. Service Oriented Security Architecture

As opposed to the client-server model where a message is sent from the client directly to the server, when using SOAP, a message travels from the requester to the ultimate receiver through zero ore more intermediaries. These intermediaries can modify the message in whole or in part and offer a very elegant way to implement additional functionality and to create value-added services. In the context of security, we can imagine intermediary services implementing different security functions (i.e. authentication, authorization, audit, etc). In this case the security system is no longer implemented in the Web service itself, rather it is external to this one. As remarked in [92], one can speak about *security unaware* services: the assumption is that any security data or security process is provided outside the application.
This idea can be taken one step further, in that the security system can be architected as a collection of Web services, each one implementing a different security function. These security services will form a kind of infrastructure that can be reused throughout the network: when a new application is deployed a security solution can be realized by putting together several security services and configuring them to work together with the newly deployed application. Enterprise Application

[3]WS-Policy has been submitted for standardization at W3C

Integration (EAI) techniques are used for combining the security services together with the service providers through the definition of orchestrations or choreographies. The approach has several obvious advantages: it is simple, has a high degree of reusability and scalability, and easily allows for extensions.
Further advantages of this solution constitute the fact that security services are independent from one another and can be hosted in different physical locations. This opens the way for the outsourcing of security management: certain security tasks can be delegated to trusted third-parties. This is especially appealing in the context of geospatial Web services, where data providers often do not have the necessary know-how available in the company to implement complex A4C[4] solutions required by online business models.

1.2 Scope

This dissertation addresses the issue of security in the context of Web services and proposes an architecture for security systems where security is externalized and realized as a set of modular security services: the **S**ervice **O**riented **S**ecurity **A**rchitecture or ***SOSA*** .
The following topics fall within the scope of this dissertation:

- An analysis of the possible approaches to implementing security systems;
- An investigation into existing approaches where external security is used;
- A detailed description of the proposed architecture from different perspectives using the ISO RM-ODP specification, including a taxonomy of possible security services;
- The description of prototype implementations of the proposed architecture;
- Practical experiences with the prototype implementations in the field of security for geospatial Web services;
- An evaluation of the proposed architecture, both from the design point of view as well as from the point of view of the performance impact on the service provider.

1.3 Outline

The outline of the dissertation is graphically illustrated in figure 1.1. A brief description of the contents of each chapter is presented below:

Chapter 2 Introduces the basic concepts required to understand this work.

Chapter 3 Defines metrics for evaluating software architectures and by this means, an analysis of different approaches to implementing security systems is performed. Existing approaches to externalize security are showed and the problem addressed in this dissertation is framed.

Chapter 4 Constitutes a preamble to chapter 5. Here, the main ideas behind the proposed architecture are presented, and some clarifications regarding design decisions are made.

Chapter 5 Contains a detailed description of the proposed architecture using the ISO RM-ODP specification. This covers the scope and the functions of the security system, the most important information elements, the distribution of the components together with the required infrastructure as well as implementation aspects.

[4]Authentication, Authorization, Audit, Accounting and Charging

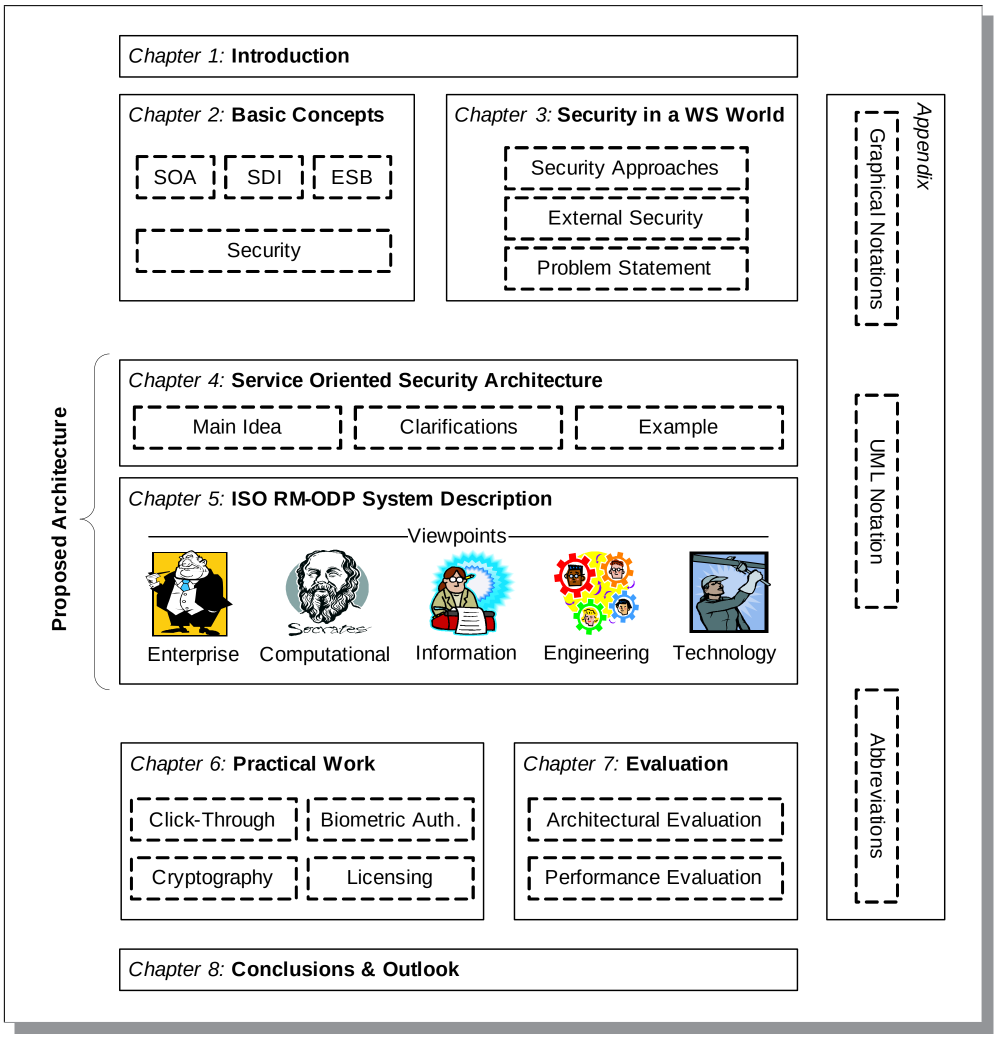

Figure 1.1: Outline of the dissertation

Chapter 6 Shows how the proposed architecture has been applied in practice in several projects where the topic of securing geospatial Web services was addressed.

Chapter 7 Evaluates the proposed architecture using the metrics defined in chapter 3. A detailed performance evaluation is presented and the impact on the performance of service providers are discussed.

Chapter 8 Concludes with the results of this work, enumerates the contributions this dissertation brings to information and computer science as well as the world of geospatial information, and provides an outlook for future research.

Chapter 2

Basic Concepts

This chapter introduces the relevant concepts and technologies necessary to understand this dissertation. Four different topics are addressed: Service Oriented Architecture (SOA), Spatial Data Infrastructures (SDI), Enterprise Services Bus (ESB) and security in the context of Web services. Because they are presented in a brief manner, readers are advised to follow the provided references.

2.1 Service Oriented Architecture

The Reference Model for Service Oriented Architecture 1.0 [140] defines Service Oriented Architecture as a paradigm for organizing and utilizing distributed capabilities that may be under the control of different ownership domains. Because it aims at being a reference model, and as such independent of any specific standard, technology, implementation or other concrete details, the document describes the concepts relative to SOA in terms of matching one's needs with someone else's capabilities. The document notes that the following are the key concepts in the SOA paradigm:

Visibility Refers to the capacity of those with needs and those with capabilities to be able to see each other. It refers to the process of matching the needs with the capabilities.

Interaction Refers to the activity of using a capability.

Real World Effect Refers to the outcome of an interaction. As such, the purpose of using a capability is to realize one or more real world effects. This can be the return of information or the change of state of the entities involved in the interaction.

Interestingly enough, the reference model also notes that the context of SOA should not be restricted to Web services. There are various other implementation strategies by which services can be made visible, support interaction and generate effects. In fact, the document also gives a SOA example taken from the non-software world: an electricity utility. However, the most common way of implementing a SOA are Web service-based architectures, and this is the only relevant implementation of SOA in this dissertation.
A less abstract definition can be found in [93] where SOA is described as an environment where software applications expose functionality via a service provider messaging interface. As such, other software agents can act as service consumers by using the functionality exposed by the service providers.

Similar to the client-server model, in a SOA an entity can be both service consumer and service provider, although in one interaction it acts as either consumer or provider. An example for this is the classical airline ticket booking service: an application uses the booking system of several airlines in order to find the cheapest ticket for a given trip - in this case the application acts as a service consumer. At the same time the application provides a service to its customers (i.e. finding and booking the cheapest airline ticket).
In the rest of this section I will first present the most important concepts that are related to service oriented architecture and then briefly introduce the technologies and standards that are associated with the SOA paradigm.

2.1.1 SOA with Web Services

What is a Service?

According to [140] a service is a mechanism to enable access to one or more capabilities, where the access is provided using a prescribed interface and is exercised consistent with constraints and policies as specified by the service description.
A service is accessed by means of a service interface and is usually opaque to the service consumer (its internal implementation is not revealed). The consequence of executing a service is the realization of a real world effect.

What is a Web service?

According to [49] a Web service is 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.

Remarks

- This definition is very rigid because it also includes the communication protocols and the service description language. The mentioned protocols (XML, SOAP, WSDL) are used in the implementation of the security system (see technology viewpoint, section 5.6). However, the security system could also be implemented using other messaging systems, and as such these standards are not a requirement for the ***SOS*** concept.
- A definition which does not reference any specific protocols or standards is the one found in the OGC Glossary of Terms ([59]): Web services are self-contained, self-describing, modular applications that can be published, located, and invoked across the Web. Web services perform functions that can be anything from simple requests to complicated business processes. Once a Web service is deployed, other applications (and other Web services) can discover and invoke the deployed service.

2.1.2 Typical Interactions in a SOA

Figure 2.1[1] shows the typical steps to engaging a Web service in a SOA:

[1]This figure is taken from [49]. For copyright information see `http://www.w3.org/Consortium/Legal/2002/copyright-documents-20021231`.

1. The requester and the provider become known to each other
2. The two entities agree on the description of the service and the semantics of the messages
3. The semantics and service description are realized by the software agents
4. The interaction takes place, thus the requester and the provider exchange messages

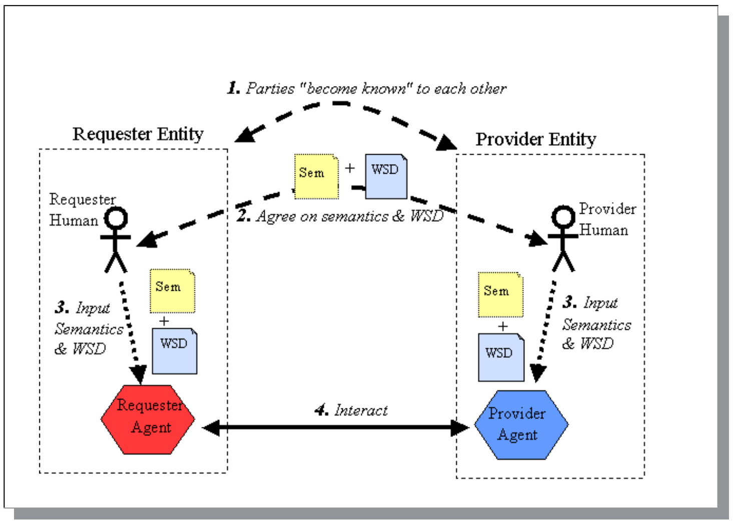

Figure 2.1: Interactions in a SOA

We therefore find that the most important components in a SOA are the following:

Discovery Is the process by which the service requester and service provider become known to each other (it is actually enough if the requester knows the service provider). This can be realized in various ways, from out-of-band non-automated processes - for example a user receives the service description per email - to fully automated machine-to-machine interactions such as UDDI.

Description Once the service provider is known, the requester and the provider need to agree on the way the messages are exchanged (protocol, ports, etc.) as well as the contents of the messages. One way to realize the service description is the Web Services Description Language (see 2.1.7 for more details). Note that the two entities need to agree on both the syntax as well as the semantics of the messages exchanged - WSDL only addresses the first one.

Messaging The most important component is the message exchange. This can be realized for example by using SOAP together with a transport protocol such as HTTP (more about that in 2.1.8). Note that there are different message interactions possible: most commonly, the two way request-response interaction is used; however other exist: one-way, notify, etc.

2.1.3 Concepts and Definitions

Agent A **software agent** is a software program acting on behalf of a person or organization [49].

Choreography A **choreography** defines the sequence and conditions under which multiple cooperating independent agents exchange messages in order to perform a task to achieve a goal state [49].

Message A **message** is the basic unit of data sent from one Web service agent to another [49].

Message Exchange Pattern A **message exchange pattern** is a template, devoid of application semantics, that describes a generic pattern for the exchange of messages between agents. It describes the relationships (e.g., temporal, casual, sequential, etc.) of multiple messages exchanged in conformance with the pattern, as well as the normal and abnormal termination of any message exchange conforming to the pattern [49]. MEPs are usually of small scale (whereas choreographies are of larger scale.

Orchestration An **orchestration** defines the sequence and conditions in which one Web service invokes other Web services in order to realize some useful function [49]. It can be differentiated from a choreography which describes the interactions between several agents, without having one orchestrator.

Policy A **policy** is a constraint on the behavior of agents or people or organizations [49]. The constraints can take various forms from a specific protocol to be used when accessing the service to providing identity information based on which access control is done. Most Web services (for obvious reasons) usually have such constraints that need to be enforced. Enforcing and realizing such policies is one of the main goals of this dissertation.

Service Intermediary A **service intermediary** is a Web service whose role is to transform messages in a value-added way [...] specifically, we say that a service intermediary is a service whose outgoing messages are equivalent to its incoming messages in some application-defined way [49]. Service intermediaries are implemented in SOAP (see 2.1.8) and are an important concept in both ESB and the realization of ***SOSA*** .

2.1.4 Standards and Technologies for SOA

As you can see in figure 2.2, SOA requires many interrelated technologies and standards.

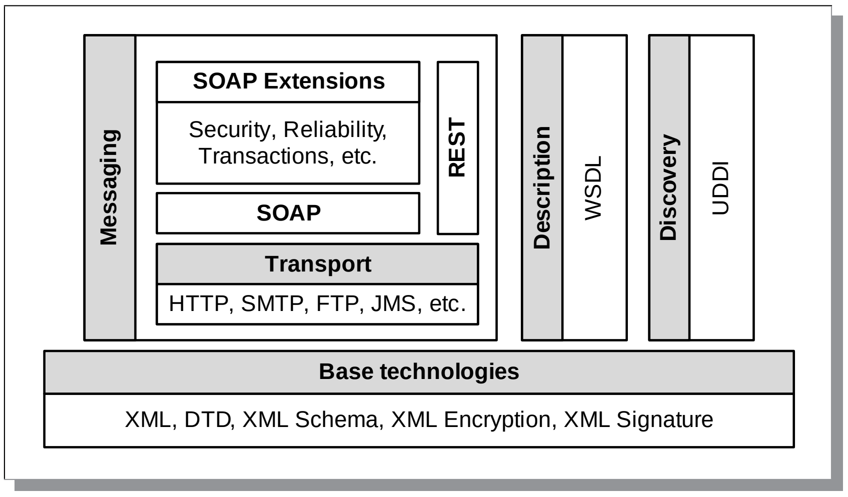
Messaging
SOAP Extensions
Security, Reliability, Transactions, etc.
SOAP
REST
Transport
HTTP, SMTP, FTP, JMS, etc.
Description
WSDL
Discovery
UDDI
Base technologies
XML, DTD, XML Schema, XML Encryption, XML Signature

Figure 2.2: Technologies related to SOA

The foundation of Service Oriented Architecture are the technologies placed on the bottom of the drawing. These are XML, DTD and XML Schema. They are standardized by W3C and are briefly

introduced in sections 2.1.5 and 2.1.6.
Building on the base technologies you can see three columns representing the three important components of a SOA: messaging, description and discovery. The latter two are introduced in sections 2.1.7 and 2.1.9 respectively.
The messaging column cumulates both the content of the message as well as the delivery. As far as the content of the message is concerned there are two major architectural approaches: REST and SOAP; both of them are presented in section 2.1.8. For the delivery of messages, Web services use already existing protocols such as HTTP or SMTP. The reason behind having so many choices for transporting the message is that each of these protocols has its advantages and disadvantages which make them suitable for particular scenarios. For example, HTTP is widely deployed and passes through most firewalls, SMTP is asynchronous, IIOP can be used for integration with CORBA, etc.
There is a maze of technologies and standards behind Web services, and because of the extensibility of standards such as SOAP or WS-Security, new specifications are emerging every year. For this reason, standards[2] and standardization organizations are crucial. The most important organizations that are involved in the standardization process for SOA related technologies are:

W3C The World Wide Web Consortium[3] has standardized most core technologies such as XML, XML Schema, XML Encryption, XML Signature as well as some messaging standards such as SOAP and some SOAP extensions (addressing, choreography, etc.). WSDL (version 2.0) is also in the process of standardization at W3C.

OASIS The Organization for the Advancement of Structured Information Standards[4] has been involved in several XML specifications: UDDI, ebXML, several security standards (WS-Security, XACML, SAML) and several SOAP extensions (WS-Reliability, WS-ResourceFramework, etc.)

WS-I The Web Services Interoperability Organization[5] has profiled several standards (SOAP, WSDL) resulting in so called "basic interoperability profiles" which aim at achieving full interoperability across platforms, operating systems and programming languages.

Interoperability

ISO defines interoperability as the capability to communicate, execute programs, or transfer data among various functional units in a manner that requires the user to have little or no knowledge of the unique characteristics of those units [19].
The success of Web services and what made SOA possible is that interoperability is not based on a standard programming interface but rather on a standard format for the messaging protocol. This results in a loosely coupled design which allows systems to exchange messages regardless of hardware platform, operating system or programming language.
Because of the loosely coupled design, in SOA components have few dependencies. This makes it easy for parts of the system to be designed and implemented separately or to be integrated with other already existing components. This design pattern is exploited very much by the security system proposed in this dissertation.
At the same time interoperability also comes as a challenge when designing a security system to

[2] A funny quote about standards is: *the good thing about standards is that there are plenty to choose from.* This is particularly relevant when dealing with SOA and the WS-* specifications.
[3] http://www.w3c.org
[4] http://www.oasis-open.org
[5] http://www.ws-i.org

be deployed in a SOA, because the security layer should itself be interoperable and not hinder the otherwise existing interoperability.

What Follows

The intention of the following few sections is to familiarize the reader with the most important technologies and standards related to SOA. Each technology will be *briefly* introduced by answering the following questions:

- What is it? (where appropriate, examples shall be provided)
- What is the relation with the other technologies?
- What are the features and weaknesses?
- What is the relation with this dissertation?

The reader is invited to follow the references for those technologies that are unfamiliar to him.

2.1.5 XML

The **eXtensible Markup Language** (XML) is a simple, very flexible, general-purpose markup language supporting a wide variety of applications. It is based on SGML (ISO 8879) and is currently a W3C recommendation. Much like HTML, XML was designed for describing data. However, as opposed to HTML where the focus is on the representation of data, XML was designed to describe the structure of data.
Some of the strengths of XML are: it is both human readable and machine readable, it has a hierarchical structure which is suitable for most documents, it has a self-documenting format and it is expressed as plain text which makes it platform-independent.
XML uses tags for structuring the data. The syntax in XML is defined by either Document Type Definitions (DTD) or by XMLSchema, the latter one being much more powerful (see section 2.1.6 for details).
Probably one of the most important features if we think about XML in the context of Web services is its extensibility capability and the fact that it allows several XML languages to be combined in one document by means of namespaces. These features are use extensively by standards such as SOAP and the WS-* specifications.
The implementation of the concepts described in this dissertation are based on SOAP and the WS-* specifications. Because of that, messages and other information objects are encoded in XML. This allows them to take advantage of all XML features (flexibility, extensibility, platform neutrality, ability to combine vocabularies), while suffering at the same time from the XML weaknesses (verbosity and redundancy which can lead to efficiency problems).

2.1.6 DTD and XML Schema

Both **Document Type Definition** (DTD) and **XML Schema** are schema languages for XML which are used to specify one type of XML document. This is done in terms of constrains on the structure and the contents of that document type. Through the use of document schemes, independent groups of people can agree to use a common format for documents and then exchange information. Because the document structure is defined, the exchanged information can be verified. This is especially important when exchanging messages in a SOA, because the partners do not know

each other in advance.
XML itself specifies basic syntax constrains. An XML document that conforms to the XML syntax rules is called *well-formed*. If a schema is defined for that document type and the document conforms to this particular schema, then the document is called *valid*.

Document Type Definitions

Document Type Definitions are defined in the XML specification itself [32] and come from the ancestor of XML - SGML. They provide grammar for a class of documents. The grammar is defined as a list of declarations where each declaration can be either one of: an element type declaration, an attribute list declaration, an entity (type) declaration or an annotation declaration. However, although they are included in the XML specification, DTDs have attracted a lot of criticism due to their limitations: no support for namespaces, lack of expressiveness as well as the fact that DTD are not themselves XML documents.

XML Schema

As opposed to DTDs, **XML Schemas** are XML documents themselves. Because of this, they enjoy all design advantages of XML (for example they can be validated in the exact same way an XML document is validated).
XML Schema is a W3C recommendation [17] and is the schema language of choice for many of the structured information standards that involve SOA. The main reason for that is the fact that XML Schema allows the specification of data types (such as integer, string, etc.) together with finer-grained constraints on element contents (as fine-grained as for example specifying the minimum as well as the maximum value of an integer field). Furthermore, XML schema also supports namespaces which allow several XML vocabularies to be combined into one document, as well as an object-oriented inheritance model for element definition through its extensibility mechanism.
Listing 2.1 exemplifies the basic features of XML Schema by showing the definition of a SOAP message which is structured in header and body.

```
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
  xmlns:tns="http://schemas.xmlsoap.org/soap/envelope/"
  targetNamespace="http://schemas.xmlsoap.org/soap/envelope/" >

  <!-- Envelope, header and body -->
  <xs:element name="Envelope" type="tns:Envelope" />
  <xs:complexType name="Envelope" >
    <xs:sequence>
      <xs:element ref="tns:Header" minOccurs="0" />
      <xs:element ref="tns:Body" minOccurs="1" />

      <xs:any namespace="##other" minOccurs="0"
        maxOccurs="unbounded" processContents="lax" />
    </xs:sequence>
    <xs:anyAttribute namespace="##other" processContents="lax" />
  </xs:complexType>
```

Listing 2.1: Part of the XML schema definition for SOAP messages

Because the concepts described in this dissertation are implemented in XML, the information elements are described using XML Schema. In addition, schema definitions help preventing security attacks aimed toward the application and its XML parser where the an attacker purposely sends messages with incorrect structure in order to speculate bugs. For this, an XML Validation service

is proposed.
Note that other alternatives to XML schemes exist (RELAX NG - ISO standard, Document Structure Descriptions). However because most of the standards and technologies used in this dissertation are using DTDs and XML Schema for describing information elements they are not presented here.

2.1.7 Service Description: WSDL

Web Services Description Language (WSDL) is an XML language used to describe Web services end points. The current version of this standard is 2.0 [39], which has recently been approved. However, in this dissertation the previous version is used, WSDL 1.1 [158]. This has been the de facto service description language for the past six years, and it has support implemented in most products. It is also the version endorsed by the WS-I Basic Profile 1.1 ([31] - the current one at the time of writing).
A WSDL document contains the information about how to communicate with a Web service. Specifically, it contains the following information:

- A description of the content of the messages (using XML Schema);
- The operations that the Web service supports;
- The supported transport protocol bindings (how the Web service can be accessed).

To illustrate this, I present in listing 2.2 the WSDL definition for a Web Map Service as described in [66]. Two operations are defined: GetCapabilities and GetMap - the first one returns an XML document, while the latter one returns an image. For both operations the messages are encoded as literal (no restrictions are set on the content of the request messages).

```
<?xml version="1.0" encoding="UTF-8"?>
<wsdl:definitions
  <!-- Namespace definitions are omitted -->
 <wsdl:binding
   name="WMS_SOAP_Binding" type="wms-req:WMS_XML_Port">
  <soap:binding style="document"
    transport="http://schemas.xmlsoap.org/soap/http"/>
   <wsdl:operation name="GetCapabilities">
    <soap:operation/>
        <wsdl:input>
         <soap:body use="literal"/>
        </wsdl:input>
        <wsdl:output>
         <soap:body use="literal"/>
        </wsdl:output>
        <wsdl:fault name="exception">
         <soap:fault name="exception" use="literal"/>
        </wsdl:fault>
  </wsdl:operation>
  <wsdl:operation name="GetMap">
   <soap:operation/>
   <wsdl:input>
    <soap:body use="literal"/>
   </wsdl:input>
   <wsdl:output>
    <mime:content type="image/*"/>
   </wsdl:output>
   <wsdl:fault name="exception">
    <soap:fault name="exception" use="literal"/>
   </wsdl:fault>
  </wsdl:operation>
 </wsdl:binding>
</wsdl:definitions>
```

Listing 2.2: WSDL definition for a WMS service

One of the important features that WSDL brings is the ability for a client to interact with a service without requiring custom software development or configuration. This is possible because of the standardized machine-readable service description language. In most cases, tools can automatically generate most of the code required for the client implementation, allowing the programmer to concentrate on the application logic.
Furthermore, a number of specifications such as WS-Addressing [151] and WS-PolicyAttachments [34] define bindings for WSDL. In this way, it is possible, for example, to specify the security policy for a Web service (i.e. how should the client authenticate, what parts of the message should be encrypted, etc.) using WS-SecurityPolicy [29] and attach this policy by means of WS-PolicyAttachments to a WSDL interface description. Like this, service requesters will know a priori what security tokens to include in a service request. It is even possible to have the security logic on the client side (at least partially) automatically generated by tools.
The theoretical concepts described in this dissertation are implemented using SOAP and the security services as well as the protected services are described using WSDL. The standardization of Web services description allows the realization of a flexible security framework where security services can be changed without affecting an already deployed system. Furthermore the use of WSDL tools such as WSDL2Java [42] has lowered the implementation effort.

2.1.8 Messaging: SOAP and REST

Messaging represents the core component of SOA. While other interactions (description, discovery) can take place out-of-band, interoperability at the messaging layer is critical.

SOAP

The **Simple Object Access Protocol** (SOAP) [88] is a W3C recommendation since 2003 (version 1.2). It describes a protocol for exchanging XML messages over a network. SOAP represents the foundation for the Web services stack, and as such it defines a basic messaging framework on which the upper layers (less abstract specifications) can build on. A message, as defined by SOAP, is structured in two parts:

Header This part is designed for holding control information; it contains a number of header blocks. The blocks can be used for different functions (for example the WS-* specifications leverage this feature to enhance messaging with reliability, security, etc). Furthermore, the blocks can be processed by independent nodes: each block has a target actor / role[6] as destination.

Body This is where the application payload is stored. This section can hold any number of blocks; it is also possible to have no payload.

[6]In SOAP 1.1 terminology - actor, in SOAP 1.2 terminology - role

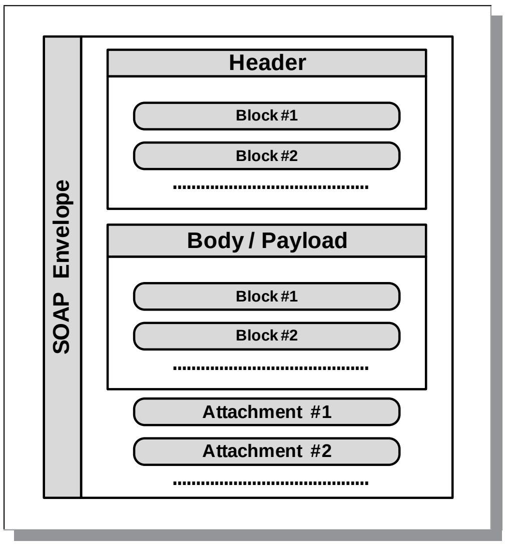

Figure 2.3: The structure of a SOAP message

Application Payload Applications are responsible for the content of the message body. For this, SOAP was designed to support two operational models:

RPC Remote Procedure Call, as the name sounds, is suitable for scenarios where the applications are more tightly coupled. In this mode, the function interfaces, together with parameters and return values are exposed by the Web service.

Document In the document model, the content of the payload is left up to the applications to define. No constraints are made on the structure of the body blocks. This model fits better more loosely-coupled scenarios.

Independent of the selected mode, two encodings are defined:

Literal In this case, the XML InfoSet is serialized and sent to the transport protocol; no information about the encoding is present in the message; only the sender and the ultimate receiver are able to interpret the content

Encoded In this case, a specific encoding is used for the payload; information about the encoding is explicitly present in the message.

According to [53], document/literal and rpc/encoded are the two frequently used styles. OGC Web services normally define document/literal bindings.

The SOAP Processing Model As opposed to the client - server model where a message is sent by the client directly to the server, when using SOAP, a message travels from the service requester to the ultimate receiver through zero or more intermediaries (see figure 2.4). The intermediaries can process the message in whole or in part (they can only modify some header blocks or they can replace the message with a totally new one).
This processing model is described in the SOAP specification itself [88]. The concept is leveraged by both ESB as well as ***SOSA*** .

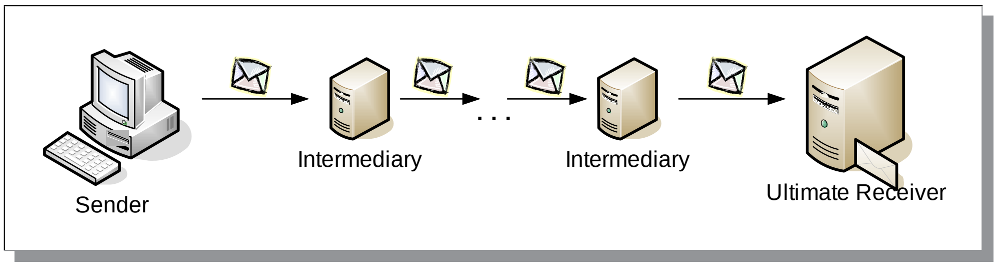

Figure 2.4: The SOAP distributed processing model

SOAP Attachments As seen in figure 2.3, besides the header and the body, a SOAP envelope can contain zero or more attachments. This makes is possible for applications to exchange messages with binary content without serializing the binary content as XML (and thus wasting performance due to parsing overhead). For this, a Multipart-MIME encoding is used and each of the attachments will be accordingly encoded. The SOAP message then references these attachments. The exact encoding is specified in [23], [30] and [150].
The OGC WMS specification recommends that returned images be sent as SOAP attachments. ***SOS!e*** (the prototype implementation of ***SOSA***) was also deployed and tested for such a service; see chapter 6 where the practical experiences are described.

Transport Protocol One of the main design considerations for SOAP was the ability to transport the messages by a multitude of transport protocols. One of the frequently used transport protocols is HTTP, because of its simplicity and wide deployment due to the Internet. This is also the binding included in the SOAP specification [89]. Listings 2.3 and 2.4 show a WFS DescribeFeatureType request and the corresponding response; the two communicate via HTTP.

```
POST /axis/services/ManhattanDataProxy HTTP/1.0
Content-Type: text/xml; charset=utf-8
Accept: application/soap+xml, application/dime, multipart/related, text/*
User-Agent: Axis/1.4
Host: iisdemo.informatik.unibw-muenchen.de:80
Cache-Control: no-cache
Pragma: no-cache
SOAPAction: ""
Content-Length: 4155

<?xml version="1.0" encoding="UTF-8"?>
<SOAP-ENV:Envelope
  xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/">
  <SOAP-ENV:Header>
    <!-- Header contents is omitted for clarity -->
  </SOAP-ENV:Header>
  <SOAP-ENV:Body>
    <ns1:DescribeFeatureType xsi:schemaLocation="..."
      outputFormat="XMLSCHEMA" service="WFS" version="1.0.0"
      xmlns:ns1="http://www.opengis.net/wfs"
      xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
    </ns1:DescribeFeatureType>
  </SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

Listing 2.3: SOAP-over-HTTP request

```
HTTP/1.1 200 OK
Server: Apache-Coyote/1.1
connection: close
date: Fri, 26 Jan 2007 19:39:46 GMT
server: Apache-Coyote/1.1
Content-Type: text/xml;charset=utf-8
Connection: close

<?xml version="1.0" encoding="utf-8"?>
<soapenv:Envelope
  <!-- Namespace declarations are omitted for clarity -->
  <soapenv:Body>
    <xs:schema version="1.0"
           attributeFormDefault="unqualified"
       elementFormDefault="qualified"
           targetNamespace="http://www.census.gov">
             <!-- Rest of SOAP message is omitted for clarity -->
  </soapenv:Body>
</soapenv:Envelope>
```

Listing 2.4: SOAP-over-HTTP response

Other protocols can also be used for transporting SOAP messages. Examples are: SOAP-over-SMTP, SOAP-over-JMS, etc. The implementation of ***SOSA*** uses SOAP-over-HTTP, mainly because the current implementations for OGC services use SOAP-over-HTTP.

REST

SOAP has come to criticism due to its complexity. Another architectural style has been proposed that better leverages the architecture of the web today (which is based on HTTP): **Representation State Transfer** (REST). The model is described in depth by Roy Fielding in his dissertation [73]. In REST software agents use URIs to invoke Web services and manipulate resources. This is easily implementable using the methods provided by the HTTP protocol: GET, POST, PUT, DELETE. For example, to search for books on eBay, one would execute a GET for the following URL ([67]):

```
http://rest.api.ebay.com/restapi?CallName=GetSearchResults&Query=book
```

Listing 2.5: eBay REST API example

The debate whether to *SOAP* or to *REST* has been analyzed by many (including [49], [43]). Furthermore, according to [49], the two architectural approaches are not irreconcilable, as SOAP 1.2 can be used in a manner consistent with REST.
OGC Web services use primarily the REST approach, mainly because SOAP was not around when the first OGC services were defined. However, new OGC service specifications also define SOAP bindings (for example WFS 1.1 - [154]).
SOSA uses the SOAP model, because it is reacher in features and more standardized. Some ***SOSA*** principles could also be applied to RESTful Web services (such as the principle of separation of concerns). However, because intermediaries are not explicitly defined in the REST model, and because REST is more appropriate for light-weight services, I consider a ***SOSA*** implementation for REST unfeasible.

2.1.9 Discovery: UDDI

Before a service can be invoked, the requester and the provider need to have knowledge of each other. This process is called discovery and refers to obtaining the information necessary to invoke a

service, including service endpoint, interface and protocol information. It is worth stating that the discovery phase can either take place automatically, using a communication / brokering protocol or by an out-of-band process.
The **Universal Description, Discovery and Integration** (UDDI) specification falls in the first category, proposing a machine-to-machine discovery protocol. It is an XML-based, platform-independent registry service for businesses. Industries and companies publish their services in the registry so that clients can then find them by means of queries. UDDI, encapsulates the functionality of the following 3 types of registries:

White Pages Listing of name, address and contact information;

Yellow Pages Listing of services indexed by service type (usually based on standardized taxonomies);

Green Pages Technical information about the services exposed by each business.

The UDDI specifications describe both a data model as well as a SOAP interface for managing registry entries (publish, modify, delete) and for querying the registry for information.
Both UDDI Version 2 ([10]) and UDDI Version 3 are standards from OASIS. The latest UDDI version is 3.0.2 ([15]) which is currently a committee draft.
Service discovery and UDDI are not directly used in this dissertation. Rather, it is assumed that the service requester and the service provider are known to each other in advance (the discovery phase has happened out-of-band). However, since the aim of this dissertation is to create a flexible security framework based on SOA, we can envisage that security services may be published in an enterprise UDDI registry. In this way, requesters could discover and integrate security services without having prior knowledge of these services. For more about this, see section 8.4 where future work is described.

2.2 Spatial Data Infrastructures

In this section I'm introducing the basic concepts about Spatial Data Infrastructures. The purpose of this introduction is to familiarize the reader with the standards and technologies related to geospatial Web services. In chapter 6 I will show how ***SOSA*** was applied to geospatial Web services in several projects.

2.2.1 Introduction

According to the SDI Cookbook [113] from GSDI the term **Spatial Data Infrastructure** (SDI) is often used to denote the relevant base collection of technologies, policies and institutional arrangements that facilitate the availability of and access to spatial data. The SDI provides a basis for spatial data discovery, evaluation, and application for users and providers within all levels of government, the commercial sector, the non-profit sector, academia and by citizens in general. The cookbook continues by stating that an SDI hosts geographic data, attributes, metadata, means to discover, visualize and evaluate the data, as well as some method to provide access to the geographic data.
Today's SDI implementations provide data as well as access to data by means of services (which are discoverable). Besides data delivery services, SDIs also implement a growing number of spatial analysis and processing services. From an architectural point of view, SDI implementations

follow a lot of the SOA principles. Actually, from a *service-oriented* point of view, an SDI is a service-based infrastructure where the exchanged data is spatial (geographical) information and the provided services are relevant to the information exchanged. As in the case of SOA, services are loosely-coupled and the publish-find-bind paradigm is followed. SDI implementations are usually deployed using the same technologies as the ones used in SOA.

2.2.2 Interoperability for Geospatial Information

The importance of interoperability when coming to using distributed, heterogeneous geospatial information was highlighted in may publications, including [112, 113, 64, 107]. As in the case of SOA, a successful SDI is based on standards for both data as well as service specifications. The main standardization organization active in the geospatial domain is the Open Geospatial Consortium (OGC), formerly known as OpenGIS Consortium[7].
As stated on their website OGC is a non-profit, international, voluntary consensus standards organization that is leading the development of standards for geospatial and location based services. Its mission is to serve as a global collaboration forum for users and developers of spatial data products and to advance the development of standards for geospatial interoperability.
OGC standards follow a set of guidelines that are collectively called The Abstract Specification. The documents are named Topics, and are intended to provide conceptual foundation for the implementation specifications. The topics cover issues such as geospatial features, imagery, metadata, service architecture and semantics.
As mentioned earlier the geospatial community is just a branch of the general IT community. Because of that, the OGC specifications rely on a couple of foundation standards which are defined by other organizations. [112] enumerates 9 of them which are listed in table 2.1. The first four are important SOA technologies, and have already been presented in 2.1. As for the rest of them, JPEG2000 and TIFF are well-known standards for image encoding (GeoTIFF extends TIFF allowing the image to be geo-referenced), the EPSG Geodetic Parameter Dataset is a categorization of coordinate reference systems and Z39.50 is a protocol used for searching and retrieving information from remote computer databases.

XML, Version 1.1 (W3C)
XML Schema, Version 1.0 (W3C)
Simple Object Access Protocol (SOAP), Version 1.2 (W3C)
Web Services Description Language (WSDL), Version 1.1 (W3C)
Geodetic Parameter Dataset, Version 6.9 (OGP, formerly EPSG)
Geographic Tagged Image File Format (GeoTIFF), Version 1.0
JPEG 2000 (ISO/IEC)
Information Retrieval Z39.50 (ISO)

Table 2.1: Foundation standards for OGC specifications

2.2.3 Standards in an SDI

[112] proposes the definition of a set of compatible standards that would represent an evolution cycle similar to the mobile communications community. As such, the paper considers the following standards to be included on the *SDI 1.0* list:

[7] http://www.opengeospatial.org

OGC Web Map Service (WMS) Produces map images of spatially referenced data from geographic information (further described in 2.2.6) [63];

OGC Web Feature Service (WFS) Delivers geographical objects (vector-based geometries together with attributes) in the form of GML documents (further described in 2.2.7) [154]; the interface can also be used to update the database behind the service by inserting / modifying / deleting geographical information;

OGC Web Coverage Service (WCS) Supports the networked interchange of geospatial data as "coverages" containing values or properties of geographic locations (from the specification [153]); it is similar to the Web Map Service interface, but instead of presenting the client with an already rendered picture, it returns the raw information allowing the client to further process this information before rendering;

OGC Catalog Services for the Web (CSW) Defines an interface to discover and query metadata about resources (data, services or other relevant resources);

OGC Filter Encoding Specification Defines an XML grammar for expressing geospatial queries. It is used by other specifications (WFS, CS-W);

OGC Geography Markup Language (GML) Defines an abstract model for expressing geographical features together with an XML-based language (further described in 2.2.4);

FGDC Content Standard for Digital Geospatial Metadata A standard from the U.S. Federal Geographic Data Committee describing an abstract model for content, relationships, obligations and repeatability of properties describing geospatial information.

Besides these standards there are many other, some of them are not as widely deployed the ones above (the Web Coordinate Transformation Service [153] is one such example), while others are currently being developed - this is the case of the Sensor Web Enablement activity that currently takes place within the OGC. Abstract Topic 12 [114] presents a taxonomy for geographic services which contains a couple of dozens of services divided in 9 categories.
Furthermore, there are also numerous activities coming from the industry where geospatial Web services are being deployed and the interfaces are made public for the community to take advantage of. This is currently the case for Google Maps [87], Google Earth [86] or Microsoft VirtualEarth [109]. Another similar initiative comes from the GIS market leader ESRI: ArcWeb Services [70]. All these industry activities are based on SOA technologies as described in 2.1.
In this dissertation we show how the ***SOS*** concept can be applied to only two service interfaces: WMS and WFS. The two are of relevance to this dissertation because they have the widest deployment[8] and because they represent two different classes of services: WMS serves bitmap images, while WFS serves vector graphics expressed in an XML language. ***SOSA*** can be applied in a similar fashion to all other service interfaces.

2.2.4 GML

The **Geography Markup Language** (GML) is an OGC standard for encoding geographic features in XML developed by OGC ([104]). This allows applications to encode geographic information in a compatible manner and thus to interoperability exchange such information. There are currently three major versions of this standard:

[8]Considering only the OGC Specifications.

Version 1 This was the first version of GML; it was defined using DTD. It is rarely used in present.

Version 2 This was the first GML version to use XML Schema. The specification [84] is relatively simple, and it is widely deployed today. [112] recommends this version to be on the SDI 1.0 list of standards.

Version 3 This version introduces a lot of new features, but it also makes the specification quite complex[9]. The last version of GML 3 is expected to become an ISO standard (19136).

GML defines an abstract model which is specified by means of XML Schema. This model consists of primitive object types such as feature, geometry, coordinate reference system, time, etc. In order to use GML, applications need to define a so called *application schema* which defines the concrete objects that are to be encoded using GML. In GML language, these are called features. For example an aviation application would define features such as runway, taxiway, heliport, radar station, etc. The application schema (also defined as XML Schema) references the base GML types, but this schema is customized for the application - it defines for each feature type the exact attributes as well as the associated geometry. It is important to note that GML defines the syntax but not the semantics.

Features In GML features are the representation of a physical object. They can have zero or more attributes and zero or more geometries. To continue our example, a runway can be described as polygon and have several attributes describing technical aspects such as the types of plane that are able to land on it or the date of the next scheduled repair.

Geometries The types of geometries supported by GML vary depending on the GML version. Version 2 supports point, line string and polygon. GML 3 introduces more complex geometries such as curves. The coordinates of the points forming the geometries are defined relative to a coordinate reference system which needs to be explicitly specified. For this purpose GML uses the EPSG Geodetic Parameter Dataset (see table 2.1).

GML Profiles These are restrictions of the GML specifications (defined as either schema or as text document) and were introduced in GML 3, in part because of the complexity of this specification. They are intended to simplify the specification for those applications that do not need complicated features. Several profiles exist: Point Profile (only defines point geometry), Simple Features Profile, GML in JPEG2000 (for georeferencing JPEG2000 imagery), GML profile for GeoRSS (for including geographical references in RSS feeds).

GML Usage GML serves as an encoding format for exchanging geographical information and as such numerous GIS applications currently support it. Some of the profiles are also used in applications not related to the GIS world - for example the GML profile of GeoRSS or the Point Profile. However, here GML presents interest because it is the data encoding format for the Web Feature Service. The data returned by this service is GML. Furthermore when new objects are inserted or when existing objects are modified, the input data is expected to be GML encoded.

[9]The GML 3.1.0 specification document [85] has 601 pages.

Examples In order to give the reader a feeling of GML, an example taken from the GML 2.1.2 specification [84] is provided. Listing 2.6 shows fragments of a *GML application schema*. It contains the definition of a GML feature called Dean, having several attributes, one of them being its location described by a point. Listing 2.7 shows a document conforming to the schema (in GML terminology a *GML document instance*).

```
<element name="Dean" type="ex:DeanType" substitutionGroup="gml:_Feature"/>

<complexType name="DeanType">
 <complexContent>
  <extension base="gml:AbstractFeatureType">
   <sequence>
   <element name="familyName" type="string"/>
    <element name="age" type="integer"/>
    <element name="nickName" type="string" minOccurs="0"
     maxOccurs="unbounded"/>
    <element ref="gml:location"/>
   </sequence>
  </extension>
 </complexContent>
</complexType>
```

Listing 2.6: GML application schema example

```
<Dean>
 <familyName>Smith</familyName>
 <age>42</age>
 <nickName>Smithy</nickName>
 <nickName>Bonehead</nickName>
 <gml:location>
  <gml:Point>
   <gml:coord>
    <gml:X>1.0</gml:X>
    <gml:Y>1.0</gml:Y>
   </gml:coord>
  </gml:Point>
 <gml:location>
</Dean>
```

Listing 2.7: GML document instance example

Relation to this Dissertation

This dissertation shows how the ***SOS*** concept can be applied to SDI. Chapter 6 shows how WMS and WFS services can be enhanced with security functions. Because the WFS specification uses GML as encoding format, the ***SOS*** concept and the implementation should accommodate the specific requirements of GML.
One of the unique aspects of designing security systems for SDI services, is the geographical aspect of the data. This has implications particularly on the authorization process, where data should be restricted for specific geographical areas. This has been investigated in [107] and the results have been further developed in this dissertation.
Other challenges include the GML encoding, and the fact that little has been done to secure it. In chapter 6, I will show how GML can be combined with XML Signature [159] and XML Encryption [160] to accommodate a land register information system scenario.

2.2.5 OGC Service Specifications

All OGC service interfaces follow the same design principles defined in the OGC Web Services Common specification [115]. This document specifies aspects that are, or should be, common to all or multiple OWS interface Implementation Specifications, particularly the contents of operation requests and responses, the parameters included in the requests and the format of the corresponding responses. Additionally, two different encodings are defined for parameters: XML and KVP[10]. Other common aspects of OGC services are defined in the Abstract Specification, Topic 12 [114].
Because some of the OGC services were defined before most SOA technologies became popular, these services currently do not adhere to the SOAP / WSDL / UDDI technologies. For messaging the OGC services mainly use the REST paradigm (as presented in section 2.1.8), for service description they use an own mechanism (capabilities documents) and for service discovery an own registry service is defined (the catalog service). The OGC currently invests effort into correlating its specifications with the mainstream IT technologies.
The general message exchange pattern for an OGC service is depicted in figure 2.5. Before interacting with the service, a client will first make a getCapabilities request. The response to this service call is an XML document containing the service metadata. By parsing this information the client will be able to make valid requests to the service. The interaction continues with service specific requests (for example map requests).

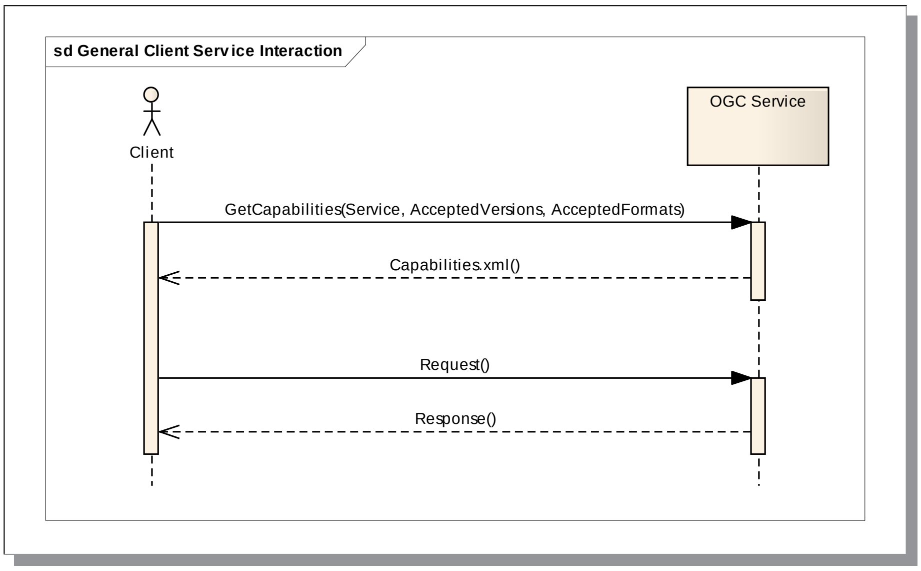
sd General Client Service Interaction
Client
OGC Service
GetCapabilities(Service, AcceptedVersions, AcceptedFormats)
Capabilities.xml()
Request()
Response()

Figure 2.5: General message exchange pattern for OGC services

Capabilities Documents As opposed to WSDL documents that only contain information referring to the service, OGC capabilities documents also contain information describing the data served by the service. A typical Capabilities document will contain metadata describing the service (administrator, etc.), metadata describing the operations as well as metadata describing the geographic data offered by the service.

Bindings [115] defines recommended encodings for requests using HTTP GET and HTTP POST. Most service specifications usually define these bindings, but some of them also define other bindings

[10]Key-Value-Pair

in addition to those. For example the WFS 1.1 specification defines a SOAP binding and the CS-W 2.0 specification defines a Z39.50 binding.

HTTP GET Binding

The HTTP GET binding follows the REST paradigm where requests are expressed as URLs. The parameters are encoded as Key-Value-Pair and the request is transmitted as an HTTP GET request. The response is the HTTP message response. A request example is provided in listing 2.8, where a GetCapabilities request is executed for a WCS service.

```
http://hostname:port/path?SERVICE=WCS&REQUEST=GetCapabilities&
  ACCEPTVERSIONS=1.0.0,0.8.3&SECTIONS=Contents&
  UPDATESEQUENCE=XYZ123&ACCEPTFORMATS=text/xml
```

Listing 2.8: HTTP GET example

HTTP POST Binding

The HTTP POST binding was developed because of the shortcomings of the GET binding, namely the fact that URL length are limited (not by the HTTP specification itself [130], but by the current implementations [50]). This makes it impractical to have parameters with large data values URL-encoded. For this, an XML encoding is defined for the request. This XML message is then sent via HTTP POST to the service. The response is the HTTP message response. This binding is very similar to SOAP-over-HTTP, the only difference being how the request is encoded.
A request example is showed in listing 2.9. Here a client requests the schema definition for feature types TreesA_1M and RoadL_1M from a WFS.

```
<?xml version="1.0" ?>
 <DescribeFeatureType version="1.1.0" service="WFS">
     <TypeName>ns01:TreesA_1M</TypeName>
     <TypeName>ns02:RoadL_1M</TypeName>
 </DescribeFeatureType>
```

Listing 2.9: HTTP POST request example

SOAP Binding

The SOAP binding is not currently defined in [115]. However some of the newer specification define it (for example WFS 1.1). Furthermore, there are other documents motivating why this binding should be introduced and describing how it shall be realized ([66, 79, 141]).
[141] investigates the possibilities and recommends that the SOAP bindings for all OGC service follow the document/literal encoding style. For this, the body of the message shall contain the XML POST encoding for the request. The response will be transmitted as SOAP message and, depending on the content type, it will be either placed in the body section (if XML) or as attachment (if it is an image or other media type). Listing 2.10 shows the same request as in the previous section, encoded as SOAP message.

```
<?xml version="1.0" ?>
<soap:Envelope xmlns:soap="...">
  <soap:Header>
   ... ... ...
  </soap:Header>
  <soap:Body>
    <DescribeFeatureType version="1.1.0" service="WFS">
      <TypeName>ns01:TreesA_1M</TypeName>
      <TypeName>ns02:RoadL_1M</TypeName>
    </DescribeFeatureType>
  </soap:Body>
</soap:Envelope>
```

Listing 2.10: SOAP request example

The advantages of using SOAP are the fact that one can leverage the WS-* specification and the processing model defined by SOAP.

Relation to this Dissertation

The implementation of the ***SOS*** concept is based on SOAP. Furthermore there are numerous security standards (such as WS-Security and its extensions, SAML, etc.) that are mainly used together with SOAP. For these reasons, only the SOAP profiles for the OGC services are considered in this dissertation.

2.2.6 WMS

The **Web Map Service** (WMS) produces maps of spatially referenced data dynamically from geographic information [63]. The maps are either bitmap, already rendered images such as PNG, GIF, JPEG or vector graphics such as SVG or WebCGM. The use of transparency for background allows a client to overlay maps from different services.

Service operations

The WMS interface defines the following three operations:

GetCapabilities Returns metadata about the service operations and its data. The data is structured in layers that form a hierarchical structure. The capabilities document contains information intended for both machines (output image formats, layer IDs, etc.) as well as information intended for humans (textual description for layers, contact information for administrator, etc).

GetMap Returns a map representation of the data from a requested area. The area is specified as bounding box, which is described by the coordinates of the two corner points and the coordinate reference system. Other arguments include: the image output format, dimensions of the image, the layers (in order), a rendering style for each layer, background color, transparency of the background, etc.

GetFeatureInfo This operation is optional. It requires the X,Y coordinates of a point and it returns information about the features located at the specified coordinates.

2.2.7 WFS

The **Web Feature Service** (WFS) allows a client to retrieve and update geospatial data encoded in Geography Markup Language (GML) from multiple Web Feature Services [154]. It defines operations for data access and operations for data manipulation. [154] defines the following three types of WFS Service:

Basic WFS Provides only operations for access to geographical data as well as metadata; the supported operations are GetCapabilities, DescribeFeatureType and GetFeature;

XLink WFS In addition to the basic WFS implements the GetGmlObject operation;

Transactional WFS In addition to the basic WFS it provides data manipulation functionality; the additional operations are: LockFeature, Transaction.

Service operations

The WFS interface defines the following six operations:

GetCapabilities Returns metadata about the service and the geographical data it serves. The geographical data is structured in feature types, representing different classes of geographical objects. Each feature type has associated metadata (description, bounding box, etc.) as well as an enumeration of the supported operations for this feature type. Additionally, capabilities documents contain information about the filter operations implemented.

DescribeFeatureType Returns an XML schema description for the requested features

GetFeature Returns features in GML format (it is possible to specify the desired GML version). A client can formulate complex queries that contain both spatial as well as non-spatial conditions which are encoded using the Filter Encoding specification.

GetGmlObject Allows retrieval of features and elements by ID from a web feature service. A GetGmlObject request is processed by a WFS, and an XML document fragment, containing the result set, is returned to the client. The GetGmlObject request provides the interface through which a WFS can be asked to traverse and resolve XLinks to the features and elements it serves. [154]

LockFeature Is used in combination with the transaction operation to lock a feature so that data hazards are avoided and consistency is ensured.

Transaction Allows data manipulation using the WFS interfaces. The operation supports insert, delete and update operations.

2.3 Enterprise Services Bus

Enterprise Services Bus refers to both a software architecture and a class of software products used for the realization of SOA. The concepts behind ESB are described in depth in [56, 100]; [134, 138] also give a good overview of the concepts. This section will only introduce the basics about ESB.
I start with a couple of definitions, then introduce the ESB architecture, present its capabilities and list the architectural patterns that are associated with ESB. This leads to a number of reusable services that are described at the end of the section.

2.3.1 Definitions

Message Oriented Middleware (MOM)

According to Webopedia [3] MOM is a type of software that connects two otherwise separate applications. Furthermore, [56] states that Message Oriented Middleware is a concept that involves the passing of data between applications using a communications channel that carries self-contained units of information (messages). In such a system, communication usually takes place asynchronously. Applications rely on the MOM system to deliver the messages to their destinations. Example of MOMs are JMS, SOAP Web services, CORBA, etc.

Enterprise Services Bus (ESB)

According to Webopedia [3] ESB is an open standards-based distributed synchronous or asynchronous messaging middleware that provides secure interoperability between enterprise applications via XML, Web services interfaces and standardized rules-based routing of documents.
According to [56] the ESB provides a highly distributed, event-driven Service Oriented Architecture (SOA) that combines Message Oriented Middleware (MOM), web services, intelligent routing based on content, and XML data transformation.

2.3.2 Characteristics of an ESB

The ESB (see figure 2.6) is a MOM and as such it provides the connectivity layer between service providers and service consumers. It is the duty of the ESB to deliver messages from the requester to the ultimate receiver.
The ESB provides a further abstraction layer on top of the communication protocol facilitating the integration of applications developed using different MOMs. This is possible due to a common format for all messages circulating on the bus. For those applications that do not natively support the common messaging format, adapters (connectors) are provided that transform messages from their native format to the format of the ESB before placing them on the bus.

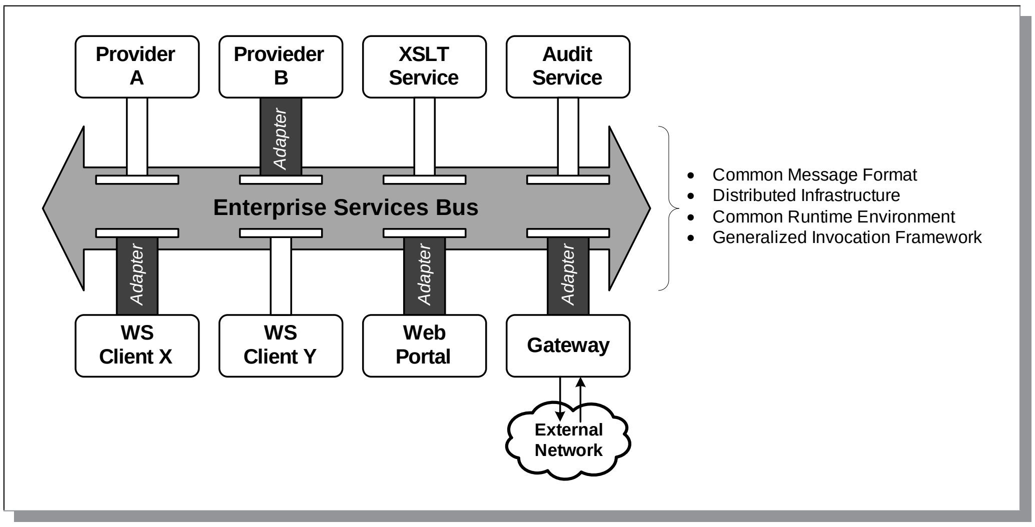

Figure 2.6: Enterprise Services Bus

[134] recognizes the following as being the most important characteristics of an ESB:

- The use of explicit implementation-independent interfaces to define services;
- The use of communication protocols that stress location transparency and interoperability;
- The definition of services that encapsulate reusable business functions.

Figure 2.6 shows an ESB connecting two service providers, two client applications and two infrastructure services - the XSLT service and the Audit Service (these services can be integrated in workflows together with other services). Additionally, two other client variations are showed: a Web Portal - user interactions are mapped to Web service invocations, and a Gateway which connects the ESB to an external network (possibly converting messages from other protocols and at the same time providing some security features).

ESB vs. Point-to-Point

The first integration architecture that emerged was point-to-point (showed in figure 2.7). In such an architecture each component will integrate with the other components separately (there is no before-planning). Because of this, in literature [56] it is sometimes called the *accidental architecture* suggesting that this is how integration evolves if it is not planned in advance. This architecture is typical for tighter-coupled components, such as when using Remote Procedure Call.
The disadvantages of such an architecture are obvious: if we expect n components to intercommunicate then each component needs to implement $n-1$ interfaces, bringing the total implementation effort at $n(n-1)/2$ implementation modules. Of course, in practice, not always all the components need to intercommunicate, but nevertheless this architecture is impractical for systems containing more than a few components.

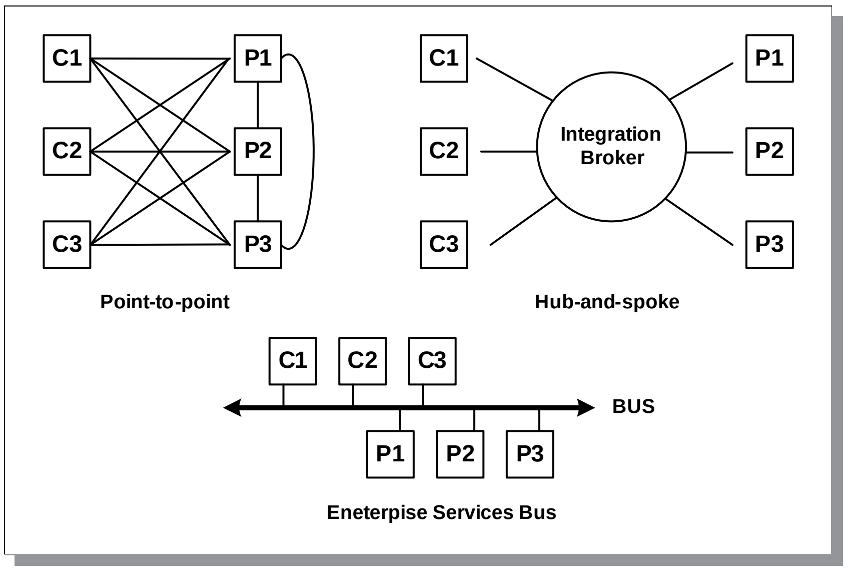

Figure 2.7: ESB vs. point-to-point and hub-and-spoke

ESB vs. Hub-and-Spoke

Another integration architecture is hub-and-spoke. The ESB is often compared in literature ([56, 134, 100]) with this more traditional integration approach. Some of the advantages of the ESB over the hub-and-spoke approach are:

- In hub-and-spoke there is one monolithic broker application which does all the integration. This provides a single point of failure. In contrast, in an ESB the service invocation is distributed (there is no single point of failure).

- The ESB architecture is more scalable because of its distribution. In an ESB it is possible to deploy services on an as-needed basis.
- ESB is based on standards in contrast to the hub-and-spoke solutions which are usually based on proprietary solutions.

2.3.3 Capabilities of an ESB

As a middleware system, ESBs usually provide several capabilities. These capabilities can used by the applications that are integrated by means of an ESB with little or no effort. The work described in [134, 56] lay the foundation for the following.

Communication Capabilities

Message Routing This can be either content-based or itinerary-based.

Messaging Styles Usually both synchronous as well as asynchronous messaging are supported.

Message Exchange patterns Several patterns are usually supported: request-reply, one-way only, solicit-response, notification.

Transport Protocols Several protocols are usually implemented: HTTP, SMTP, JMS as well as proprietary protocols such as IBM MQ, Microsoft MQ, etc.

Quality of Service Capabilities

Message Delivery This includes support for delivery models such as once-and-only-once, at-most-once, at-least-once, etc. as well as constraints such as "messages are to be received in the same order in which they were sent" or "failure to deliver a message be made known to both the sender and receiver." Because these assurances need to be fulfilled even in the case of system failure, asynchronous messaging and store-and-forward delivery are usually requirements. Message delivery is tightly bounded to the transport protocol - some of the transport protocols support reliable messaging by design. Were this is not the case, WS-* specifications can be used, namely the WS-Reliability standard from OASIS ([16]).

Transactions This includes support for both short duration as well as longer running transactions; for each of these various transactional models are possible. These can either be supported by proprietary middleware technologies, or through the WS-Coordination framework, which includes WS-AtomicTransaction [24] (for short lived, ACID transactions), as well as the WS-BusinessActivity specification [25] (for longer running business transactions).

Integration Capabilities

Orchestration This includes both the orchestration runtime environment as well as editors for orchestration scripts. The most common orchestration script is BPEL4WS.

Choreography Supported through choreography specifications. Message routing is one alternative for realizing choreographies.

Integration Connectors Different ready-to-use connectors for integration with different types of applications and databases. These connectors are integrated in the application by means of APIs and allow the application to be plugged to the bus.

Protocol Transformation Similar to connectors but at the protocol level; they allows for different protocols to be converted to the common messaging format of the bus.

Various Programming Languages API support for different programming languages (C, C++, Java, C#, etc).

Management and Infrastructure Capabilities

Publishing / Discovery Typical to SOA, these features allow for services to be invoked at run-time without previous knowledge. They are usually realized by means of registries (such as UDDI) and description languages (such as WSDL).

Monitoring Ability to monitor specific parameters (in Java this can be realized by means of JMX).

Metering Ability to meter specific service parameters.

Logging Used for debugging as well as audit purposes.

Load-Balancing The ability to balance the load among different service providers.

Fault-Tolerance Because of it's distributed nature, fault-tolerance comes as a really important aspect.

Security Most ESBs come with some support for authentication, authorization as well as cryptographic functions.

2.3.4 Enhancing Service Capabilities through Mediation

Because it relies heavily on message routing, ESBs are sometimes compared to decentralized proxies that have the ability to manage, monitor or enhance the capabilities of services. In a typical interaction, a message from a service requester will be routed by the ESB through several intermediary services before finally reaching its destination (see figure 2.8). Each of these intermediary services will apply some transformations to the original message. Thus, from this perspective, the ESB services can be seen as a list of chained proxies. They manage, monitor or enhance the capabilities of the destination service.

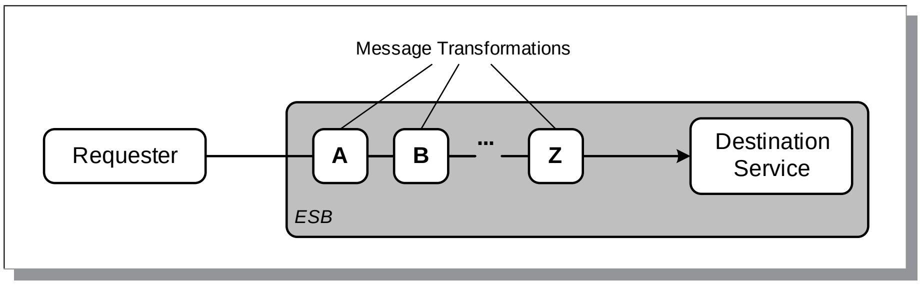

Figure 2.8: Mediations in an ESB

In reality, the ESB is a lot more than a chained list of simple proxies because different message routing patterns can be applied - thus dynamically assigning the intermediary services that process a certain message (and their order) and messages can be split and aggregated (see next section) - thus resulting in more than one execution chain. Nevertheless the concept of monitoring, managing and enhancing the capabilities of a service through consecutive transformations of the request / response messages is fundamental for the ESB architecture [56, 134, 92].

Examples of enhancing the capabilities of a service through service-mediations are: supporting several transport protocols through protocol conversion or advertising different service interfaces through data type conversion. As more complex transformations are available, other service capabilities such as security functions (identity, authorization, etc.) and quality of service (reliability, availability) can be supported through ESB mediations [92]. In this dissertation, I'm investigating how ESB mediations can be used to implement and manage the security of services.

2.3.5 EAI Patterns in an ESB

There are many integration patterns for messaging systems; [95] lays the foundation work for this. Most of these can be applied in the context of SOA / ESB. However, due to the model induced by the ESB architecture, I consider the following six patterns to be most relevant.

Content-Based Routing

In this pattern (see figure 2.9), after the message has arrived it is inspected and, depending on its content, it is forwarded to one service or another. This type of routing is similar to the routing of IP packets with the remark that IP packets are only routed according to one field of the message (the IP address). Routing policies for messages can be more complex as any part of the message can be used for routing. If messages are XML-based, then routing policies can be easily defined using XPath expressions.

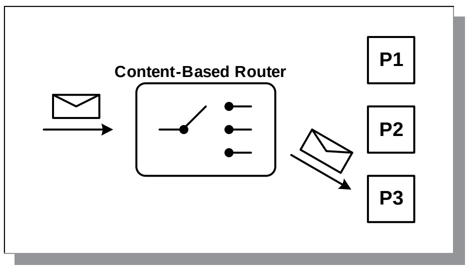

Figure 2.9: The content-based routing pattern

Itinerary Routing

In this pattern (see figure 2.10), a complete itinerary is attached to the message before the message being forwarded to the first process. The message then follows the path described by the itinerary (each process will forward the message according to the itinerary).
Note that itinerary routing can be combined with content-based routing, so that the itinerary for the message to follow is determined based on the content of the message.

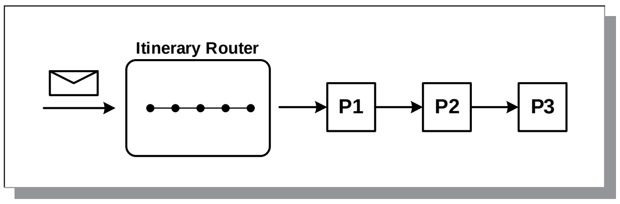

Figure 2.10: The itinerary routing pattern

Dynamic Routing

Dynamic routing (see figure 2.11) is a variation of content-based routing where the routing polices change based on execution events. We can continue the analogy to the IP network where most of the routing policies are dynamically determined.

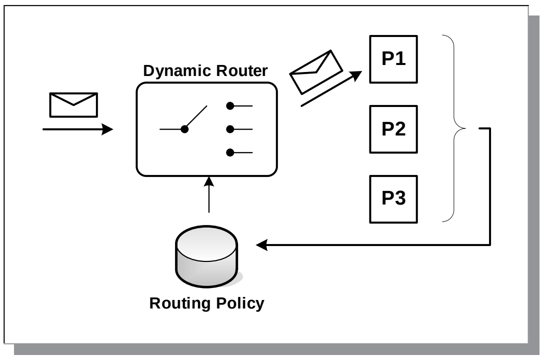

Figure 2.11: The dynamic routing pattern

Splitter/Aggregator

In the case of a splitter (figure 2.12) a message is divided into several messages which then follow independent paths. The reason for splitting the message can be for example a divide-et-impera strategy where a bigger problem is divided into smaller, less complex, subproblems. This also follows the ESB re-usability principles.

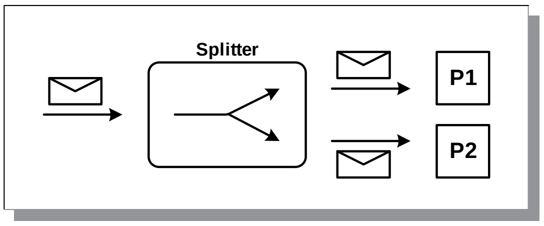

Figure 2.12: The splitter pattern

The opposite service is an aggregation service (figure 2.13) which does a join operation on messages. Several messages are combined together and the results are forwarded as a single message. As opposed to the split operation which is always executed without delays, when aggregating several messages, the service needs to wait until all messages are received before producing the result.

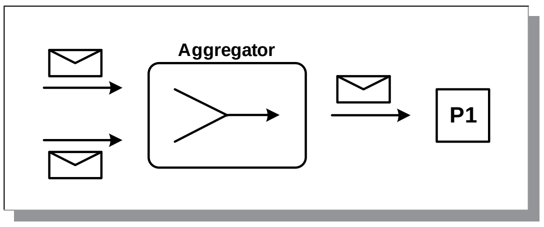

Figure 2.13: The aggregator pattern

Orchestrator

After receiving a message, an orchestration service (figure 2.14) invokes several services before forwarding the resulted message. The service should not only be able to invoke the services, but also deal with synchronization issues and error-handling.
The effect is similar to the itinerary routing pattern, in that several services are executed. However in itinerary routing the execution is done in a distributed manner, while in this case the execution is centralized. This allows an architect to more easily define complex invocation scripts (such as the ones containing branches, loops, etc). A popular language for expressing orchestration scripts is BPEL4WS.

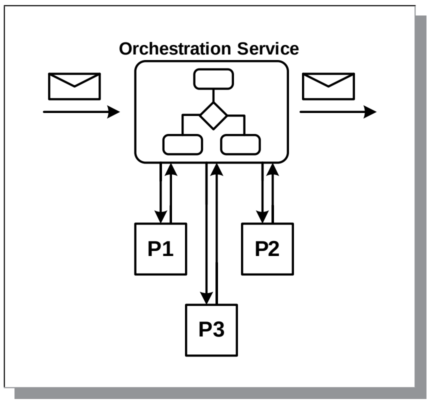

Figure 2.14: The orchestrator pattern

Message Transformation

Message transformation (also called mediation) involves changing the content of the messages. The following three types of transformations are most important:

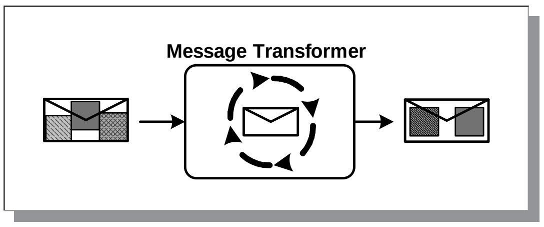

Figure 2.15: The message transformer pattern

Envelope Wrapper The original message is wrapped by an enveloping message. The enveloping message usually provides extra-features that are used through the rest of the processing chain. One such example is a service converting OGC HTTP POST requests to SOAP messages.

Normalizer The original message is converted into a reference format. This pattern is applied when plugging new applications to the ESB: messages generated by these applications are converted from their native format into the common messaging format used by the ESB.

Content Transformer The content of the original message is modified. We distinguish two important transformations:

Enriching In this case extra information is added to the message. For example, a gateway may extract information from the transport protocol and the append it to the message; an example of such information is authentication information for a SOAP message travelling over HTTPS/SSL.

Filtering In this case parts of the message are removed. An example is a policy enforcement point which removes parts of a request in order to enforce the security policy.

2.3.6 Infrastructure Services in an ESB

From the capabilities and the integration patterns previously illustrated, we come to the conclusion that some of the features that an ESB is equipped with should be realized as services. This will allow other services to leverage their functionality using the above presented integration patterns. [56] recognizes the following three as possible infrastructure services.

Orchestration Service An orchestration service follows the previously described orchestrator pattern. Because orchestration languages are being standardized (i.e. BPEL4WS) and because many implementations for runtime environments are now available, an orchestration service represents a quick and simple way to aggregate several services. This task is simplified by the availability of graphical tools for writing orchestration scripts and viewing how the processes execute.

Message Transformation Service A message transformation service implements the content transformer pattern. If bus messages are encoded in XML, then XSLT can be used to specify the mediations, making this a very flexible and standards based solution. As XSLT transformations represent a significant computational-load, these services can be replicated and distributed over the network (perhaps on more powerful machines having specialized hardware).

Message Storage and Caching Service The implementation of reliable messaging requires messages to be stored in the case of a system failure (when the service is back, the messages are forwarded). For this, a message storage service is required. This can be equipped with caching services in order to boost performance. This is a common functionality that can be leveraged by many applications.

2.3.7 Implementations

There are numerous implementations of the ESB model. Most of them are commercial (some of these are updated versions of the integration brokers used a decade ago), but open-source implementations are also emerging. Some of the most important implementations are listed below:

- Cape Clear
- IBM WebSphere MQ
- Mule (open-source)

2.3.8 Relation to this Dissertation

ESB is the architecture of choice when it comes to integrating security services in this dissertation. Although other architectures are possible (for example hub-and-spoke), ESB has been chosen because of the following arguments:

- ***SOSA*** involves splitting the security system into small reusable services. ESBs are more appropriate for integrating services with little functionality.

- As mentioned in section 2.3.2, ESBs are more oriented toward standards. In ***SOSA*** , both messaging standards and security standards are very important. Furthermore, the bus architecture provides an efficient way to join standards together by means of mediations, which makes it possible to realize systems supporting multiple security standards ([35, 60] underline that multiple security standards is one of the key requirements for Web services).

- The common bus messaging format, combined with the integration patterns presented in section 2.3.5 result in a relatively simple design model for systems. We will show later how this model can be leveraged in designing security systems.

Furthermore, besides a convenient architectural model, ESBs also provide a number of infrastructure capabilities that can be leveraged by security systems. Some of these capabilities are present at the protocol level (such as quality of service assurances), while others are provided as services (as seen in section 2.3.6).

2.4 Security in the Context Web Services

In this section, I'm introducing the basic concepts related to information security. I'm starting by defining the purpose of security, then analyzing the different types of threats addressed to information both while it is in transit, as well as when it is at repose. After this, the main security functions are presented, together with the related technologies and standards that are applicable in the context of Web services.

2.4.1 The Security Stack

In literature it is often said that no security system is 100% secure. However if the system's designers take the possible threats into consideration and employ adequate protection strategies, an acceptable level of security can be achieved. In this section we will look at the possible threats and will present technologies that can be deployed as countermeasures.
In the case of Web services, the goal of a security system is to protect information. According to [71, 136], we can assume that information is found in one of two distinct states: at repose or in transit. For each of the two states, different threats are relevant. The two models are presented in the following sections.

Information at Repose

Information at repose can either be in storage (files, databases, etc.) or be in the course of processing. In both cases, the possible threats are addressed to the system which hosts the data.
Information can be processed by either a Web service or a client application accessing a Web service. Figure 2.16 shows how these run on top of a Message Oriented Middleware, which in turn runs as an application inside an Operating System, which runs on a hardware computer system. Information in storage (databases and files) resides inside the Operating System. This approach to structuring security is also discussed in [118, 117].

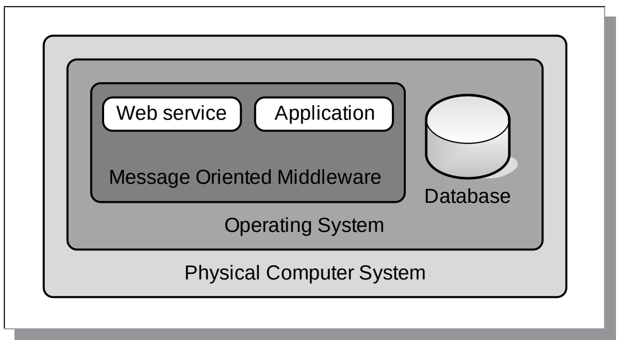

Figure 2.16: The security stack: Data at repose

Physical Computer System At this layer, threats can come from malicious persons that have access to the machine where the application is running. A further threat is represented by hardware failures. Countermeasures for these threats consist of access restrictions to hardware equipment together with redundancy (backup, uninterruptible power supply, etc.).

Operating System At this layer, threats can come from malicious persons with access to the computer (most operating systems support multiple users). Further threats are represented by viruses, Trojans and other software that can interfere with the MOM or the stored information. It is important to remark that the operating system is situated between the human person (user of the client application or administrator of the Web service) and the Web service application. Countermeasures are good administration, anti-virus and anti-spyware software, intrusion detection and intrusion prevention systems, etc.

Message Oriented Middleware MOMs implement the Web services protocols and represent the interface between the application and the network. They can be implemented as either application servers (AXIS / Tomcat) or as lightweight libraries. In both cases, threats are represented by malicious persons and misconfiguration of the MOM.

Web Service At this layer we place the Web service or (on the client machine) a software application accessing a Web service. For this layer, the security functions described in 2.4.2 are relevant.

Information in Transit

By means of Web services, software systems exchange information over a network. The information is segmented into messages which travel by means of various network protocols. Threats can be targeted at any of these protocols.
Figure 2.17 presents the 7 layer OSI stack together with some relevant protocols for each layer. OSI does not further specify the application layer, however I chose to segment this layer into two sub-layers, because Web services are traveling on top of transport protocols, such as HTTP, FTP, SMTP, etc.
Firewalls are the most widely deployed countermeasure for preventing network threats. Depending on the network layer where they act, [71] recognizes the following types of firewalls:

Packet-Filtering Firewalls These firewalls work at layer 3 of the OSI stack and filter IP packets based on information found in the header of the package (source, destination, port).

Circuit-Level Firewalls These work at layer 4 (TCP) of the OSI stack and are able to monitor TCP handshaking and TCP sessions, and filter traffic based on more sophisticated criteria.

Application-Level Gateways Work at the top layer of the OSI stack and are able to understand the traffic for specific application protocols. One such example are the XML firewalls which are firewalls designed to filter SOAP traffic. Typical tasks for them are XML validation or protection against common types of attacks (replay attacks, DoS attacks, etc). Because XML firewalls understand the SOAP protocol, they often offer further capabilities to protect information passing through the firewall such as XML encryption or the ability to digitally signing messages.

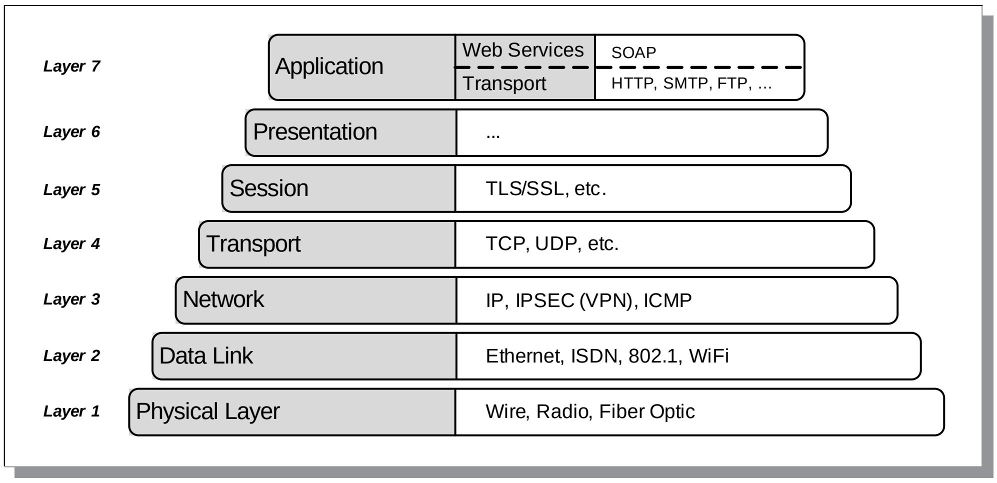

Figure 2.17: The security stack: Data in transit

Each of the seven layers has some security features built in by design: some data link protocols (such as WiFi) support data encryption and authentication, at the network layer IPSec can be deployed, while at the session layer TLS/SSL is very popular; furthermore, at the transport layer there are several possibilities. Normally the following rule applies: the lower the layer, the coarser the mechanisms. However, as [72] remarks, current implementations of Web services usually take advantage of the security features of the lower layer protocols (TLS/SSL, HTTP authentication are examples) because current message-level security implementations are relatively poor in performance.
As also remarked in [62], the distributed processing model of SOAP coupled with the protocol layering model presented here has implications on the architectural design of security systems for Web services. Figure 2.18 shows how one message can be processed by several nodes, and how it can travel on different protocols between these nodes, each of these protocols conveying some security information. In such cases, solutions need to be developed for the security information to travel end-to-end and not only peer-to-peer.

Relation to this Dissertation

In this dissertation I discuss how security functions can be externalized and implemented as reusable services. For this, we are looking at securing information while it is in transit and while it is processed by systems. Physical security, Operating System security and MOM security are out of the scope of this dissertation.

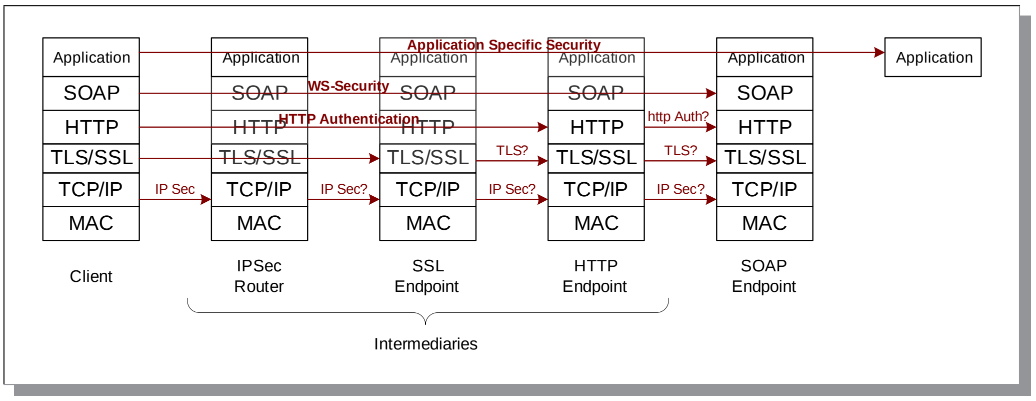

Figure 2.18: Intermediary nodes processing security information

This dissertation addresses the security at the Web Services sub-layer of the application layer. The purpose here is to architect a distributed security system, composed of reusable security services which satisfies the security requirements for Web services. Integrating lower layer security with security at the messaging layer (as exemplified in the previous section) is considered as one requirement for the security system.

2.4.2 Security Functions

In the following I'm introducing the security functions. For each function I will give a definition from the literature, describe it and enumerate the relevant technologies and standards in the context of Web services.

Authentication

Definition According to [131], **authentication** is the process of verifying an identity claimed by or for a system entity. The same source recognizes that authentication consists of the following two steps:

Identification At this point an entity presents an identifier to the security system. The purpose of this step is for the security system to be able to differentiate the requesting entity from the other exiting entities. Identification is vital to other security functions: authorization, audit, accounting.

Verification Presenting or generating information that corroborates the binding between the identifier and the entity.

There are numerous technologies and standards that address this issue in the context of Web services. These technologies address both the messaging layer as well as the transport protocol. Popular technologies that work at the transport layer are IPSEC, SSL/TLS and HTTP Authentication. At the messaging layer WS-Security [35] addresses authentication by providing the possibility to attach various security tokens to SOAP messages.
In Web services, software agents authenticate to services. The agents can act on behalf of a human person or on behalf of another service. In the case of a human, the authentication depends on 4 factors: something a user is (e.g. fingerprint), something a user has (e.g. private key), something a user knows (e.g. password) or something a user does (e.g. voice recognition).
Some widely deployed methods for authentication are the following: username / password, PKI /

digital signatures (implemented in various technologies - WS-Security, SSL, etc.), Kerberos, SAML, LDAP, RADIUS, etc. The number of authentication methods represents a challenge when designing security systems for Web services, because systems are expected to work with several authentication methods. In order to accomplish this, it is often necessary to map identities or to translate security tokens.

Authorization

Definition (1.) An **authorization** is a right or a permission that is granted to a system entity to access a system resource. (2.) An **authorization process** is a procedure for granting such rights. (3.) To **authorize** means to grant such a right or permission [131]. In literature [18, 147, 21], authorization is divided into the following two steps:

Decision Knowing the resource to be accessed and its context, based on a set of credentials provided by the requester, decide whether to grant access or not. The decision is based on an authorization policy.

Enforcement Enforce the decision taken in the previous step.

There are various models for designing and implementing security policies: *Discretionary Access Control (DAC)* - where the identity of the requester is stored together with its permissions, *Mandatory Access Control (MAC)* - where access is regulated based on a mandated regulations determined by a central authority and *Role-Based Access Control (RBAC)* - where users are grouped together into roles and permissions are assigned for each role. Policy models are detailed in [137] and do not constitute an aspect of this dissertation.
The most important authorization frameworks in the context of Web services are the following:

AAAArch [132, 147, 148] - describes an architectural framework for the authorization of Internet resources and services. The model adheres to the principle of separation of concerns and divides authorization into 4 services: Policy Retrieval Point, Policy Information Point, Policy Decision Point and Policy Enforcement Point. This segmentation suggests two other steps in the authorization process, namely the retrieval of the policy and gathering information against which policy conditions are to be evaluated.

XACML [18] - proposes both a policy language (XML based) and a service model. The service model contains the following entities: Policy Administration Point (similar to the Policy Retrieval Point), Policy Information Point, Policy Decision Point and Policy Enforcement Point.

SAML [22] - proposes a general purpose framework for exchanging security information. SAML can be used in combination with XACML, for encoding both policies and authorization decisions.

Confidentiality

Definition Referring to data, **confidentiality** is the property that information is not made available or disclosed to unauthorized individuals, entities, or processes [131].
Confidentiality is achieved by means of encryption. As Web services are XML-based, the most important encryption technologies relevant to Web services are XML-ENC [160] and WS-Security [35]. The latter describes how XML-ENC should be used in conjunction with SOAP messages.

Integrity

Definition Referring to data, **integrity** is the property that data has not been changed, destroyed, or lost in an unauthorized or accidental manner [131].
In practice, implementations for determining data integrity rely on hashing algorithms and digital signatures. The relevant technologies for Web services are XML-DSIG [159] and WS-Security [35] (WS-Security describes how XML-DSIG can be applied to SOAP messaging).

Non-Repudiation

Definition **Non-Repudiation** is the capability to ensure that a transferred message has been sent and received by the parties claiming to have sent and received the message [3]. According to [131] there are two types of non-repudiation:

Non-Repudiation with Proof of Origin Provides the recipient of data with evidence that proves the origin of the data, and thus protects the recipient against an attempt by the originator to falsely deny sending the data;

Non-Repudiation with Proof of Receipt Provides the originator of data with evidence that proves the data was received as addressed, and thus protects the originator against an attempt by the recipient to falsely deny receiving the data.

The most important methods to achieve non-repudiation are: digital signatures, confirmation services and time-stamps.

Privacy

Definition **Privacy** is the right of individuals to control or influence what information related to them may be collected and stored and by whom and to whom that information may be disclosed (from [9]).
Privacy is usually required by national laws and services that fall under the incidence of these laws must comply. There are no technologies for dealing with privacy. In the context of Web services, it is rather an administrative issue to make sure that the applications are compliant with the privacy laws. Privacy can be breached if the other security functions are breached (i.e. authorization is bypassed and personal information is accidentally released).

Availability

Definition **Availability** is the property of being accessible and usable upon demand by an authorized entity [9].
One of the possible reasons for a Web service not being available is a successful Denial of Service (DoS) attack. Such attacks can be detected and solutions for this can be used for Web services. Other reasons can include failure of software and hardware components or network connectivity. To prevent this from happening, good planning and redundancy are important.

2.4.3 Annex Security Functions

The following functions are not primarily concerned with the security of the system. However, because they rely on other security functions (i.e. charging requires the service to know the identity of the requester), they will be presented here and be regarded as security functions in the rest of this dissertation.

Accounting and Metering

Definition **Accounting** is the process of identifying, measuring and communicating economic information to permit informed judgments and decisions by users of the information [46].
Accounting is used in combination with charging. Other reasons for deploying accounting in a security infrastructure are: generation of statistics, capacity planing (this relates to availability), etc. Depending on the reason for deploying accounting, additional information might be collected - for example identity information if the requester is to be charged.
Accounting is very much dependent on the service offered (for example, grid services are accounted differently than Geo Web services). Accounting for geospatial Web services is discussed in [76].

Charging

Definition According to [1], **charging** refers to the process of fixing or asking as fee or payment.
Charging requires identification of the requester and accounting for the operation to be executed and / or the requested data. As also the case for accounting, charging is dependent on the service offered and on the infrastructure. Charging for geospatial Web services is discussed in [152].

Chapter 3

Security in a Web Services World

At the start of this chapter several properties used in the evaluation of software architectures are introduced. After this, three different approaches to implementing security for Web services are presented: *embedded in the application*, *embedded in the middleware* and *external*. The three approaches are analyzed using the previously introduced properties and I conclude that the approach of externalizing the security system has clear advantages. Several existing approaches in this direction are presented. Finally, the problem addressed in this dissertation is framed.

3.1 Architectural Evaluation of Security Systems

In this section I will present a couple of relevant properties of software architectures that can be used in the evaluation of architectural designs of security systems for Web services. The analysis is based on the work of Fielding [73] who presents criteria for the evaluation of network-based application architectures.

For such an evaluation, one needs to differentiate between the *functional* requirements and the *non-functional* ones. The properties described in this section are related to non-functional requirements.

As far as the functional requirements are concerned, the most important one for a security system is to correctly implement the security functions that are needed by the application which it protects. These requirements have been described in 2.4. Because, from a functional perspective, the security system is bound to the application (one influences the other), additional functional requirements are derived from the unique aspects of the application. These are specific to each application and it makes no sense to analyze them here.

Furthermore, it is important to understand that not all criteria have the same importance. Application design is a matter of trade-offs especially when it comes to security. Therefore it is hard to provide a normalized metric for comparing designs of security systems (especially since requirements are often dictated by the application being protected). However, approaches exist: [73] describes a model where numeric weights are associated with each property to indicate its relative importance and relevance to the a particular design.

Performance

Performance can only be accurately measured once a design is implemented (I will show accurate measurements of the ***SOS!e*** framework in chapter 7). However, even without having an implementation, it is possible to evaluate parameters based on the number and the type (i.e. API vs network-based) of inter-component communications and thus compare different designs without necessarily implementing these designs.

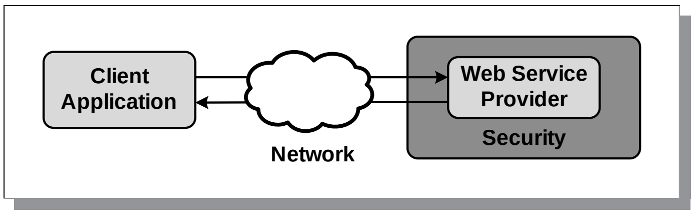

Figure 3.1: System deployment when evaluating performance

When evaluating performance parameters, it is important to have in mind the deployment of the system. Figure 3.1 shows this. I'm assuming that a client application is issuing requests to the service provider. These requests and (possibly also the responses) will go through the security system. Because there are several implementation possibilities for the security system, the two are not clearly separated.
According to [73], performance parameters can be divided into two distinct categories: network performance and user-perceived performance.

Network Performance This is a measure of the communication between the client and the Web service. The important parameters are:

Round-trip-time Is the time between the moment the request was sent until the response is received. If we consider the security system as a component separate from the service provider, we find that $rt = t_{network} + t_{sec} + t_{ws}$, where rt is the round-trip-time, $t_{network}$ is the time the information travels over the network, t_{sec} is the time consumed by the security system and t_{ws} is the time required by the Web service to process the request.

Throughput Represents the rate at which information is transfered between the client and the service provider. In an ideal system with several components connected in serial, the total throughput would be the minimum value of the throughput of all system's components[1]. Thus, $tp_{total} = min(tp_{ws}, tp_{sec})$, where tp_{total} is the total throughput, and tp_{ws} and $tp_{security}$ are the throughputs of the Web service and the security system respectively. However, in practice, because components influences each other in multiple ways, the total throughput is smaller than this.

Bandwidth Is the maximum available throughput over a given link (from one client application to one Web service).

From the formulas above, it is clear that the security system plays an important role in the overall network performance. Care should be taken so that it does not become a performance bottleneck.

[1]We assume that the components are not working in pipeline.

User-Perceived Performance Represents the impact that the network performance has on the user[2]. The important parameters are:

Latency Is the period between initial stimulus and the indication of a response;

Completion Is the amount of time required to finish an application action.

Both parameters are very much dependent on how the client application is built and on the Web service interface. Sometimes it is possible to obtain acceptable latency in spite of poor network performance. Streaming responses and background processing are some solutions in this direction.

Scalability

Scalability is the ability of a system to handle a growing amount of work in a graceful manner or to be readily enlarged [48]. In order to be scalable both the Web service as well as the security system need to be scalable. The most determining factor for scalability is whether the load can be distributed among different components. This counts for both the security service as well as the Web service.

Simplicity

Simplicity is the property of being simple. The following three properties contribute to simplicity:

Complexity Normally regarded as the opposite of simplicity, complexity is a measure of the number of inter-related parts of a system.

Understandability Is a measure of how easy a person can understand a system.

Verifiability A system is verifiable if it is possible to prove or disprove its correctness with respect to a certain method.

In literature [137, 117] it is often remarked that simplicity is a key requirement for security systems. Such systems need to be understandable (so that the operators can use it without making mistakes) and verifiable (in most cases this is a requirement from the start). The two properties go hand in hand with complexity (the less complex a system is, the easier is to understand it and to rigorously verify it).
As also recognized in [73], the main approach to inducing simplicity into systems is the principle of separation of concerns. Applying this principle means separating the overall functionality and allocating it to different components. This leads to a system design with several less-complex components which are easier to understand and test.
However, it should be noticed that interactions between components also introduce some degree of complexity. Therefore, the more components a system has, the more complex the system. In conclusion, it is a good practice to split the functionality into different components, but enough functionality should be allocated to each component so that a compromise is achieved between the simplicity of each component and the simplicity of the whole system.

[2]User-perceived performance only makes sense when the client of the Web service is operated by a human person. In a B2B scenario, only network performance is important.

Changeability

Changeability represents the capability of a system to be changed without a negative impact. The modifications can be either done when the system is not running (offline), or dynamically at runtime (the later is obviously harder to realize).
The most important aspects of changeability are the following:

Configurability Refers to the ability of a security system to support post-deployment changes in behavior through configuration changes. Changing the authentication method or integration with other existing user or policy repositories are examples for this.

Extensibility Sometimes called forward-compatibility, refers to the ability to add functionality to an existing system without having to make major changes to the system infrastructure. Extensibility is very much an architectural and design issue: if a system is not designed with extensibility in mind, it is hard to achieve this in the implementation phase. Best practices for designing extensible systems are component-based design where the coupling between components is minimized [73].

Reusability Refers to the ability to use components of a software system into other applications, without modifying them.

The fact that Web services are open systems designed to bind different applications that were developed using various technologies and which normally reside in different administrative domains make changeability an extremely important aspect.

Portability

Portability is the quality of a system of being able to run on various platforms. Web services were designed to be platform independent, in the sense that the service and the client can run on different platforms. However, this does not make the Web services portable, since they often rely on the middleware layer to perform various tasks (among them security functions). In such cases it is hard to change the middleware system, even if the application Web service was developed using platform-independent technologies such as Java.
Furthermore, security systems and security relevant information (policies, repositories) usually date before the Web services era. It is often challenging to integrate Web services with legacy security systems when the two reside on different platforms.

Reliability

According to IEEE, reliability is the ability of a system or component to perform its required functions under stated conditions for a specified period of time [80]. In the context of distributed systems, this is often seen as the degree to which a system is susceptible to failure at the system level due to partial failures of its components.
Because the security system is coupled to the Web service (one can not run without the other), reliability renders as an important aspect of both. General design techniques for enhancing reliability of systems are redundancy, monitoring and avoiding single points of failure.

3.2 Security Implementation for Web Services

As seen in section 2.4, security encompasses several functions. When it comes to implementing these functions in a Web service, several approaches are possible. In the following I'm presenting three different architectural choices (see figure 3.2) according to where the security system is located in relation to the application Web service and the middleware[3]. Each of the approaches is then evaluated.

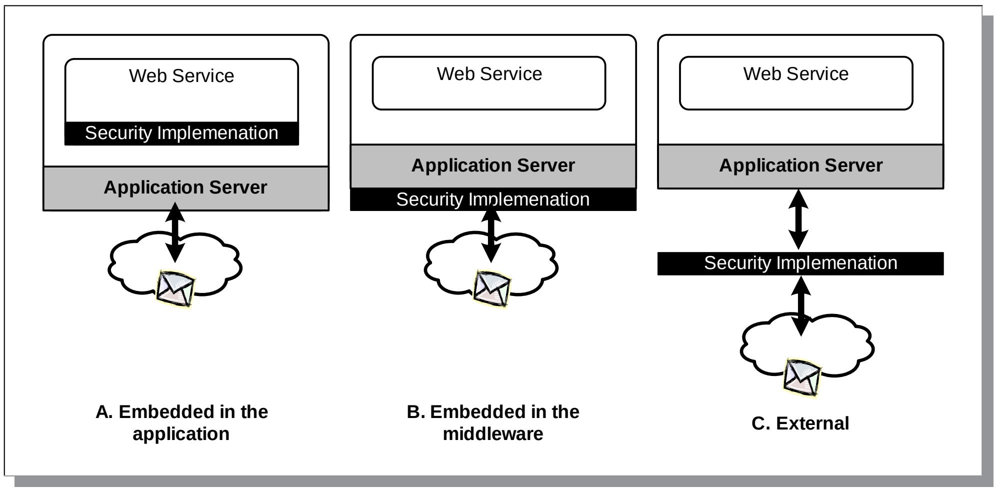

Figure 3.2: Security implementation approaches

Evaluation Methodology

The evaluation of the approaches is done according to the properties defined in the previous section. As no exact metrics are defined for the properties (for most of them it is also impossible to define such metrics), the evaluation is done in terms of positive / negative influences of the approach on the properties. A plus (+) indicates positive influences, a minus (-) indicates negative influences, while a plus-minus (±) indicates both positive and negative influences, depending on the situation. Several signs should be considered as an accumulation.

Embedded in the Application

In this case the security functionality is coded in the application itself. The programmer writing the Web service is responsible for writing the code for the security logic. For this task he will probably chose to implement some of the functionality himself, while reusing some code in the from of 3rd party libraries to implement other security aspects.
The security implementation is coupled with the application (and the 3rd party libraries) by means of APIs. Because the security logic is realized in the application itself, there is no requirement that these APIs be standardized. However, standardized APIs also exist, as for example the Java Authentication and Authorization Service (JAAS) [110].
This implementation choice is analyzed in table 3.1; the findings are backed by [52].

[3][41] makes a similar categorization. However, only approaches A and C are considered there.

Property	Analysis	Evaluation
Performance	Because most of the communication between components is done via APIs, network performance is very good in this case. Furthermore, because the application does not have functional requirements from the middleware, a fast light-weight middleware can be deployed (instead of a heavy-weight application server). This would further increase the performance by minimizing the time a message spends in the middleware.	+ +
Scalability	Because the system is monolithic, load-balancing can not be implemented. This results in a less scalable solution.	-
Simplicity	The monolithic aspect of the system affects both its understandability and its verifiability. These can be increased through the use of standardized APIs (such as JAAS) and modularization of the code. A good design should apply the principle of separation of concerns and split the security system into simpler modules.	- -
Changeability	In such a monolithic solution, changing the security system usually requires recompilation of the code. Other solutions exist (such as plug-ins), however this approach does not encourage neither reusability nor extensibility.	- -
Portability	The system is platform-dependent: it is tightly coupled to the middleware and the 3rd party libraries.	-
Reliability	A single component also means a single point of failure.	-

Table 3.1: Security embedded in the application - analysis

Embedded in the Middleware

In this case the security functionality is provided by the middleware where the Web service is executing. This is the case for most application servers such as the Systinet Server for Java [4] or Apache AXIS[5], where the server provides a rich set of security features in the form of interceptors (or handlers) that can be invoked before and after the execution of the Web service. Implementing a security policy for the application in this case is a deployment process, where the application administrator configures the interceptors and their execution order.

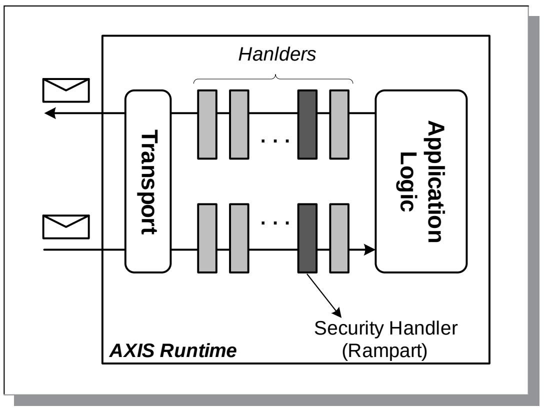

Figure 3.3: The concept of handlers in application servers

Figure 3.3 shows the principle of handlers as implemented in AXIS2 [77]. The Rampart module implements WS-Security, providing several security functions such as authentication, encryption, digital signing. Other functions can be implemented as separate user-defined handlers.
An noticeable improvement from the previous approach is that here, the security implementation

[4]http://www.systinet.com/products/ssj/overview
[5]http://ws.apache.org/axis, http://ws.apache.org/axis2

is separated from the application logic.
This implementation choice is analyzed in table 3.2.

Property	**Analysis**	**Evaluation**
Performance	Little network communication is involved here (both security system and application run on the same machine). Because security is implemented in the middleware, light-weight solutions are excluded and therefore some latency can be expected in the middleware. However, overall, performance is good.	+
Scalability	This solution scales better than the previous one because application servers usually include load-balancing features. However, load-balancing is achieved by replicating the whole service (together with the security system).	±
Simplicity	Separating the security system from the application is a step toward separation of concerns. Such a system is easier to understand because common APIs are used (the ones provided by the application server). Modularization can be achieved by means of handlers, although, normally there is no separation of concerns applied to the security system (for example in AXIS2 there is only one module implementing authentication, authorization, encryption and digital signatures).	±
Changeability	Such a solution usually (Systinet, AXIS2) relies on configuration. Extensibility is not hindered (handlers can be added / modified / removed) but it is also not encouraged (extending the existing handlers is not trivial). Reconfiguration and extension can only be done offline (requires the restart of the application server). Handlers can be reused in a similar fashion to libraries.	+
Portability	The application itself is portable. Because the application does not implement security it can easily be moved on another middleware. In this case the new middleware will be configured to secure the freshly moved application. The security system is not portable. It is tightly coupled to the middleware: any customization done to the security system will not work on another middleware system without modification.	±
Reliability	Application servers usually provide replication functionality. As previously remarked, the system can be replicated as a whole; it is not possible to replicate it in parts.	±

Table 3.2: Security embedded in the middleware - analysis

External

In this case security is implemented outside the middleware. The Web service is loosely-connected to the security implementation through a messaging interface. From this perspective, the security implementation can also be regarded as an application implemented on top of a middleware system. Several approaches in this direction are presented and compared in the next section. An analysis for this approach is showed in table 3.3.

Mixed

It is also possible to have mixed approaches where some security functions are embedded in the application, other are provided by the middleware, while some more are externalized. One such example is an application server which implements WS-Security and JAAS by means of handlers. Such a handler authenticates messages and creates a Principal object which is passed to the virtual machine executing the application code. The application is then implementing authorization

Property	Analysis	Evaluation
Performance	The application communicates with the security system via network (which is a lot slower than API communication) and therefore the performance is worse than in the other approaches. However, lightweight middleware systems can be used for both application and security system.	-
Scalability	The application and the security system can be scaled separately (depending on the requirements). The security system can be further modularized, and each of its components can be replicated as needed so that the load is distributed on those components that perform resource-intensive tasks. For example, as also remarked in [116], cryptographic functions can be separated from the rest, replicated and distributed on powerful machines.	+ +
Simplicity	Because there are clearly specified interfaces between the security system and the application, the system is easier to understand. To some degree, this follows the Aspect Oriented Programming (AOP) paradigm [101], as one aspect of the system's behavior - the security aspect - is separated from the rest of the system, both conceptually, but most importantly also in the implementation. Bringing the idea even further, the security system can be modularized in the same fashion (components being realized as different Web services). Such a division facilitates rigorous testing. Complexity is increased to some degree due to the increased number of inter-related components, but the gain in understandability and verifiability more than make up for it.	+ +
Changeability	Both extensibility and reusability gain from the fact that the application is clearly separated from the security system, communicating via a standardized message interface. In this approach, dynamic changes are also possible via redirects (as described in [121]).	+ +
Portability	Both the application as well as the security system are platform independent.	+
Reliability	Because components communicate via network (which is an unreliable medium) the overall probability that the system will fail is increased. However, both application as well as the security system can be easily replicated in order to introduce redundancy.	-

Table 3.3: Externalized security - analysis

checks using the standard JAAS APIs. Further services, such as logging, can be externalized and implemented as SOAP intermediaries.

3.2.1 Analysis Overview

Table 3.4 summarizes the evaluation of the three approaches to implementing security systems. From this we can clearly remark that the first approach in spite of good network performance lacks a lot of the properties that would make it suitable for the context of Web services. This is also one of the reasons why it is rarely deployed. Such a solution appears feasible for situation where performance is the number one requirement.
The second approach seems to compensate the pros and the cons for all properties. It is also widely deployed in the world of Web services. Such a solution is easy to deploy (usually only requiring the configuration of out-of-the-box products) and fits most of the needs. However, relying on the middleware to provide security has its disadvantages, for all with respect to scalability, extensibility, reusability and portability.

The third solution relies more on network communication, and therefore a lower performance is expected. However, it is very scalable, understandable, verifiable, easily extensible, while at the same time facilitating re-usage. In this dissertation I will further investigate this approach.

Solution	Performance	Scalability	Simplicity	Changeability	Portability	Reliability
Embedded in the application	+ +	-	- -	- -	-	-
Embedded in the middleware	+	±	±	+	±	±
External	-	+ +	+ +	+ +	+	±

Table 3.4: Comparison of the three approaches

3.3 External Security Approaches

This section introduces a number of approaches present in existing products and standards, where different aspects of security are externalized from the protected Web service. At the end of this section the presented approaches are analyzed to determine the advantages and the shortcomings for each of them.

3.3.1 Perimeter Security

A popular security pattern is enforcing the security at the network border. In this way, before a message enters the private network (where the sensitive resources are located) tasks such as authentication and message validation are performed. Only valid messages from known parties are allowed in the private network.
There are various patterns for this, which include the popular Demilitarized Zone (DMZ). In the following, I will show two patterns that are relevant to securing Web services at the application layer.

XML Firewall

XML Firewalls (sometimes also called XML Security Gateways) are devices that understand the Web services traffic (they work at layer 7 of the OSI stack). Such devices usually support SOAP security standards such as WS-Security, SAML, XACML and are able to satisfy a number of security functions such as validate XML messages, perform authentication, encryption / decryption, digital signatures, etc. The performance is usually one or two orders of magnitude higher than that of traditional application software because of specialized hardware.
The deployment architecture (see figure 3.4) is similar to that of a network firewall. A message originating from an application located in the external network must pass through the XML firewall in order to reach the service to whom it is addressed.

Perimeter Service Router

A Perimeter Service Router (PSR) is a Web service which acts as an intermediary between the applications located in the external network and the protected services located in the private network [94]. As far as security features, the functionality is similar to that offered by an XML Firewall.

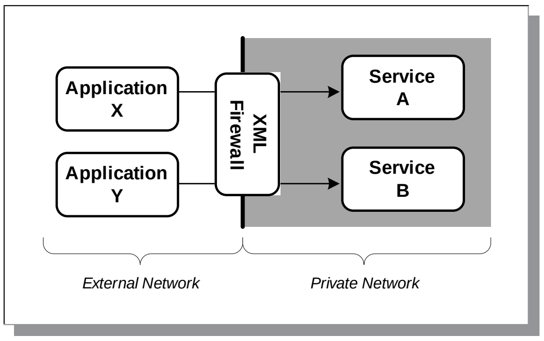

Figure 3.4: XML firewall

The difference between a PSR and an XML Firewall is that an XML Firewall is transparent for the applications (it does not have an endpoint address), while the PSR has an endpoint address for each of the protected services.

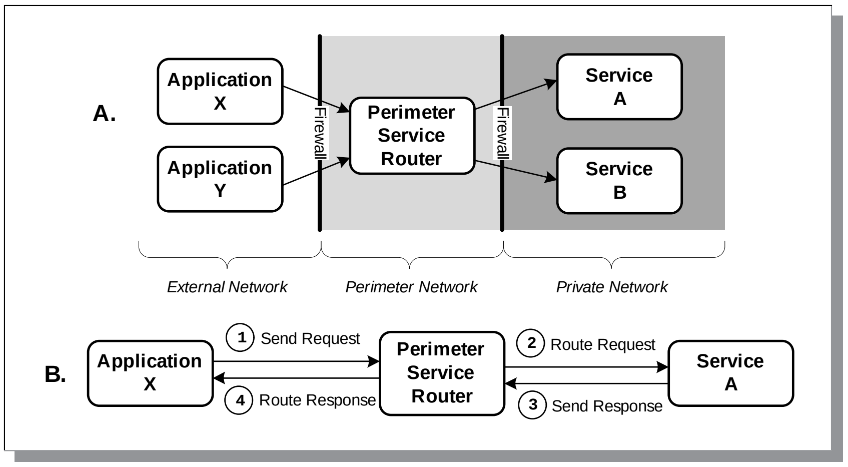

Figure 3.5: Perimeter service router

A PSR is usually deployed in a Demilitarized Zone (figure 3.5, A). A message originating from an application will be addressed to the PSR. After performing all inbound security tasks, the request message will be routed to a service inside the private network (figure 3.5, B); the response message will be routed through the PSR back to the client.

Products

There are various implementations of the two patterns, both in hardware and software products. Companies offering such products include DataPower (IBM), Vordel and Xtradyne. To provide the reader with an overview of the capabilities of such a product, I present the main characteristics of the WebSphere DataPower XML Security Gateway XS40 from IBM [96]: XML well-formedness checks, XML schema validation, XML filtering, XML denial-of-service protection, encryption, digital signatures, WS-Security authentication, fine grained authorization using XACML, content-based routing.

3.3.2 Enforcement Points

The XACML specification proposes a service model (figure 3.6) where resources are protected by means of Policy Enforcement Points (PEPs). According to the XACML standard ([18]), the PEPs should be placed at a choke point which can not be bypassed. This one is usually not located at the perimeter of the network. Instead it is situated as close as possible to the resource being protected (this is totally opposed to the perimeter security philosophy)

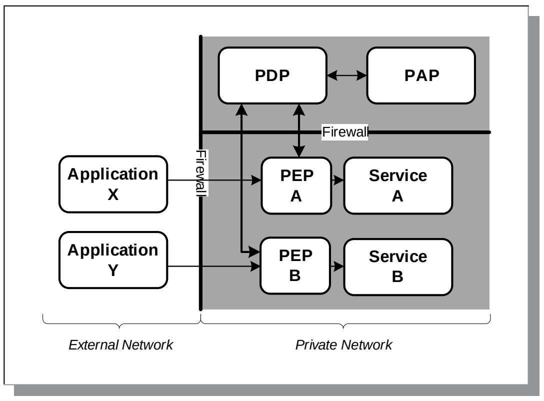

Figure 3.6: XACML model: enforcement points

Requests targeting protected services, first reach the PEP and only after this one has received a positive response from the decision point (PDP) and has discharged any associated obligations is the request forwarded to the service.

3.3.3 Security Services as SOAP Intermediaries

In Part 1 - Messaging Framework, the SOAP protocol specification describes the role of SOAP intermediaries (these have been introduced in section 2.1). One type of intermediaries are *active intermediaries* which undertake additional processing and can modify the outbound SOAP message in ways not described in the inbound SOAP message [88]. The specification further states that the potential set of services provided by an active SOAP intermediary includes security services and content manipulation services.
Examples of active intermediaries providing security services are also given: an encryption service which encrypts parts of the SOAP header and / or parts of the SOAP body. Furthermore, part 0 of the same specification provides an example of a logging service which is also realized as a SOAP intermediary [111]. In this case, the service is a forwarding intermediary, which only processes the messages it receives and relays them further.

3.3.4 Mixed Approaches

Oracle WS Manager

A mixed approach of the two previously presented patterns is implemented in the Web Services Manager product from Oracle (described in [54]). The relevant parts of such an infrastructure deployment are presented in figure 3.7.
A request targeting a service in the private network is first intercepted by a gateway component. This one is placed in a DMZ and has the role of hiding the internal organization of the network (similar to a PSR). After having passed through the gateway, messages then hit a further enforce-

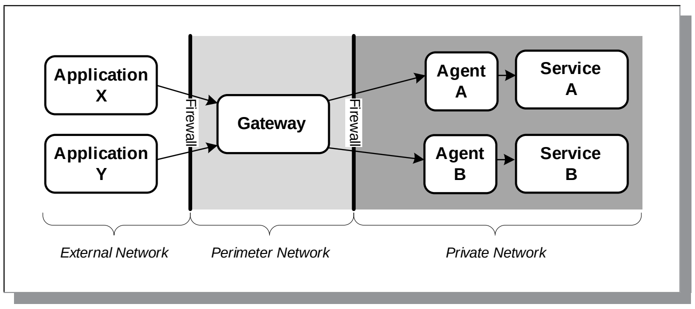

Figure 3.7: Oracle WS Manager architecture

ment point called Agent which is located in the vicinity of the service. Each service is protected by an Agent which is installed in the same process space with the service it protects [54].
Both Gateway and Agents perform similar tasks (enforce access control, make audit traces, implement cryptographic functions, etc.). However, they serve different purposes: while the gateway intercepts messages addressed to all services, it makes sense to have it enforce coarser grained authorization polices and perform other security tasks that are relevant for all services. On the other hand, because agents are located near the service, they can perform service specific authorization enforcement or encryption (in order to also protect messages while travelling within the private network).

ESB Products

Because of their distributed nature, ESB implementations are ideal candidates for externalizing security. As there are so many ESB implementations, I chose to analyze the security features of two popular ones, which I consider to be representative: CapeClear ESB 6.7 [57] and Mule 1.4[6] [2]. Both Mule and Cape Clear provide a rich set of security features. Most of the basic security functions (such as authentication, authorization, cryptographic functions, etc.) are implemented as interceptors which are executed before and / or after the protected Web service is executed. From this point of view, the implementation resembles that of a traditional middleware systems such as Apache AXIS (as previously analyzed in 3.2).

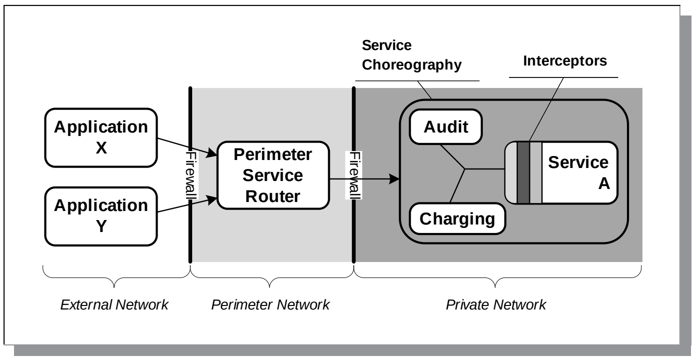

Figure 3.8: Security in existing ESB implementations

However, ESB systems do provide opportunities to integrate services by means of message routing. Both implementations provide simple logging services which can be used for audit purposes. In this

[6]The reasons for choosing the two are: Mule is one of the most popular open source ESB implementation, Cape Clear is a commercial ESB with a comprehensive documentation describing the security features.

way, messages travelling between two endpoints can be deviated to an audit service. Cape Clear ESB further leverages message routing to implement a service that follows the perimeter service router pattern (see 3.3.1).
Additionally, the rich set of tools for service orchestration and choreography provided by ESBs can be leveraged to integrate a protected Web service with other Web services implementing annex security functions, such as for example charging. Figure 3.8 illustrates this.

3.3.5 Conclusions

In the this section I showed a couple of approaches to externalizing security from Web services which are currently implemented into products. The following is a brief analysis of these approaches, with regard to the architectural properties described in section 3.1.

Perimeter Security Solutions These solutions concentrate all security functions at the network border. The performance of such solutions is usually good because of optimizations in software and hardware. Such solutions can also scale - by use of classical load-balancing techniques - and have a good degree of understandability and verifiability. However, due to the monolithic nature of such solutions they are harder to extend. Furthermore, such a solution is not always suitable, because the internal network is considered 100% trusted (there is no security applied within the internal network perimeter). Additionally, annex security functions (such as accounting, charging) are not addressed by these solutions.

Enforcement Points These enforce security as close as possible to the protected resource. However, in the way they are defined in XACML, they only address one aspect of security, namely authorization.

Oracle WS Manager This product conveniently combines the advantages of perimeter security solutions and enforcement points. However, this approach also has a monolithic nature, as both gateway and agents combine several functional aspects of security.

ESB Implementations ESBs provide support for service choreography and orchestration which can be leveraged for security purposes - one good use for this is the integration of annex security functions such as charging. However, as remarked in 3.3.4, most of the basic security functions are implemented as interceptors (handlers) provided by the middleware.
It's worth remarking that the use of interceptors increases both simplicity (through separation of concerns) and reusability (the same interceptor can be attached to several services). However, because interceptors are specific to each middleware system they can not be reused with a different middleware. This would only be possible if the desired functionality would be exposed as a reusable service.

3.4 Problem Statement

Based on the analysis from section 3.2, we can clearly see that the approach of externalizing the security system has obvious advantages. In section 3.3, I showed that some approaches in this direction already exist and have been implemented into products over the past four years. However, all of the presented approaches have some shortcomings, as discussed in section 3.3.5.
In this dissertation, I am further developing the idea of external security and proposing a model

where security is realized as small modular services. In order to have a simple, understandable and verifiable design, the principle of separation of concerns is applied. By following this principle, the security system is divided into services according to functionality, and a taxonomy of possible security services is developed. These services can be regarded as infrastructure services, as they can be shared by applications belonging to the same network. This makes the design highly reusable. Additionally, through the use of standardized messaging interfaces, overall system portability is ensured.

Enterprise Application Integration (EAI) techniques are used to glue the security services together with the application Web services. Because of flexibility, the Enterprise Services Bus pattern is used. ESBs support both service orchestration and service choreography and implementations usually come equipped with simple-to-use orchestration editors and runtime environments which can easily be used to architect a security solution from security services. Perhaps one of the most important features of an ESB is message routing and the mediation pattern, which allows functionality to be built in the system in a totally transparent fashion. Leveraging this makes the proposed security system easily configurable, extensible and scalable. Furthermore, because of message routing, changes to the system can also be performed online, at runtime, by using the same technique used when upgrading web servers: traffic is temporarily redirected to a redundant host (in terms of ESB, this translates to temporarily route change).

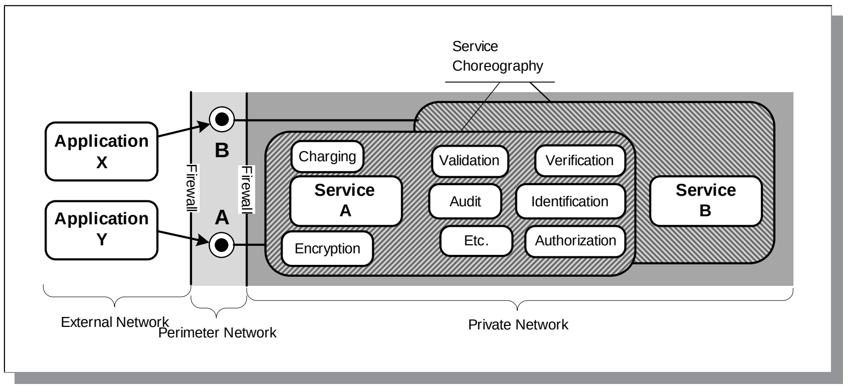

Figure 3.9: Problem statement: The Service Oriented Security Architecture, ***SOSA***

Figure 3.9 sketches the concepts by showing two services - A and B, having different policies that are realized through different service choreographies. A number of security services are shared between the two (Validation, Identification, Authorization, etc). Service A additionally incorporates a Charging service (we're assuming a scenario where this service requires payment).

3.4.1 SOSA , SOS!e and the Rest of the Dissertation

Because this model is applied to services and Service Oriented Architectures and because the core idea is to think of security in terms of reusable services, I named this model the **S**ervice **O**riented **S**ecurity **A**rchitecture or ***SOSA*** .

The main ideas behind this model together with some clarifications regarding design decisions are introduced in chapter 4. A detailed description by means of the ISO RM-ODP is presented in chapter 5. In the Technical Viewpoint, section 5.6, a prototype implementation of ***SOSA*** is described: the ***SOS!e*** framework - **S**ervice **O**riented **S**ecurity, an **I**mplementation **E**xperiment.

In chapter 6, I'm showing how ***SOSA*** has been applied in several projects in the context of geospatial information, and presenting different security services that have been developed for the ***SOS!e*** framework. The intention is to demonstrate the applicability of the concept. In chapter 7, an evaluation of the proposed architecture is performed.

3.4.2 Assumptions

The idea of exposing security functions as services in the network might seem like a bold one. This section tries to clarify questions that might come into the readers mind. Furthermore, because I'm addressing a number of different issues, a frame for this dissertation is defined by describing which issues are explicitly not addressed here.

Network Security

SOSA suggests that security be separated from the application service. By means of routing and orchestration scripts, a message addressed to the application service will be processed by the security services, before and after this message reaches the application service. Because the security services are external to the application service and both communicate through a network, a couple of question arise:

Is the Application Web Service Left Unprotected in the Network? Yes, the application Web service is *security unaware*. A proxy is exposed in the perimeter network (see figure 3.9). In this way, a service requester accessing the application from the external network is not able to see the Web service deployed in the private network. However, somebody with access to the private network can bypass the security services and access the application directly. A further similar security threat is eavesdropping from the private network.
These issues are not addressed in this dissertation, and are not further discussed. If they are considered a threat, lower layer security methods can be applied. Examples include conveniently partitioning the internal network by use of firewalls and switches or applying authentication and authorization at the transport layer. The result of that would be the equivalent of a trusted network which only binds the security services and the protected application services.

What If the Communication between the Services is Compromised? Messages travel within the private network between different security services. This introduces the risk of messages being stolen, deviated or altered while they are in transit over the network. These threats can be directed either at the messaging middleware (which can be compromised) or at the network (intercepting the message while over-the-wire). In this dissertation these issues are not addressed. It is considered that these issues are solvable through the use of lower layer security (as previously described).

Services Trust

A service request to the protected service translates to several messages being exchanged between different security services. Each service has to fulfill their assigned task - examples of tasks are: encrypt the message, identify the requester. The following questions arise:

How do I Know that a Service is Fulfilling its Task? After all security services have highly important tasks - if a service is tasked to identify the requester and it always returns "administrator", the service not only renders itself useless, but it sabotages the whole system.
In this dissertation the emphasis is on environments where the behavior of the services is *know and*

trusted by the operator of the system. Scenarios such as dynamically looking up a security service in a public UDDI directory and then invoking that service are not addressed.
One topic which is addressed here, is the case of *B2B integration* where the choreography can also include services hosted by a partner. The partner can be totally trusted or can only be partially trusted. In the latter case, digital signatures and encryption can be used to protect information while message correlation techniques can be used to ensure that messages are not dropped or altered.

How do I Know that the Right Services are Invoked? It is assumed that part of the work involved in exposing a protected service is designing a choreography for security services. This one is produced by a responsible system administrator. Furthermore, it is assumed that the execution environment (the messaging middleware and network) is trusted.

Chapter 4

Service Oriented Security Architecture

This chapter constitutes a preamble to the next one where ***SOSA*** is described in detail. Here, the main ideas behind the proposed architecture are presented, and some clarifications regarding design decisions are made. These clarifications refer to the types of interactions the security services may implement, the way security services are aggregated together (invocation vs. mediation) and the communication between security services. At the end of the section, an illustrative example is presented.

4.1 Service Oriented Security

In the context of SOA, a service is a repeatable task which is published in order for applications to leverage it in application-specific ways. Services typically deal with specific tasks / solve specific problems, and applications are created by combining several services together in order to provide value added functionality. Modern design encourages the separation of business functionality from the infrastructure. In the context of security infrastructure, this separation of concerns allows the creation of applications which are security *unaware* [92] - the premise is that any security data or security process is provided outside the application itself. In section 3.2 it was showed that externalizing security from the service implementation has clear advantages, at least from a design point of view. One of the most important advantages is the increased understandability of the system, due to the separation of the security aspect from the system's logic, which is reflected in both implementation and deployment. Other advantages are increased scalability, reusability of components and simpler, more verifiable components.
In this dissertation the idea of external security is pushed one step further: not only is the security functionality separated from the application, but the security implementation itself is split into a number of security services. The reason for this is given by one of the main principles of SOA, which states that complex business functions should be decomposed into those elements that are repeatable, and these elements be exposed as services [92]. If we consider security separated from the application and we have in mind that a lot of the security tasks are repeatable (most applications need authentication, encryption, etc.) applying this principle to the security implementation is only natural.
The result is a design where instead of having the security aspects bundled together with the

business application, these are implemented as a series of security services. If building a SOA business application requires the combination of several business services, building a security solution according to the ***SOSA*** principles requires the combination of several security services. Furthermore, if the security services are realized in the same manner as the business services, the same technologies and patterns can be used to combine the two.

Potential Security Services

As in the case of designing a SOA application, the first step is to gather and analyze the requirements, identify the repeatable elements and then take into consideration the granularity trade-offs. The security requirements for Web services have been discussed in section 2.4. These are authentication, authorization, confidentiality, integrity, non-repudiation, privacy, availability, accounting and charging. All of these tasks are repeatable elements: each of these tasks can be applied to all messages addressed to a business service; several of these tasks may be executed in the same way for several business services. Therefore, services may be realized that address any of the above listed requirements; in section 5.3 the requirements are analyzed, and a list of possible security services is proposed.
However, when taking the decision which security aspects to be implemented as services, the granularity aspect should be taken into consideration. For example authorization may be realized as two distinct services (decision and enforcement, as logically divided in the ISO 10181 model [4]) or it may be realized as a single service implementing both tasks. These aspects are specific to the context of each application. For each context, an analysis needs to be done, and when making this analysis, again, the repeatable elements need to be identified.

4.2 Service Definition

If we follow the SOAP / WSDL / UDDI model, a service defines one or more operations. Each of these operations provides some functionality and defines an interaction pattern (i.e. request-response, one-way, etc). Other components can interact with the service by implementing the interaction patterns defined by its defined operations.
The goal here is to have security services defined in the same way the other services are defined. In the following we will investigate the possibilities of defining security services, namely what kind of interaction patterns should security services expose and how can security services be combined together.

4.2.1 Interaction Types

WSDL 1.1 [158] defines four types of interaction primitives for service operations:

One-Way The service accepts an input message but does not generate any output message. In this interaction pattern, a requester sends a request, but does not expect any answer.

Request-Response The service accepts an input message and generates an output message. To interact with the service, a requester sends a request and then, at a later point in time, receives a response.

Solicit-Response The service first sends an output message and then expects an input message. In this interaction, the service solicits information from the client (by sending it a request) and then expects an answer back from the client.

Notification The service sends only output messages to the client. In this interaction, the client receives notification messages from the service, without requesting information.

The first two interaction primitives assume that the service is invoked by a requester (by sending a request message), while the last two primitives assume that the service is acting by itself (needs not be invoked by the client).
As mentioned earlier, security services are invoked for each message addressed to (or coming from) the protected business service, and their task is to satisfy the security functions. Therefore, only the first two interaction patterns are appropriate. Of these two, considering the requirements, probably the most appropriate is the request-response pattern, because security services are normally expected to produce some result (i.e. encryption, authentication, etc.), or at least return a status (i.e. authorization, charging, etc). Nevertheless, it is possible to realize security services that implement the one-way interaction pattern (i.e. audit).

4.2.2 Aggregation Methods

From the point of view of how security services are composed together, the following two possibilities exist:

Invocation In this case one service explicitly invokes another one. The invocation takes the form of a message being sent from the invoker to the invoked service and, assuming a request-response pattern, an answer is sent back at a later point in time. A typical example for this type of service aggregation is remote procedure call (RPC). Figure 4.1A shows how several security services can be combined together in order to enforce security policies on messages addressed to a Business Service: an entity called Security System invokes security services S#1 to S#n and then combines the results.

Mediation This case assumes an Enterprise Services Bus (ESB) invocation framework. In such an environment, a request is routed by the ESB middleware from the service requester to the service provider through several intermediaries that apply transformations to the original request. From an execution point of view, the effect of this is that the output message of one service is the input message for the next one along the chain. Such message transformations are called mediations [56, 92] and are meant to provide a decoupling between the format of the service request and the format of the service invocation. However, mediations need not be limited to format conversions - value added services may be added along the chain, such as security services. Figure 4.1B shows how a request is processed by security services S#1 to S#n, before reaching the business service.

Both of these possibilities are valid ways for implementing security services, and this is confirmed by existing approaches ([111, 33, 40], section 3.3). However, choosing one over the other has implications on the service interface definition and the processing model. These will be presented and analyzed (having the context of security in mind) in the following.

Invocation

Message Contents In this case the service invocation is explicitly done by the entity that creates the request message and sends it (thus invoking the service). It is possible to define a service interface where the semantics of the input message are well specified and reflect the functionality

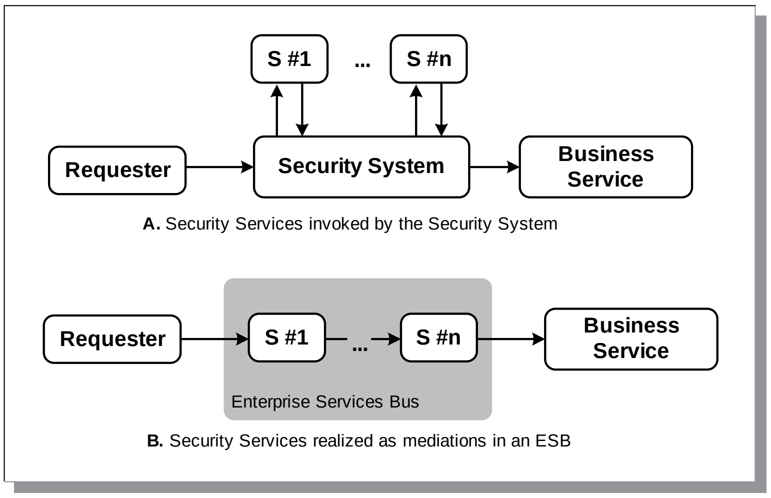

Figure 4.1: Invocation vs. mediation

of the service. The same thing goes also for the response message. The service interface definition is similar with the definition of a function call in a procedural programming language:

doService(input Arguments, output Argments)

Synchronous Interactions If we assume a request-reply pattern (as in the case of most security services), we can clearly see, that such an interaction is synchronous in nature. The entity invoking the service will have to wait until the response is received.

Execution Control / Context Because there is one entity that invokes one or more services, this one can maintain context information between invocations.

Security Services Because in this way it is possible to specify clear semantics for the input and output messages (for example using WSDL), there have been several standardization efforts to specify interfaces for services implementing security functionality. Some of the standardization efforts related to SOAP Web services are listed below:

XACML [18] Defines a service interface for the Policy Decision Point (PDP) and Policy Administration Point (PAP). The PDP can be queried for authorization decisions by a Policy Enforcement Point (PEP), while the PAP is normally invoked by the PDP in order for this one to retrieve authorization policies.

WS-Trust [40] Defines a service interface for a Security Token Service (STS). The interface defines among others an operation for verifying the validity of authentication security tokens.

WS-Federation [33] Defines service interfaces for Attribute Services and Pseudonym Services. These services act as Policy Information Points and can be queried for information related to the requesting entities.

Digital Signature Service [38] Defines a service interface for signing and verifying XML documents and other data.

Message Exchanges The number of message exchanges required to invoke N security services is $2N$ (for each invocation, one request and one response message are necessary); the topology for this is showed in figure 4.2A.

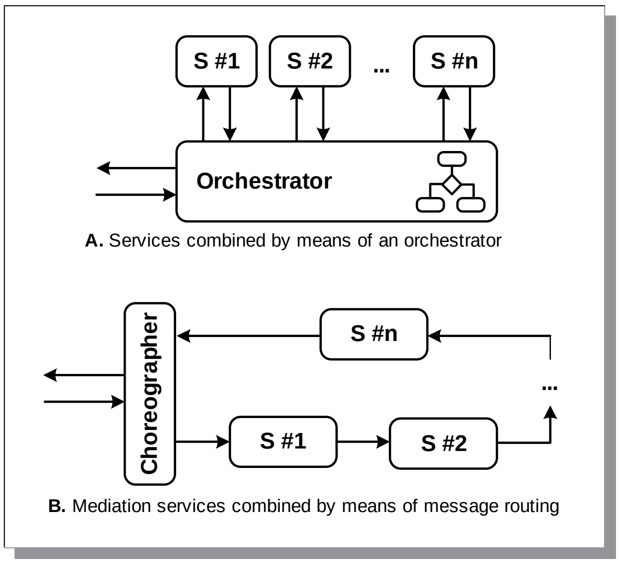
S #1
S #2
...
S #n
Orchestrator
A. Services combined by means of an orchestrator
Choreographer
S #n
...
S #1
S #2
B. Mediation services combined by means of message routing

Figure 4.2: Combining together N security services

Mediation

Message Contents In this case the service invocation is done implicitly: a message that travels from the original requester to the service provider is routed through intermediary mediation services. These services are transparent to both the original requester and the service provider (it is the middleware layer that invokes the mediation services).
As mentioned before, the processing of the message is similar to a pipeline: the output of one service is the input of the next one along the chain. It is up to the ESB middleware (and the administrator configuring the message routing rules), to route the message in such a way that each service receives a message that it is able to process.
Because most security tasks deal with the information of the original message, a convenient way of realizing security services, is to always include the original message in messages flowing between mediation services. Of course, this may be changed (i.e. filtered, encrypted, etc.) as required by the security policy. Furthermore, additional information (destined for the sole use of security services) may be appended to the message as needed. The intention of this information is to convey partial processing results which are then communicated from one service to the other.
According to the SOAP processing model [88], such services act as forwarding intermediaries, and they can be either active (modifying the payload) or passive (only modifying the header information). The definition of the service will look as follows:

doService(input Message, output Message)

The important difference from the previous case, is that the input message is similar to the output message. Both messages contain the original request, together with some information destined for the use of intermediary services.

Asynchronous Interactions As opposed to the previous case, here the communication is asynchronous by nature. The services process messages coming from the middleware and then give the back. The services follow a *send-and-forget* strategy, as also described in [56].

Execution Control / Context The service execution order is reflected in the message routing rules. Common strategies (introduced in section 2.3) are itinerary routing and content-based routing. Because both service execution as well as message routing are done in a distributed manner, there is no single entity that can store context information (as in the previous case). The only way to store context, is by including this information in the message.

Security Services Because in this case the security services act as intermediaries, and because the semantics for the service can not be clearly defined (as in the previous case) such service interfaces have not been standardized. However, there are several existing approaches in this direction:

SOAP 1.2 [111, 88] Describes intermediary SOAP nodes. The specification mentions two types of intermediaries: forwarding intermediaries (which only relay messages without modifying the payload) and active intermediaries (which also modify the payload). Examples in the context of security are also given: logging (forwarding intermediary), encryption (active intermediary).

XML Firewalls Introduced in section 3.3.1, also act as mediation services. They are able to perform several security tasks (authentication, authorization), modify messages (for example through encryption) and also append information to messages (SAML assertions confirming that the authentication and authorization processes have been fulfilled).

XACML [18] Defines the Policy Enforcement Point (PEP). Although this entity is not specifically defined as a standalone service in XACML, if the PEP is to be realized as a service, than it would act as a mediation service, dropping unauthorized messages and forwarding the authorized ones. An RPC-style definition for this service would make little sense.

ESB In this article [92], a concept where several security functions (authentication, authorization, data integrity, data confidentiality) are realized as mediation services and integrated by means of an ESB is presented.

Message Exchanges The number of message exchanges required to combine N mediation services is $N + 1$; the topology for this is showed in figure 4.2B. The number of exchanged messages is almost half as in the previous case. Because message parsing is usually time and memory consuming, the expected performance is better in this case.

Comparison Overview

In the previous subsections the differences between the two aggregation models have been presented. As seen, each of the two models has its advantages and its limitations. A summary of what was discussed is showed in table 4.1. The last row of the table shows the advantages of each approach according to [56]: combining services by means of a process orchestrator allows conditional splits and joins, while the combination of mediation services by means of message routing has the advantage of being highly distributed.
In this dissertation, the focus is on developing a flexible security framework which is based primarily on the mediation pattern. RPC-like services are not excluded as they can be invoked by mediation

	Invocation	**Mediation**
Messaging Type	Synchronous	Asynchronous
Service Composition	Orchestration	Choreography
Execution Control	Centralized	Distributed
Service Definition	doService(input Arguments, output Result)	doService(input Message, output Message)
Semantics of the Request and Reply Messages	Different	Similar
Communication between Services	Realized by the orchestrator, through invocation parameters / return values	Additional information appended to the message (Annotations)
Execution Context	Maintained by orchestrator	Stored in the message
Number of Message Exchanges (see figure 4.2)	$2N$	$N+1$
Advantages (according to [56])	Conditional splits and joins when aggregated through an orchestration engine	Highly distributed, metadata-driven

Table 4.1: Invocation vs. mediation

services, either directly or through orchestration services. Such examples are showed in figure 4.3: in 4.3A, service Y (a mediation service) invokes service M in order to accomplish its task (in a real-world example, a PEP service may call a PDP service requesting a decision); in 4.3B, there is an orchestration service which is realized as a mediation service, and this one invokes services M, N and P.

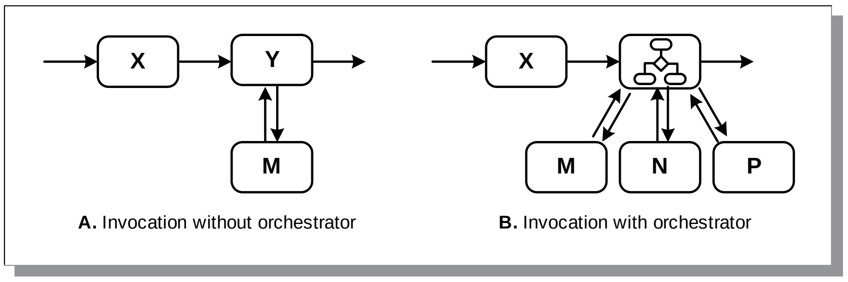
X
Y
M
A. Invocation without orchestrator
X
M
N
P
B. Invocation with orchestrator

Figure 4.3: Examples of mediation services invoking other services

Some reasons for realizing security services as mediation services are listed below:

1. Most security tasks deal with the information contained in the original message. Some tasks (such as for example security tokens verification), require the whole message (both payload / body and meta-data / header) - the reason for this is that in order to prevent replay attacks, most security tokens are bounded to the message.

2. The mediation model fits a lot better than the invocation one for some security services: logging, policy enforcement point, encryption, digital signature, accounting.

3. Because there are less messages being exchanged ($N+1$ as opposed to $2N$) the overall performance of the security system should be better.

4. Mediation services are adequate for asynchronous processing. Asynchronous messaging allows the deployment of the system in more scenarios (synchronous messaging can be realized on top of asynchronous messaging).

4.3 Communication between Security Services

Each security service will process incoming messages in order to accomplish its task. Some tasks may require several services to cooperatively process the message (for example authorization normally requires the identification of the requester). In such a case, intermediary processing results (in the previous example, the identity of the user) need to be exchanged between the services. The two distinct possibilities for this are showed in figure 4.4 and described below:

Shared Database The two services communicate by means of a *shared database* (this integration pattern is detailed in [95]): the first service stores the intermediary processing results in a database, while the second one queries the database for this information. This approach is illustrated in figure 4.4A.

Annotations The first service appends the intermediary processing results to the message before dispatching it to the next service. This approach corresponds to the *content enricher* message pattern described in [95]). In this dissertation this is called an annotated message, because the dispatched message consists of the processed input message, witch is annotated with the intermediary results. This approach illustrated in figure 4.4B.

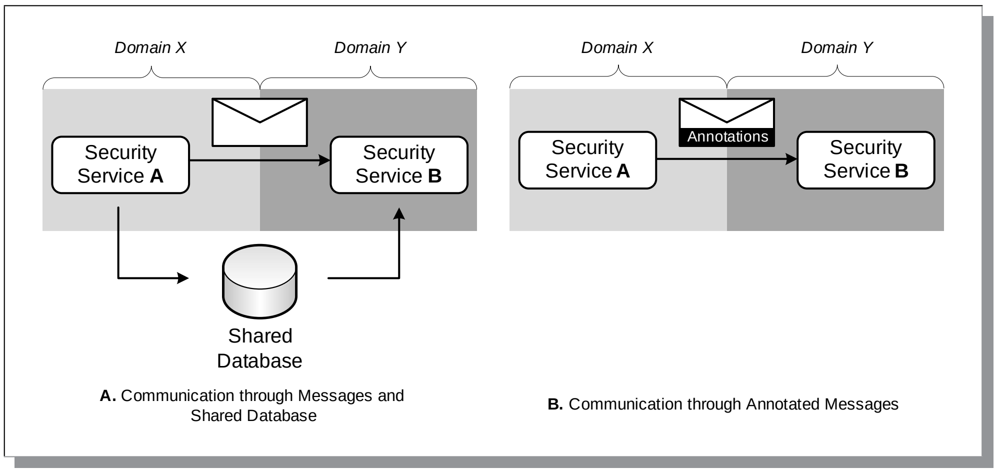

Figure 4.4: Communication between services

In order to deploy the first approach, both services need to have access to the shared database. This might be an issue if the two services are located in different administrative domains. Furthermore, an additional entity is introduced to the system - the shared database - which results in communication overhead and additional management.
The second approach, where the intermediary processing results are appended to the processed message does not have these shortcomings. Furthermore, since intermediary results refer to the processed message, it makes sense to have the two together.
For these reasons, in the rest of this dissertation, it is assumed that security services exchange information with one another by means of annotated messages.

4.4 Illustrative Example

In the previous sections I showed the main characteristics of security services: the interface of the services, service coupling approaches and ways of communication between services. In the next chapter, I will describe in detail the issues involved in designing and deploying a ***SOSA*** security system. Before that, I will first present an illustrative example. The purpose of this example is to show, for a concrete case, what the security requirements are, how these requirements can be addressed in implementation, how the security system can be divided into security services, what functionality is assigned to each service and how the services communicate with one another. Furthermore, for better understanding the model, an analogy between the security system proposed in this dissertation and a document-based workflow is showed at the end of the section.

4.4.1 Scenario

A Web Map Service[1] is serving maps to users. The system offers different types of subscriptions. Based on the subscription type, users have access to maps covering different areas, containing different types of information, and having different levels of detail. The system uses a flat-rate charging scheme, therefore there is no need to meter service usage for each user.

4.4.2 Message Routing Solution

From the scenario above, the following security requirements can be identified:

Authentication Because the system has different users with different permissions, it is necessary to authenticate the user's requests. Assuming that users send a security token containing their claimed identity together with their map request, it is necessary to verify that this information is valid. Furthermore, once the identity is known and has been verified, it is necessary to determine further identity attributes - in our case, the subscription type. We therefore conclude that the following two tasks need to be performed for each request:

1. Verify that the identity information is valid;
2. Determine further identity attributes (the subscription type).

Authorization Based on the subscription type, it is necessary to determine whether a given map request is permitted or not.

Audit In order to ensure that the security policies are respected, the system should also log significant information found in incoming map requests and the generated responses. This information may be used at a later point in time in order to determine the cause of system failures or security breaches.

In order to produce an understandable design, the security requirements are assigned to services according to the separation of concerns. As a result, we determine that the following three security services must be implemented:

Authentication Verifies the validity of identity tokens. Furthermore, based on the identity information, the subscription type is determined (the service connects to a user repository where this information is stored). Finally, just before dispatching the message, an annotation containing the subscription type is appended.

Authorization This service extracts the information regarding the subscription type from the message. Afterward, the message payload is inspected and it is determined whether the request it is permitted or not. If the request is not permitted an exception will be returned. Otherwise the message will be forwarded.

[1]For details about WMS see section 2.2

Log This service inspects the message (either request or response), extracts appropriate information for audit and persistently stores this information.

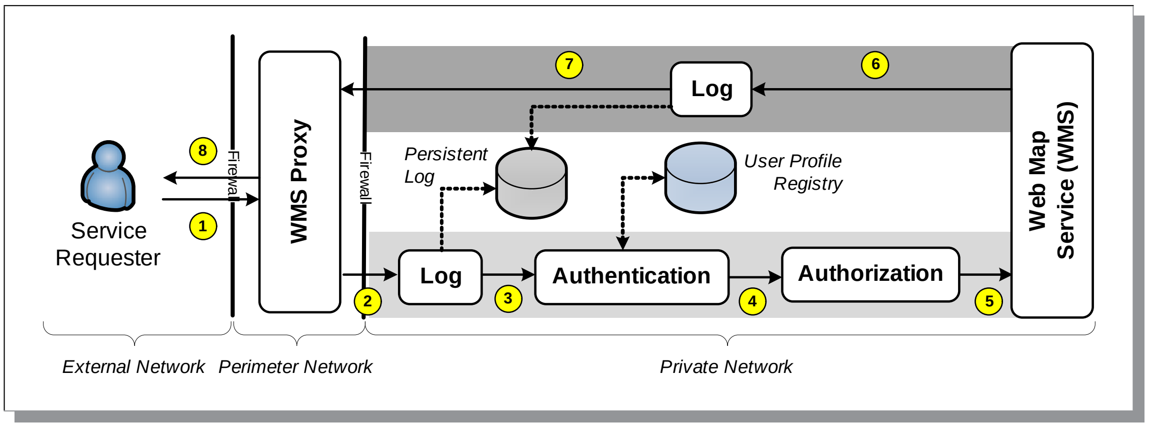

Figure 4.5: Security system using message routing

The deployment is showed in figure 4.5. Because the WMS service is *security unaware*, this one has been deployed in the private network. However, an endpoint needs to be accessible to the users which are located in the public network. For this reasons, a proxy service is deployed in the perimeter network - WMS Proxy. The task of this service is to route the messages according to the itinerary showed in the figure and to correlate the request and response messages (communication is, as showed, asynchronous). This is conform to the *Reverse Proxy Pattern*, described in [95, 92]. In the following, the communication between the entities depicted in figure 4.5 is, step by step, described:

1. The service requester creates a request message, containing a map request (as payload) and a security token asserting its identity. The message is sent to the WMS Proxy.
2. The WMS Proxy appends the itinerary to the message and sends it to the first service along the path - the logging service.
3. The logging service logs relevant information from the message. This service will log all messages, including the ones that will later be rejected by the authentication and authorization services. In this way, brute force attacks can later be detected.
4. The authentication service verifies the security token and annotates the message with the users subscription type. If the security token is not valid, an exception will be returned to the client.
5. The authorization service determines whether or not the request is allowed based on the subscription type.
6. The WMS processes the request and creates a response message containing the requested map.
7. The logging service logs relevant information from the response message.
8. The response is correlated with the request and is sent to the service requester.

4.4.3 Analogy to Real-Life Paper-Based Communication

The message flow and the processing model are to some degree similar to the communication involved in a document-based workflow. An illustrative example is presented in figure 4.6.

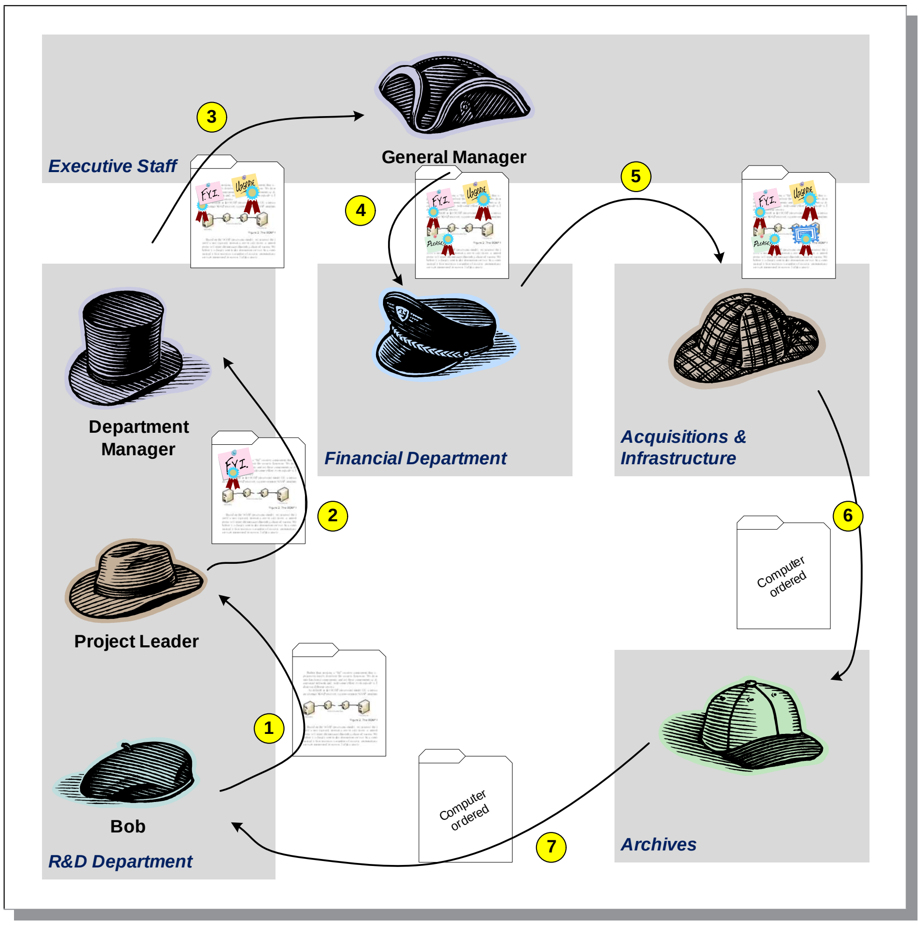

Figure 4.6: Analogy to document-based work flows

In the scenario we assume that Bob, a programmer employed in the R & D Department of a large corporation, needs a new computer in order to better perform his work. For this, he writes a request (perhaps motivating his demand) that he signs and sends it to his supervisor, the Project Leader (step 1). The Project Leader inspects the request, perhaps annotates the request with some remarks, approves it and then sends it to the Department Manager (step 2). This one will do some more checks, approve it, perhaps annotate it with further remarks, and will send it to the General Manager (step 3). The General Manager will inspect the request, inspect the annotations of both the Project Leader and the Department Manager and approve it, with the condition that enough investments funds are available in the budget. After this, the request is sent to the Financial Department (step 4). Here, the request is inspected (all requests must be previously approved by the executive staff), the costs for the new computer are evaluated, the company's investment budget is checked and finally, the request is approved. The request is then sent to the Acquisitions and Infrastructure department (step 5). Here the computer is ordered and a message is sent to Bob, letting him know that his new computer is on its way. However, because of internal regulations, the response is first sent to the company's archives, and only then sent to Bob (steps 6 and 7).
The persons involved in the workflow, follow a similar processing model to the security services:

1. They first analyze the document they received (inspecting the request and the annotations of previous persons);

2. Then the request is processed - the necessity is evaluated, funds are allocated (the security services process incoming messages in an analogous way);
3. Then the request may be annotated; the reason for annotating a message is to share some information (related to the request) with the persons along the communication chain;
4. Finally, the message is forwarded to the next person along the chain;
5. At any time, the request may be rejected (for example if there are not enough funds) or deviated from its regular itinerary (for example, if more information is needed in order to take the decision).

Chapter 5

Service Oriented Security Architecture - System Description

In this chapter, I will describe in more detail the scope and the functions of the system, the most important information elements, the distribution of the components together with the required infrastructure as well as implementation aspects. In order to capture all these topics in a clear and consistent manner, the proposed system will be described from five different viewpoints, as specified by the ISO RM-ODP [7].
The chapter starts with a presentation of the ISO RM-ODP standard and the reasons for choosing it as a means to describe the proposed security system. The next five sections describe ***SOSA*** from five different viewpoints, as required by the ISO RM-ODP: section 5.2 - enterprise viewpoint, section 5.3 - computational viewpoint, section 5.4 - information viewpoint, section 5.3 - computational viewpoint and section 5.6 - technology viewpoint.

5.1 The Reference Model for Open Distributed Processing

The Reference Model for Open Distributed Processing [7, 5, 6, 8] is a joint effort of the ISO and ITU-T to develop a coordinating framework for the standardization of open distributed processing [129]. It is an international standard for architecting open distributed processing systems that provides a framework of building distributed systems in an incremental manner.
The RM-ODP standards have been widely adopted and are used in the specification and architecture of distributed systems. [139] elaborates more on this topic and also presents three different examples for modeling architectures using RM-ODP: a proposal for the use of RM-ODP for the development of convergent applications, a generic architecture for customer relation management (CRM) systems and a framework based on ODP for the integration among different computer architectures. Furthermore, RM-ODP also constitutes the basis for several other standards. These include the OGC Reference Model [14], the OGC GeoDRM Reference Model [149], the ISO 19100 series of geomatics standards and the OMG object management architecture.
The reason for choosing the RM-ODP as the means to describe the proposed security system is that RM-ODP provides a rich set of architectural concepts and specification techniques that can be applied to distributed systems of different sizes and complexities. By using the RM-ODP, the proposed security framework can be defined in a consistent way. At the same time, because of

the separation of concerns introduced by the reference model, it is possible to specify for several parts of the system in a implementation-independent fashion (the dissertation will propose a SOAP implementation, but the security framework might as well be implemented for other messaging systems).
As distributed systems tend to be quite complex, the RM-ODP provides five viewpoints from which to describe ODP systems. These viewpoints abstract the system from different perspectives in order to simplify the description of complex systems. Table 5.1 introduces the different viewpoints and presents the main concerns for each of them.

Viewpoint	**Description**
Enterprise	Concerned with the scope, role, policies and activities of the system
Computational	Concerned with the description of the system as a set of interfaces, without regard to distribution
Information	Concerned with the information that needs to be stored and processed by the system
Engineering	Concerned with system distribution and the infrastructure required to support it
Technology	Concerned with the choice of technology chosen to implement the system

Table 5.1: The ISO Reference Model for Open Distributed Processing - viewpoints

Note that each of the viewpoints is a specification of the whole system - [7] advises that the different viewpoints should be seen as projections on to certain sets of concerns rather than layers or design methods. In that sense, the five viewpoints are interconnected, and influence each other. Figure 5.1 shows a graphical representation of the relationship between the different viewpoints. It also shows how the viewpoints can be related to the software engineering process, as described in [129].

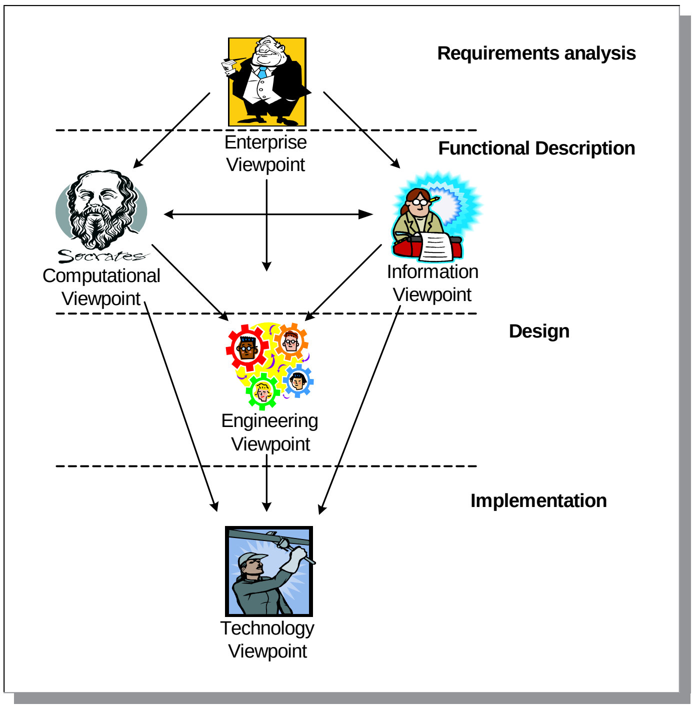

Figure 5.1: The five viewpoints of the ISO RM-ODP

5.2 The Enterprise Viewpoint

According to the RM-ODP, the Enterprise Viewpoint should address the roles played by the system, the activities it engages in and policy statements, including those that define the system in relation with its environment.
At the Enterprise level, the security system is seen as an atomic component. The different security services are not visible. Therefore, the requirements set here, are general requirements which are valid for most security systems for Web services.

5.2.1 System's Environment

In Web service interactions, the actors are a service requester, a service provider and, if we follow the SOAP model [88], one or more intermediaries. These entities can be grouped into administrative domains. An administrative domain can be as big as a corporate network or as small as an application client running on a user machine. Whenever a message enters or leaves an administrative domain, a security system might be needed. Its task is to ensure that the policies for the respective administrative domain are respected.
In terms of the relation with its environment the security system can be seen as a mediator[1] (intermediary) between clients (service requesters) and service providers, where the two are located in different administrative domains - figure 5.2 presents this metaphor. As such, the security system receives requests[2] from the clients and, after fulfilling various security functions (this might include audit, filtering, etc.) the security system will decide whether to authorize the request or not. If the request is authorized, the security system will hand it to the service provider. In a similar fashion, the security system receives responses from the services and, after similar processing, the response will be returned to the service requester.

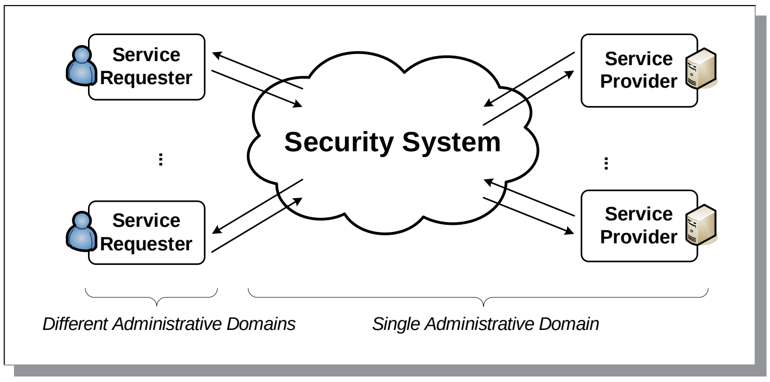

Figure 5.2: The mediator metaphor

Realization

There are various models, scenarios and implementations for security systems. For example the security system can be tightly-coupled to the service provider, forming a monolithic system. In such a case, the arrows in figure 5.2 are realized by API calls. The security system can also be loosely-coupled to the service provider, and in this case the arrows in the picture are realized by

[1]Traditionally, security systems are associated with a gatekeeper (like Cerberus from the Greek mythology). The reason for choosing another metaphor is that the purpose of the security system described here is much broader, going beyond restricting access to services.
[2]A request-reply message pattern is assumed for simplicity reasons. The concepts are also valid for other message patterns such as one-way only.

means of messages. Furthermore, the security system (or at least parts of it) can be attached to the service requester. One such example are trusted computing systems (like the ones associated with DRM systems) where enforcement is implemented locally, on the service requester side.
In this dissertation, we assume that the security system communicates by means of messages with the service requester and the service provider.

5.2.2 System's Role

From an enterprise perspective the scope of the system is to provide security functionality to one or more services (as illustrated in the mediator metaphor, figure 5.2). Figure 5.3 presents a UML sequence diagram illustrating the role played by the security system. For simplicity reasons, in this diagram only one service requester is accessing one protected service. However, as depicted in the mediator metaphor one security system can protect several different services against several different requesters.

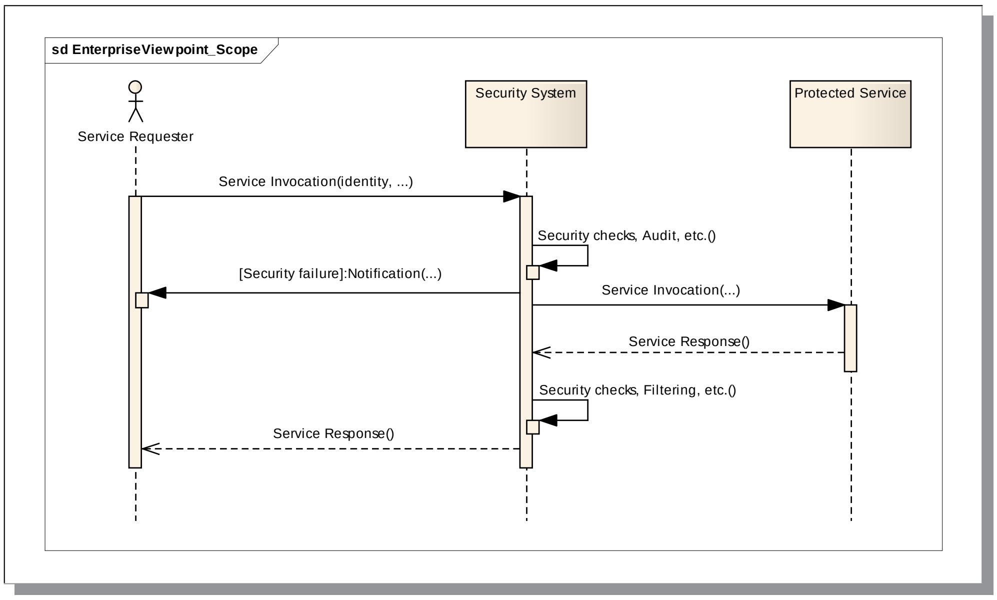
sd EnterpriseViewpoint_Scope
Service Requester
Security System
Protected Service
Service Invocation(identity, ...)
Security checks, Audit, etc.()
[Security failure]:Notification(...)
Service Invocation(...)
Service Response()
Security checks, Filtering, etc.()
Service Response()

Figure 5.3: The role of the security system

As seen here, the security system is located between the client and the protected service and has the following tasks:

- It intercepts all messages incoming from the client;
- These messages are inspected together with the client's credentials, and the security policies are evaluated; this might include the verification of client's identity, enforcement of access control, etc.;
- If the security system decides that the client is not allowed to perform the service invocation, a failure notification will be sent back;
- Otherwise the message is passed to the protected service, that will process the request and return a result;
- The result is also intercepted by the security system. At this point the security system does additional operations. This might include additional checks to determine if the client is allowed to receive the result, encryption of message parts, filtering, etc.;

- Finally, the client will receive a response from the security system. This can be one of the following:
  - An exception informing it that the security checks were unsuccessful;
  - A modified version of the response sent by the protected service (filtered, encrypted, etc.);
  - The response generated by the protected service (unchanged).

At this point it is important to make the following remarks:

1. The client sends some form of credentials together with its request. Most commonly the credentials relate in some way to the user's identity (examples include usernames, digital signatures, etc). However, other credentials can be imagined, such as proofs of payment, authorization assertions, etc. Furthermore, the credentials need not be explicitly send. Instead, they can be implicitly determined by the security system (one such example are IP addresses).
2. The security system is trusted by the protected service to enforce the security policies on its behalf.
3. The security policies can be as simple as only logging the client's requests to very complex rules involving distributed access control, message filtering, encryption, etc.

5.2.3 Functions

As showed in figure 5.2, the security system is located in the same administrative domain with the protected services. It's purpose is to enforce the domain specific security policies. This includes all of the main security functions described in 2.4: authentication, authorization, confidentiality, integrity, non-repudiation, privacy and availability. Furthermore, the security service might need to provide other security related functions such as accounting and charging.
As the protected services and the security policies may be part of enterprise-specific workflows, the security system should be extensible and should facilitate the integration with other services.

5.2.4 Multiple Security Domains. Security Off-Shoring

It is expected that, at least in some cases, not all of the security functions will be performed in the same domain or organizational unit[3]. This is showed in figure 5.4.

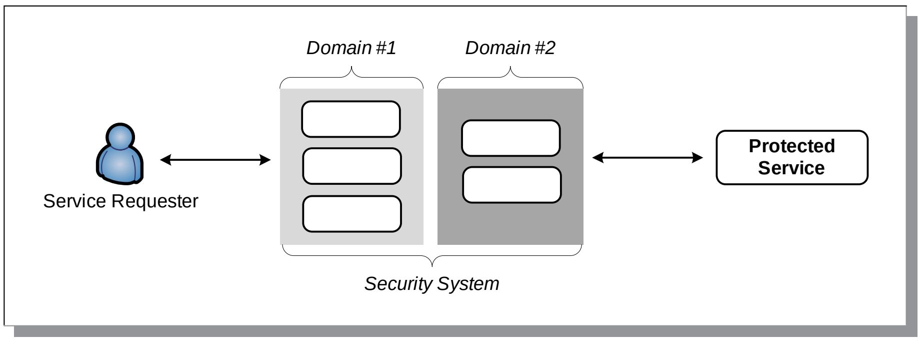

Figure 5.4: The security system spanning several domains

The motivation for this resides in the fact that a distribution of the security systems allows a more

[3]Organizational unit is used here in a broad sense: it can be a company, a department within a company, or simply a part of the network that is under a different administration, possibly (but not necessarily) having different security policies

flexible solution. While dividing the responsibility and control over of the system in the same domain allows for separation of security concerns, delegating these tasks to different organizations would allow such things as distributed decision making, infrastructure consolidation and off-shoring of tasks. Let's look at two motivating examples before moving further:

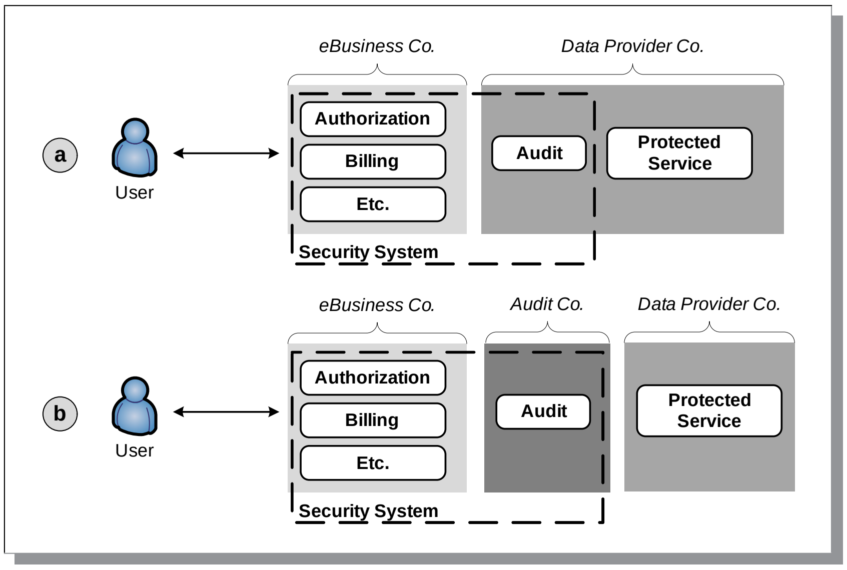

Figure 5.5: Geospatial business example

Geospatial eBusiness A geospatial data provider sets up a service that delivers geospatial information to clients. Because collecting and processing the geospatial data is expensive, the company will not offer the service for free. Furthermore, for several reasons (Data Provider Co. has little eBusiness know-how, etc.) they decide not to sell the service themselves. Instead, they decide to use intermediate companies that advertise and sell the service, returning a certain percent of their selling to them. The scenario is presented in figure 5.5. Here, the dealers (in the picture, eBusiness Co.) will only allow the users that pay to access the system. On the other hand, because Data Provider Co. expects a certain percent from what the dealers sell, Data Provider Co. will make an audit of all transactions. Therefore, the security system is composed from different components that reside in different companies (see dotted line). The example can be extended to a scenario where Data Provider Co. is not doing the audit, but instead is delegating this task to a third party (Audit Co.).

Distributed Decision Making This is the case where the authorization decision can not be taken in a single point. As an example, let's look at the case of a bandwidth broker in a multi-domain scenario (this use-case is also described in RFC2905 [148]): a user wants a connection to another host with specific QoS parameters (see figure 5.6). In order to do this, he will make a request to the local bandwidth broker. The local bandwidth broker will inspect the network topology, verify if the requested bandwidth can be allocated for the connection with the neighboring network and, if this is possible, authorize the request. However the user needs an end-to-end connection with the requested bandwidth! In order for this to happen, all the bandwidth brokers responsible for the intermediary networks need to accept the user's reservation request. Because the networks are typically under different administrative domains (usually different operators), and because the requests for bandwidth are made dynamically, it is not possible to have a centralized place where the decision can be taken.

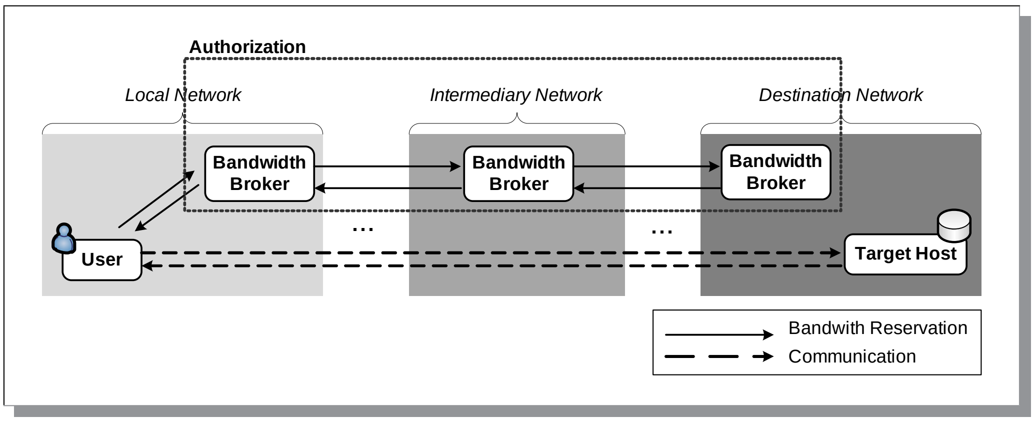

Figure 5.6: The bandwidth broker example

5.3 The Computational Viewpoint

According to the RM-ODP [7], the Computational Viewpoint defines the functional decomposition of the system into a set of services that interact at interfaces. Furthermore, this viewpoint should capture the details such as internal actions of these services, interactions that occur between services and contracts between the services and their environment, without regard to distribution [129, 14]. Accordingly, in this section we will decompose the security system into functional components (security services) and will investigate the behavior and requirements for these components. We will do this by analyzing the internal actions of the services and the interactions between them. Furthermore, we will investigate the types of services that can be found in a security system and present a classification based on the functionality of services.

5.3.1 Computational Object

Before going any further, lets first look at what objects and computational objects are in terms of the ISO RM-ODP. [20] describes objects and computational objects as follows:

Object An *object* is a model of an entity. An object is characterized by its *behavior* and, dually, by its *state*. An object is distinct from any other object. An object is encapsulated, i.e. any change in its state can only occur as a result of an internal action or as a result of an interaction with its environment.

Computational Object A *computational object* is an object as seen in the computational viewpoint. It represents functional decomposition and interacts with other computational objects. Since it is an object, it has *state* and *behavior*, and its interactions are achieved through interfaces.

5.3.2 Security Services

As mentioned in the enterprise viewpoint, security systems can easily become quite complex. We will therefore split it into smaller parts or subsystems. From a computational viewpoint, each such a subsystem should be regarded as a *computational object*: it has *state* and *behavior* and it interacts with the other subsystems in order to fulfill the scope of the security system. Here, these subsystems are called *security services*.

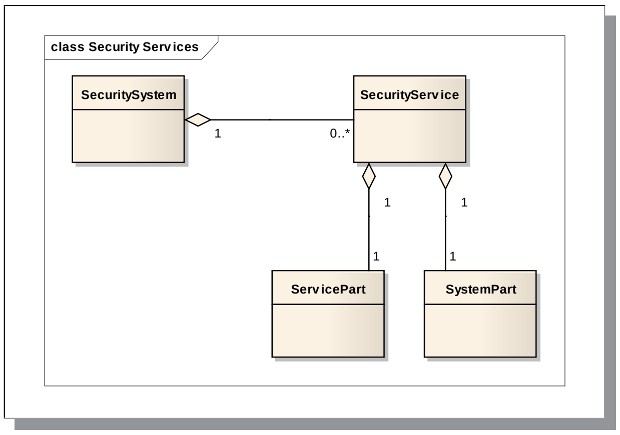

Figure 5.7: Security services

As it can be seen in the UML diagram (figure 5.7), a security system can be composed from several security services and each security service is composed from two distinct parts:

Service Part The service part is specific to the security service. It implements the functionality of the security service. For example an audit service might have a logging routine as its service part, while a non-repudiation service might have a digital signing routine as its service part.

System Part The system part defines how services relate to the system as a whole. It is concerned with the choreography/orchestration of the security services (i.e. how the security services interact in order to fulfill the purpose of the security system). In a typical implementation scenario, services run on top of some sort of a middleware system which implements the system part.

The distinction of the two parts need not necessarily be reflected in the implementation of the services/system. Nevertheless, from a computational viewpoint it is necessarily to make a clear distinction between these two types of behaviors.

5.3.3 Service Interactions. Interaction Interfaces

Figure 5.8 shows how the security system is interacting with its environment (the service requester and the service provider). If there is no security system in place (figure 5.8A), the service requester communicates directly with the service provider using external messages[4]. When a security system is deployed (figure 5.8B), this one will act as an intermediary between the service requester and the service provider. Both the communication between the service requester and the security system and the communication between the security system and the service provider is realized through external messages. This makes the security system easy to deploy because it is practically transparent to both client and protected service.
As mentioned in the previous section, internally, the security system is structured into several services. Exactly how these services are organized and distributed is not the concern of this viewpoint (we will look at this in the engineering viewpoint). Instead, here, we will investigate the behavior of these services. Figure 5.9 shows a UML diagram for this.

[4]Messages are called *external* because they are external to the security system.

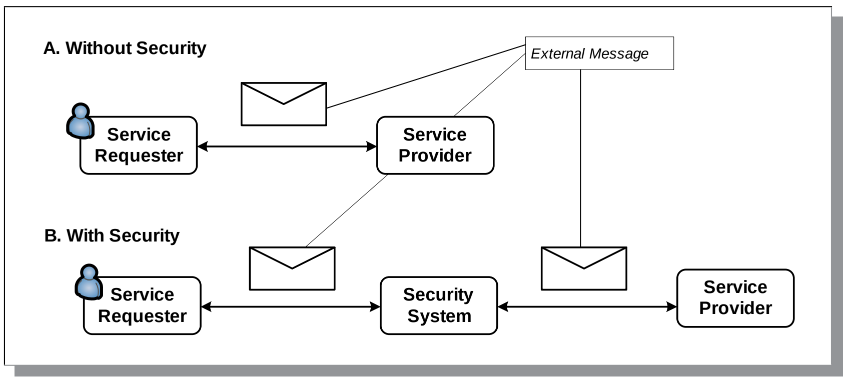

Figure 5.8: Interactions through external messages

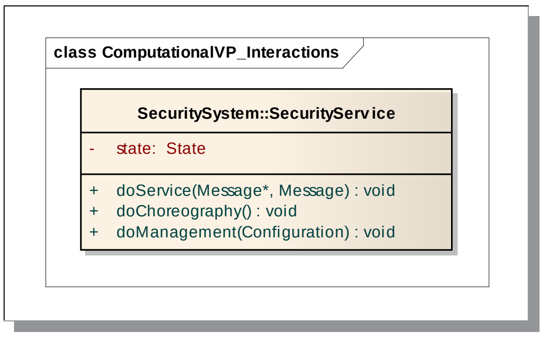
class ComputationalVP_Interactions
SecuritySystem::SecurityService
- state: State
+ doService(Message*, Message) : void
+ doChoreography() : void
+ doManagement(Configuration) : void

Figure 5.9: Behavior of security services

First, a security service has an associated state. The state is internal to the service. The service exposes a management interface - doManagement - that allows the state to be changed. This can be used to initialize the service or to change the configuration or other parameters (for example an encryption service can be configured by specifying the encryption key). Other administrative tasks (such as service monitoring, etc.) can be imagined as well. Self-understandable, stateless services are also possible: such services will not need to expose a management interface because they always provide the same response when they are interrogated with the same request.
The functionality of the service is given by the doService method. As seen in the UML diagram, the security service receives an input message, processes it and produces an output message. Without going into details (the semantics of the message will be described in the information viewpoint), it is worth noticing that internal messages[5] also contain meta-information: an internal message contains the external message (or the payload - such as the request from the client) and several blocks of meta-information (annotations, as discussed in section 4.3).
Last, in the UML diagram doChoreography realizes the choreography. It is showed as private, because other services do not directly invoke the choreography.

5.3.4 Service Workflow

Even though the services implement different functions, they all follow some general principles:

- As identified in the enterprise viewpoint, the purpose of the security system is to mediate the communication between clients and the protected services. Therefore the main task of all security services is to process messages in order to apply security functions.

[5]A message, as used internally by the security system.

- In order to accomplish the system's scope, security services must work together. As previously identified, services communicate with one another by means of annotations.

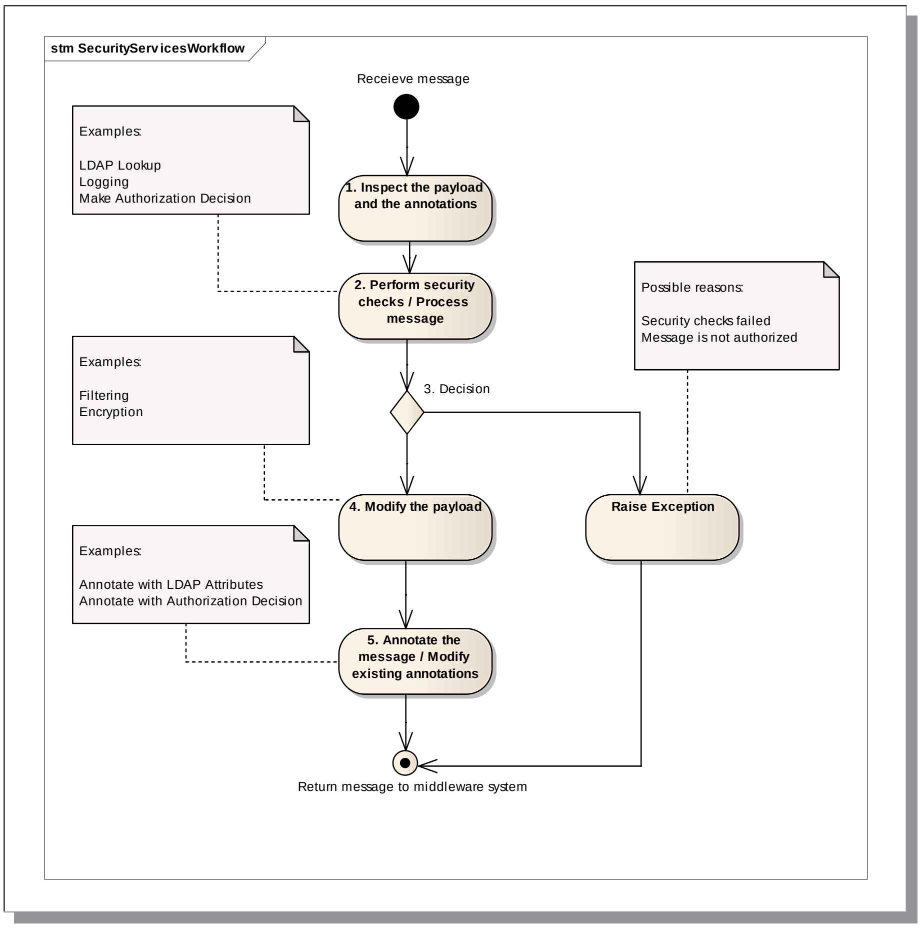
stm SecurityServicesWorkflow
Receieve message
Examples:
LDAP Lookup
Logging
Make Authorization Decision
1. Inspect the payload and the annotations
2. Perform security checks / Process message
Possible reasons:
Security checks failed
Message is not authorized
Examples:
Filtering
Encryption
3. Decision
4. Modify the payload
Raise Exception
Examples:
Annotate with LDAP Attributes
Annotate with Authorization Decision
5. Annotate the message / Modify existing annotations
Return message to middleware system

Figure 5.10: Internal workflow for security services

The general internal workflow for security services is depicted in figure 5.10 and explained in the following:

1. Upon receiving a message, a security service will first analyze the message and inspect the payload of the message and the annotations created by the services that previously processed this message.

2. Based on both the payload and the annotations, the processing logic will be applied. Examples for this are verifying the identity of a user by looking up his username in an LDAP directory or making an authorization decision based on the information in the payload and the annotations that were prepared by services that have already processed the message.

3. If the security checks failed, or the processing logic could not be performed correctly, an exception is raised. The messaging middleware will dispatch the exception (see next section for more about how exceptions are handled).

4. As a result of the second step, the payload may be modified: parts of it can be filtered out, it can be augmented or transformations can be applied (for example encryption).

5. If the service needs to share information with the other services, it will introduce additional annotations and / or modify the exiting ones.
6. Finally, the message is returned to the middleware which applies the choreography rules and hands out the message to other services.

5.3.5 Exceptions

If the security checks fail or if the processing logic cannot be correctly applied, security services can raise exceptions signaling the errors encountered.
Here, a model similar to the SOAP model [88] is used for exceptions: an exception is a message with a distinct payload containing details about the fault (reason, service that generated the fault, etc). Besides this, the exception messages may also contain annotations and other meta-information. Exception messages must be distinctly marked as such, so that the middleware and other services can recognize them and treat them appropriately.
Because exceptions are treated like regular messages, they can be processed in a similar way with request and response messages. It is up to the messaging middleware to properly route the exception messages to appropriate services that handle them.

5.3.6 Service Taxonomy

In this section we will investigate the possible types of services that could be found in a security system. For each of the services I will describe its functionality by following the steps described in the previous section: we will look at how messages are processed, how the payload and annotations are altered and what (if any) decisions regarding the choreography are taken.
The description of the security services is done from a computational point of view. We will therefore focus on the functionality of each service and on the interactions between the different services. For each of them I will specify the *prerequisites* - what the service expects from the other services that have processed the message at a previous point in time - and the *responsibilities* - the functional description of the service, including the communication between this service and other services.
Note that the security services enumerated here are not mandatory and the list of security services is not intended to be complete: not all services presented here must appear in a security system and other security services may be imagined as well. Furthermore, depending on the specific requirements of the protected service, the security services might be coarser or finer defined.

Gateway Service

On its way from the service requester to the protected service a message can travel by means of different transport protocols. All transport protocols are able to convey some kind of security-relevant information. However, when switching from one protocol to another, it is not always possible to pass this information forward - some protocols are richer in security information than others. The only way to relay such information is to extract it from the transport protocol and append it to the message.
One place where this usually represents an issue is at the network perimeter, where the message coming from the service requester enters the security system. At this point, a gateway service can be envisioned which extracts security relevant information from the transport protocol and appends it to the message in the form of annotations.

Furthermore, because the gateway service is located at the network perimeter, it must also convert messages from external messages (without annotations) to internal messages (with support for annotations). The Envelope Wrapper pattern (described in 2.3) is applied here: the original message is wrapped around an enveloping message which is semantically richer.

Prerequisites None.

Responsibilities

1. Wraps the external message in an internal message (with support for annotations);
2. Extracts security relevant information from the transport protocol layer (i.e. IP address, HTTP Authentication Information, HTTP Cookies, SSL/TLS information) and appends it to the message in the form of annotations, so that other services can make use of it.

Authentication

Authentication is a key security issue because without knowing the identity of the requester other security functions cannot be fulfilled (for example you can not charge someone unless you know who he is).
According to [131], an authentication service is a security service that verifies an identity claimed by or for an entity. The same source further differentiates between data origin authentication service and peer authentication service. In the case of Web services, where communication is message-based, data origin authentication is important - a service provider wants to know that the message originated from the claimed sender. Peer authentication, in the case of message-based systems, is also done by means of data origin authentication.
As described in section 2.4.2, authentication consists of two different activities: *identification* and *verification*. Because the two activities are distinct, they can be realized as separate security services.

Verification Service

Service requesters normally attach some information asserting their identity to service requests. This information can be of one of the following two distinct types: information identifying the requester and information corroborating the binding between the issuer and the identification information (credentials).
The task of a verification service is to check that the authentication information accompanying the message are valid.

Prerequisites The service expects some credentials asserting the identity of the requester to be present in the message. Additionally, the service should extract any other such credentials (if there are any) from the communication channel.

Responsibilities Verify that the information asserting the identity of the requester is correct. This might include checking the validity of digital signatures and certificates, verifying passwords and other authentication credentials.

Remarks

- It depends on the system's policy, whether or not a message is rejected if a credential does not validate. For example, certificates that verify but are expired might be accepted. Especially in such cases it makes sense for the service to annotate the message with information regarding the problems encountered when verifying credentials. The decision whether to drop the message or not is in this case deferred until the authorization phase.

Identification Service

Identification refers to the process of distinguishing the requester from other entities. This is done by means of attributes. Some identification information is included by the requester himself in the message. However, in most of the cases, this information is insufficient to perform the other security functions such as access control, audit, charging, etc.
The task of an identification service is twofold: first, it must verify the identification information sent by the requester. Then, it must enrich it with sufficient additional attributes for the other services to be able to perform their tasks.

Prerequisites The service expects some information related to the identity of the requester to be present in the message. Alternatively, if such information is not present in the message, the service should be able to procure it dynamically from other sources (such as the transport protocol). Furthermore, in some cases, an identity might need to be assigned to the requester (for example when a session is initiated).

Responsibilities An identification service will typically have the following behavior:

1. Verify that the supplied identification information is acceptable, sufficient and can be understood. The purpose of this step is for the service to be able to differentiate the requesting party from other entities;

2. Lookup the user in a database (such as an LDAP directory) and retrieve additional attributes about the user;

3. Annotate the message with the attributes related to the user and/or the authentication method, so that other services can make use of this information (for example the bank account might be required by the charging service).

Remarks

- It does not always make sense to split the two authentication steps into separate services. Depending on the specific requirements of the application being protected, the two services can be implemented as a single service or as separate services.

- Identification services are useful in federation scenarios where requesters may have different identities or they may be registered in different domains. The domains most probably use different identification schemes. In such a scenario, an identification service can be deployed that converts different kinds of identities to a *canonical identification schema* that is then used through the rest of the system. In this way the identification is done centrally and, thanks to the annotations, the other services can see the user as having a single identity (without regard to the authentication method used or the registration domain), thus minimizing the administrative overhead for other tasks such as rights management.

Authorization

As explained in section 2.4.2, authorization is the process of determining whether the requester may use the protected service in a certain context and the process of enforcing this decision. This is perhaps the most obvious security function of all, as for most of the applications it is crucial that only *certain* users are allowed to perform *certain* operations that involve *certain* information.
Because authorization involves several tasks, most of the standards dealing with authorization propose architectures where these tasks are distributed among different services. In the following, we will look at how authorization is distributed in RFC2904 - AAA Authorization Framework [147] and the OASIS XACML 2.0 standard [18]. OASIS's XACML standard defines the following components[6]:

Policy Administration Point (PAP) The system entity that creates a policy or a set of policies.

Policy Information Point (PIP) The system entity that acts as a source of attribute values.

Policy Decision Point (PDP) [1] The system entity that evaluates applicable policies and renders an authorization decision.

Policy Enforcement Point (PEP) [1] The system entity that performs access control, by making decision requests and enforcing authorization decisions.

IETF's AAA Authorization Framework defines the following components:

Policy Retrieval Point (PRP) An entity responsible for retrieving the authorization policies from a policy repository on behalf of the organization that requires it.

Policy Information Point (PIP) An entity that provides information against which policy conditions are evaluated (such as resource status, session state, or time of day).

Policy Decision Point (PDP) A logical entity that makes policy decisions for itself or for other network elements that request such decisions [133].

Policy Enforcement Point (PEP) A logical entity that enforces policy decisions [133].

As it can be seen, both specifications propose similar components. The PIP, PDP and PEP have the same name, perform similar tasks and are interconnected in similar ways in both models (figure 5.11 shows the data flow between the different components in XACML, as described in the specification). For policy management, XACML describes the PAP service, while [147] proposes the PRP service. However, both services have similar functionality - their purpose is to provide the PDP with policies that this one can then use in the decision making process.
Having the two service definition patterns (mediation vs. invocation) discussed in the previous chapter in mind (section 4.2.2), the four components can be implemented as follows:

PIP, PDP These two services can be realized as both mediation services as well as RPC. XACML [18] proposes an RPC definition for these services. The following sections show how the two services can be realized as mediation services.

[6]The descriptions for each component are taken from the XACML specification.
[1][18] states: *This term is defined in a joint effort by the IETF Policy Framework Working Group and the Distributed Management Task Force (DMTF)/Common Information Model (CIM) in [RFC3198]. This term corresponds to "Access 272 Enforcement Function" (AEF) in [ISO10181-3]*

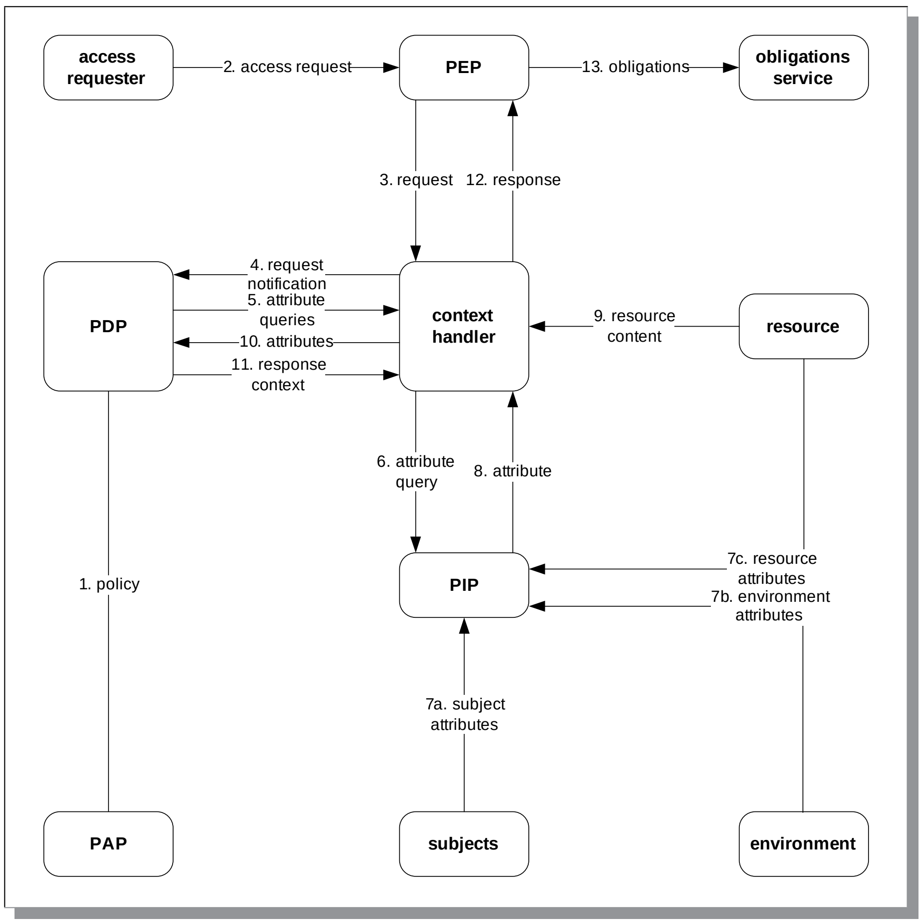

Figure 5.11: Data flow and component interconnections in XACML [18]

PEP If realized as a service separate from the rest of the security system, it only makes sense to realize this service as a mediation service.

PAP / PRP The administration of policies and the policy repository, if exposed as a service, it only makes sense to be realized as an RPC service. A possible implementation of the PAP as mediation service, would extract the policy from the repository and attach it to the message, in order for the PDP to evaluate it. Such an implementation introduces too much overhead and has little practicability.

Figure 5.12 illustrates how the authorization services can be chained together in an authorization process (in this figure, other security services are neglected for simplicity reasons). The engineering viewpoint (section 5.5) describes in more detail the how different security services can be combined and several patterns are proposed.

Policy Information Point - PIP

The role of the PIP is to provide information against which policy conditions are evaluated. It must provide the PDP with enough information for this one to be able to make the access control decision.
In XACML [18], the PIP provides information in the form of attributes. Three types of attributes are defined: *subject attributes* (refer to the service requester), *resource attributes* (refer to the

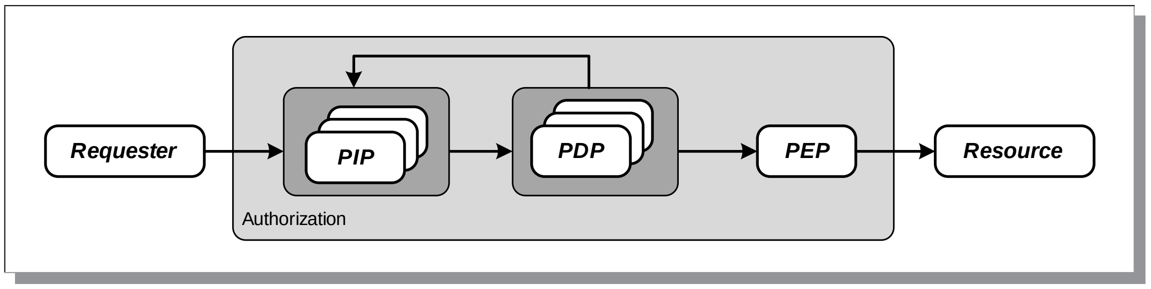

Figure 5.12: Data flow and component interconnection for authorization in ***SOSA***

resource being accessed) and *environment attributes* (refer to the context / environment).
As a security service, a PIP may provide any of the three type of attributes. It is also possible to have several PIP services, each of them providing a different kind of information. It should be remarked that the previously defined identification service provides attributes associated to the requester - *subject attributes*, and in that sense it is also a kind of PIP.

Prerequisites None

Responsibilities The service inspects the messages it receives and annotates them with attributes. These attributes will later be used by PDP services when evaluating the policies.

Policy Decision Point - PDP

As mentioned earlier the PDP's duty is to make authorization decisions based on the applicable policies. A PDP service first needs to determine the policies which are applicable for the given context (message, requester, environment). To do this, the PDP may query one or more PAP services which will provide the policies to the PDP. The mechanisms for this, together with a description of how different policies can be combined and how conflicting rules are resolved is described in the XACML specification [18].
In the policy evaluation process, the PDP may require additional information about the requester, the environment or the resource being accessed. In XACML, a *pull model* is used (see figure 5.11) - the PDP requests attributes from the context handler, a logical component which then connects to different PIP services. This model assumes PIP services expose an RPC-like interface. In this dissertation it is assume that security services are realized as mediation services. In this case, a *push model* is necessary: the PIP services annotate messages with attributes, in this way pushing the information to the PDP service.
Of course, the two models do not exclude one another, and therefore it is possible to realize PDP services which combine the two models: some of the attributes are pushed to the service by mediation PIP services, while other attributes are retrieved by connecting to PIP services which expose an RPC-like interface.

Prerequisites The PDP requires enough information in order to be able to evaluate the applicable policy. Most access control decisions are based on the identity of the requester; it is the task of identification services to provide identity attributes. Further attributes (related to the subjects, the environment or the resource being accessed) may be provided by PIP services.

Because it is possible to have several PDP services chained together, the PDP may expect that the message contains the authorization decision of other PDP services.

Responsibilities

1. Determine the applicable authorization policies. For this the PDP may query PAP services as described above.
2. Based on the authorization policies, the payload of the request (the external message) and other relevant information (identity attributes, other attributes provided by PIP services, authorization decisions from other PDP services) make an authorization decision.
3. Annotate the message with the authorization decision so that other services (PEPs, Audit, other PDPs, etc.) can use this information.

Remarks

- The decision need not be boolean (i.e. "authorized" / "not authorized"). XACML [18] defines the concept of obligations: PDP services can include obligations in their response in addition to the decision. In this case, the PEP must grant access only if the obligations can be discharged.
- Several PDP services may exist in a security system. The PDPs can authorize different parts of the message or they can be arranged in a sequence so that the decision result from one service is used as input for the other services (for more, see the Distributed Decision Making Pattern in the Engineering Viewpoint, section 5.5).

Policy Enforcement Point - PEP

As discussed earlier, the PEP's duty is to enforce the authorization decisions taken by one or more PDP services.

Prerequisites Typically, the PEP expects one or more authorization decisions to be attached to the message (it is possible to have several PDP services in a security system). Furthermore, if PDP services also implement obligations, the PEP may also expect obligations to be present in the message.

Responsibilities

1. Retrieve the authorization decision and the obligations from the message.
2. Enforce the authorization decision and discharge the obligations. The enforcement is typically done by denying the requests for which the authorization decision is not authorized or for which the obligations can not be discharged. Finer grained enforcement methods can also be imagined, where the message is filtered based on the authorization decision, so that only a subset of the original request/result passes the enforcement point (a message filter security service is showed on the UML diagram in figure 5.13).
3. Optionally, annotate the message with the reason for rejecting / filtering the message. This is useful to other security services (such as audit, etc.) and informative to the client (knowing the rejection reason, this one can fix the problem and resend its request).

Remarks

- XACML recommends that the PEP be placed at a choke point, which cannot be bypassed - this is usually close to the resource being protected. Therefore, a good strategy is to have a distinct PEP for every service being protected by the security system
- While PEPs need not necessarily know the semantics of the payload in order to enforce an authorized/not authorized decision, in order to build a PEP with filtering capabilities, detailed knowledge about the payload semantics are needed.

Audit

[49] defines an audit guard as a mechanism used on behalf of an owner that monitors actions and agents to verify the satisfaction of obligations. It is typically used to monitor the state of a resource or service and determines whether the associated obligations are satisfied.
The same source also states that by their nature, it is not possible to proactively enforce obligations; hence an obligation violation may result in some kind of retribution after the fact of the violation.
[47] further differentiates between *passive* and *active* auditing, where passive auditing is essentially logging with the expectation that the log will be later analyzed and, perhaps retributions be made. Active auditing is the complement of passive auditing and determines if the information in the message constitutes an exceptional condition (such as a violation of the obligations) and if so, takes action.
Audit guards can be implemented in several forms, and the exact behavior of an audit service is in big part dependent on the service being protected (especially if we have active audit guards in mind). In the following two sections we will propose two general-purpose security services that illustrate passive and active audit guards.

Logging Service

A logging service is a security service that logs certain parts of a message. It is important to remark that it can log both parts of the payload (external message) as well as meta-information such as annotations - for example authorization decisions, authentication statements, etc.
A logging service is an example of a passive audit guard: its main duty is to persistently store information so that later, this can be analyzed and possibly, retributions be made.

Prerequisites None.

Responsibilities To log parts of the message. Because this is a passive security service it will not produce new information that needs to be communicated to other services, and therefore will not annotate the message.

Remarks

- A logging service is transparent to the other services. It makes no difference for them if the service is there or not because other services do not require information produced by the logging service. However, having the system's choreography in mind, other services might prepare audit information for the logging service.

Notification Service

A notification service is an example of active audit guard. Based on the information found in the message (both payload and meta-information), the notification service would determine if an exceptional state is reached and in this case send a notification. The notification could be sent to a human (for example an email to the system administrator describing the situation) or to a machine (an external Web service is invoked).

Prerequisites None.

Responsibilities To determine if an exceptional state is reached and if so, to send a notification.

Remarks

- Similar to the logging service, a notification service is normally transparent to the other services.
- The task of determining whether an exceptional state is reached can also be delegated to a PDP service. In this case a notification service would resemble a PEP who's enforcement strategy is "allow access and send notification".

Cryptographic Services

Cryptographic security services are services responsible for applying cryptographic functions either to the whole message or only to specific parts of this one (the payload, parts of the payload or annotations). Through the use of cryptographic functions the following security requirements are addressed:

Confidentiality Achieved by means of encryption;

Integrity Achieved by means of digital signatures;

Non-Repudiation Achieved by means of digital signatures.

The following three sections describe three cryptographic security services: a digital signature service, an encryption service and a service for the distribution of DRM-protected media. The need for having such services separated from the protected service has been remarked in several sources including [88, 49].

Encryption

An encryption security service would simply encrypt parts of the message. As such, it ensures the *confidentiality* of the data being encrypted: only the entities possessing the decryption key are able to read the data. The data is protected while in transit against eavesdroppers that tap the communication channels and against intermediaries (such as external services or other security services, etc.) that further process the message. The encryption algorithm and keys can be either statically configured or dynamically assigned (in this case the service would also determine which key should be used for which messages). For more about granularity see section 5.3.7.
A typical usage scenario for an encryption service is where a client invokes the protected service and mandates that the response message be encrypted with his public key.

Prerequisites None.

Responsibilities

1. Determine what parts of the message shall be encrypted, what algorithms and keys shall be used;
2. Apply the encryption algorithm.

Digital Signature

A digital signature security service digitally signs parts of the message it processes. As such, it ensures the *integrity* of these parts (i.e. other entities processing the message can be sure that these parts were not altered) and *non-repudiation* (i.e. the receiving party has a proof of origin). As in the case of the encryption service, the algorithm and keys to be used can be either statically configured or dynamically assigned.
One typical usage scenario for a digital signature service is where a client invokes the protected service and mandates that the response message be digitally signed by the protected service.

Prerequisites None.

Responsibilities

1. Determine what parts of the message shall be digitally signed, what algorithms and keys shall be used;
2. Apply the digital signatures.

Digital Rights Management

[103] defines Digital Rights Management as a collection of technologies that enable technically enforced licensing of digital information. In most of the DRM systems the digital contents has an associate digital license which states the conditions under which the content might be used. In order to enforce the license, the content is usually packaged together with its license and then encrypted. Only authorized devices (that know the semantics of the package and have knowledge of the decryption key) are able to decrypt the package and make use of the content.
A DRM security service is a service that assigns a license to the content (this one is typically found in the payload), bundles the license together with the contents and finally encrypts the bundle so that only authorized devices are able to make use of the contents. It should be remarked that the emphasis is set here on associating an appropriate license to the content and not on the encryption process.
Such a security service would typically be used when distributing media that has associated intellectual property rights. There are many examples for this: music, video, books, etc.

Prerequisites None.

Responsibilities

1. Associate a license with the contents;
2. Package the license together with the contents;

3. Encrypt the package using the appropriate algorithm and keys.

Remarks

- There need not be a single content part in the payload. Instead, the payload may contain several content parts with different associated licensees.
- Another practice used in the distribution of digital media is *watermarking*: small amounts of information are embedded in the content so that at a later point in time the origin can be traced. This functionality can be implemented by a different type of DRM service.

Accounting

As identified in [69], services can be divided in two categories:

- Simple services (for example a stock quote service), where there is no quality of service guarantee, which are generally available at no charge;
- Services that clients use in business transactions, in which quality of service plays an important role, which are most probably not offered free-of-charge.

For the later category, the service provider needs to meter the usage of the service, so that he can later charge for the usage. Charging is discussed in the next section. For the purposes of accounting, it's important to notice that charging could be either done on a periodical basis (i.e monthly, yearly, etc.) or immediately.
In the proposed architecture, accounting can be implemented as a security service who's task is to measure the usage of the protected service for each of the users. How the usage is measured, depends on the kind of service provided and on the business model chosen. Examples of existing business models used for Web services are: pay-per-click, free-for-use, subscription, lease.

Prerequisites Because accounting is done on a per-user basis, the accounting service requires that an authentication service had already authenticated the user, and had annotated the message with details regarding his identity. In most of the cases, accounting is also audited. In this case the presence of an audit service is also required.

Responsibilities

1. Extract information regarding the user identity from the message (this has been previously prepared by an authentication service);
2. Compute the usage of the protected service;
3. Either store this information in a database (if the charging is done on a periodic basis) or annotate the message with information regarding the usage so that a charging service can charge for the usage of the service right away.

Charging/Billing

The main task of a charging/billing service is to charge users for the usage of the protected service; this is done based on the information provided by the accounting service.

Prerequisites A charging service requires that an authentication service has identified the user and also annotated the message with information sufficient to for charging. Additionally, because charging is done based on accounting information, the charging service requires that an accounting service has also annotated the message. Finally, as in the case of accounting, charging is also normally audited.

Responsibilities

1. Extract information regarding the user's identity, especially the information necessary for charging from the message; also extract information regarding accounting;
2. Based on the accounting information, compute the amount of money that the user needs to be charged with;
3. Charge the user;
4. Optionally, mark the message as "charged" (by means of annotations) so that authorization services can use this information in authorization decisions.

Remarks

- The accounting and charging services are tightly coupled and need not necessarily be separately implemented.

Message Validation

A message validation service must simply verify that the semantics of the message are respected. This function is important especially when messages are coming from an unreliable source. This service is intended as a countermeasure against several common types of attack targeted at systems not performing input validation: buffer overflow, SQL injections, etc. [94, 71, 56] provide further information about message validation strategies and implementation.

Prerequisites None.

Responsibilities To verify that the message is valid and if not, to drop the message and optionally raise an exception.

Replay Prevention

Another traditional type of attack is where an attacker repeatedly transmits a valid message with a malicious intent. Usually the middleware layer provides prevention mechanisms for this type of attack. However, if the middleware layer is not able to prevent such attacks, this functionality can be implemented in a standalone security service, who's task is to ensure that messages are not replayed. This is usually accomplished by the use of unique message identifiers, nonces or timestamps which are stored in a replay cache. [94, 35] further detail replay attacks in the context of Web services and provide guidance for the implementation of countermeasure strategies.

Prerequisites None.

Responsibilities To prevent messages from being replayed.

Infrastructure Services

The following sections we will present a couple of services that do not deal with security relates aspects. These services are named *infrastructure services* because they provide basic functionality to the security services. They have also been identified in [56] as specialized services in Enterprise Services Bus architectures, and have been briefly discussed in section 2.3.6.

Orchestration Service

An orchestration service is a special processing service that centrally coordinates the execution of several other services. This is usually done by invoking the other services according to some algorithmic workflow, similar to the way functions are invoked in procedural programming languages. Such a service could address sophisticated process management situations where several services need to interact in a complex workflow. Standardized scripting languages for describing business processes (ex. BPEL4WS) exist and implementations available on most of the platforms, making orchestration services a convenient choice.

Prerequisites None.

Responsibilities To centrally coordinate the execution of other services.

Message Transformation Service

A message transformation service is a processing service which transforms messages from one representation to another, according to some transformation rules. Besides modifying the representation of the message, a transformation message might also filter the contents of the message or augment it. Transformation services can be used for several reasons:

- To integrate services that use different representation format for messages;
- To filter information before invoking a service (for example for privacy reasons). In this case the transformation service removes some of the annotations and / or parts of the payload.

If messages are encoded in XML, XSLT can be used to specify the transformation rules, making this service very convenient and with a high degree of reusability.

Prerequisites None.

Responsibilities To transform messages according to transformation rules.

Message Storage

A message storage service is a specialized service which stores messages. Message storage can be either *persistent* or *temporary*. Persistent storage of data may be required in some authorization scenarios where the intervention of a human person is necessary. In this case the message needs to be stored until the human person analyzes the message and takes the authorization decision (for more, see the Asynchronous Decision Making Pattern in the Engineering Viewpoint, section 5.5). Temporary storage is useful in scenarios where caching is required.

Prerequisites None.

Responsibilities To temporarily or persistently store messages.

UML Diagrams

Figure 5.13 shows a possible UML representation for the taxonomy of security services described in this section. Figure 5.14 shows the dependencies between services, as identified in the prerequisites clause for each of the services.

5.3.7 Granularity and Reusability

The granularity of the services is also an issue to be investigated. While having fine grained defined services brings a good separation of duties which in turn makes the services reusable, this also has its negative impact because it makes the choreography of the services complex, and may render performance problems. When making a choice regarding the granularity of the services, it is important to consider all the requirements of both the service requesters and the protected service. Another issue related to granularity is reusability. There is always the choice between making a specialized service and a more general, configurable and therefore a more reusable service. The following shows some examples:

Cryptographic Services Here the cryptographic algorithms and keys can be either internally defined or they can be dynamically specified for each cryptographic operation;

Message Transformation Here the transformation rules can be statically defined, can be loaded from a repository (if some management functionality is implemented) or they can be dynamically specified for each message that needs to be transformed.

When choosing between these options there plenty of issues to be taken into consideration, such as: how often are the parameters of the security system changing, how much management should the system cope with, how often are the requirements of the protected service changing, are there off-the-shelf components that could be directly used? Ultimately, this is an implementation decision. In chapter 6, I will show how these issues have been dealt with in practice and the motivation behind the decisions taken.

5.4 The Information Viewpoint

According to the RM-ODP [6], the Information Viewpoint defines the semantics of information and the semantics of information processing in an ODP system in terms of configuration of information objects, the behavior of those objects and environment contracts for the system.
Accordingly, in this section the information existing in our security system will be described and analyzed. I will examine the information exchanged between different entities of our security system: the information used in the communication with the service requester and protected service and the information exchanged internally between different security services, including exception messages. Security services also store information internally, for their own use, such as policies, directories used to store identity information, keys, etc. A detailed information model for each service is not within the scope of this dissertation. The reason for this is that, as described in the computational viewpoint, the exact tasks for each security service depends on specific application requirements. Instead, only those general information aspects which are valid for all security services will be described here. More details about the information model for specific security services can be found in chapter 6, where the practical applications of ***SOSA*** are described.

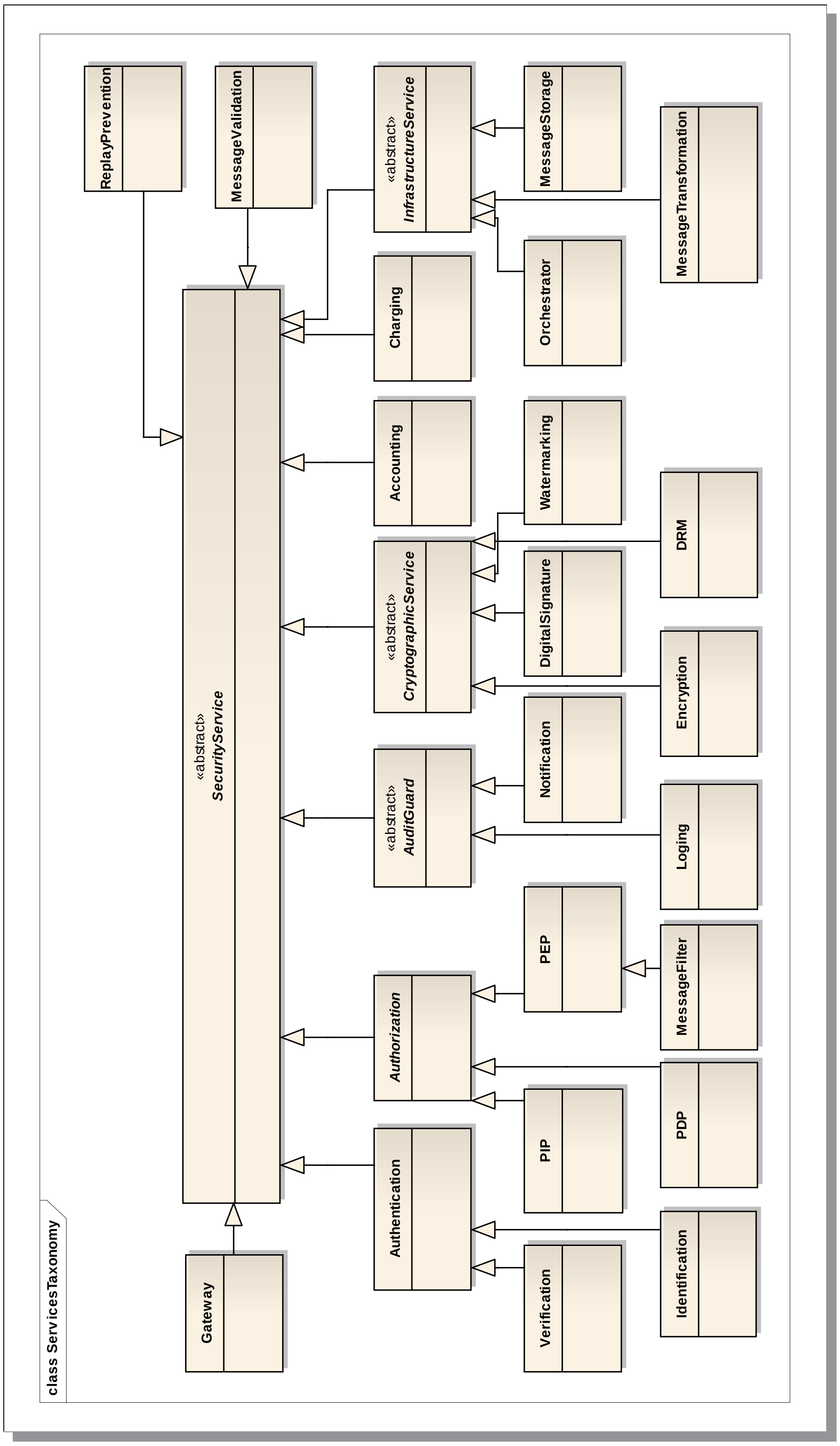

Figure 5.13: Taxonomy of security services

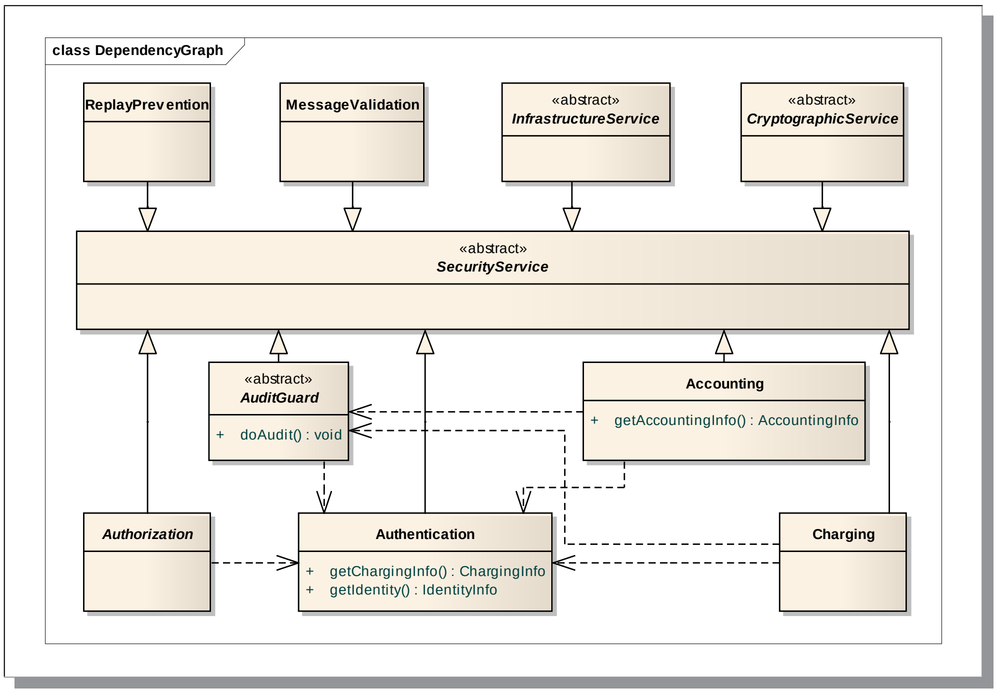
class DependencyGraph
ReplayPrevention
MessageValidation
«abstract» InfrastructureService
«abstract» CryptographicService
«abstract» SecurityService
«abstract» AuditGuard
+ doAudit() : void
Accounting
+ getAccountingInfo() : AccountingInfo
Authorization
Authentication
+ getChargingInfo() : ChargingInfo
+ getIdentity() : IdentityInfo
Charging

Figure 5.14: Dependency graph for security services

5.4.1 Messages

A message is the basic unit of data sent from one entity to another; entities are the service requesters, the security services and the service providers. From the security system's point of view, we can differentiate between:

Internal Messages Messages exchanged within the security system, between security services;

External Messages Messages that are not internal to the security system, that is, messages that either originate from the service requester or messages that are sent to the protected service.

As described in the enterprise viewpoint, the purpose of the security system is to enforce the security policy on messages addressed to / originating from a protected service - these are *external messages*. The security system is (as described in the computational viewpoint) composed from several different security services that communicate among them by means of *internal messages*.
In the following both internal and external messages are described. External messages are requests to the protected service or responses from this one. For this reason, the model described in W3C's Web Services Architecture [49] is used. Internal messages are built upon the same model, leveraging the header mechanism.

External Messages

An external message is the basic unit of data exchanged outside the security system. The main parts of an external message are the message header and the message body. The header is usually built in a modular fashion and contains information that is not specific to the application and is mainly intended for the processing of the message. The body of the message contains application specific information.
Another important feature of messages is the addressing information. Each message has exactly

one message sender and one or more message recipients. The addressing information is embedded in the message in well-known locations so that transport services are able to locate it in order to deliver the message.
External messages can be as easy as an HTTP GET request, in which the HTTP headers are the header and the URL encoded parameters are the body. The addressing information is expressed by the URL and information found in the TCP/IP protocol. However, more advanced messaging systems such as SOAP or JMS would fit better the requirements of the security system described here, because they provide richer support for processing among multiple messaging agents.

Internal Messages

An internal messages is the basic unit of data exchanged between security services. An internal message contains an external message (the message that is being processed - such as a request from the client or a message generated by the protected service) and several annotations.
Because security services are also Web services (in the same way as the protected service), the communication among them is subject to the same requirements as the communication external to the security service: there is a need for addressing information (so that messages get delivered) and for a separation between header and body to facilitate modular processing of the message. Because the requirements are similar, internal messages can be implemented on top of external messages, by adding support for annotations. This is illustrated in the UML diagram 5.15.

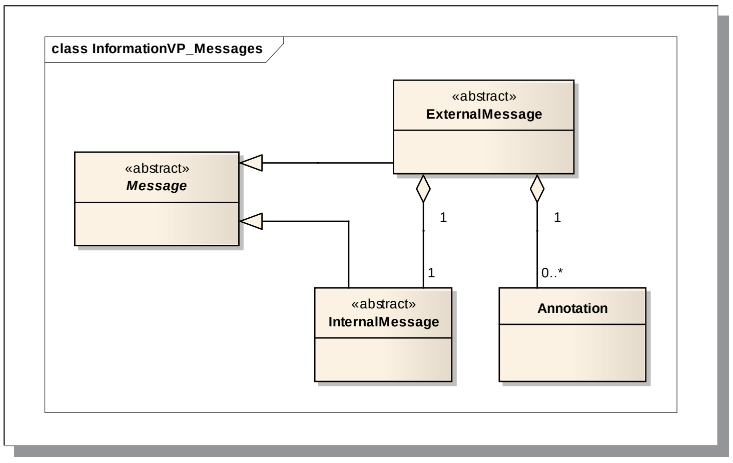

Figure 5.15: Messages

Annotations

An annotation is the basic data structure used by security services to communicate information to one another. As also described in section 4.3 and the computational viewpoint, after processing a message, security services are able to annotate it with the results from the processing routine. In this way, this information remains bound to the message, and can be used by subsequent security services when processing the message. Annotations are described in detail in the next section, where an information model is presented and analyzed.

5.4.2 Information Model for Annotations

Annotations can be seen as assertions: an annotation asserts some fact about something (for example "user john is authorized to borrow books from the library"). These assertions are created by one service and are destined for one or more services that will later process the message. Upon receiving a message, security services read the annotations and can take action depending on the contents of the annotations. In complex systems, where trust relations between security services need to be taken into considerations, certain security services may chose to ignore some annotations or raise exceptions (for example if they do not trust the issuer of certain annotations).

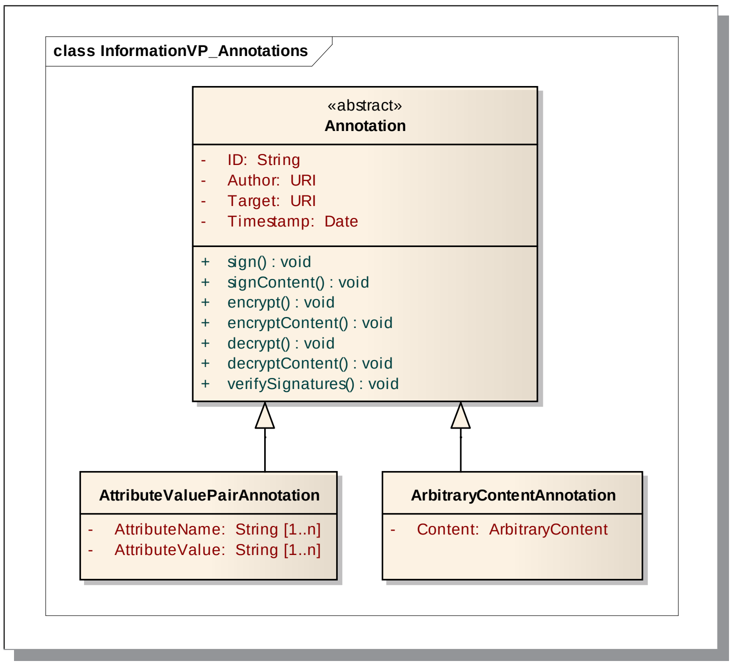
class InformationVP_Annotations
«abstract»
Annotation
- ID: String
- Author: URI
- Target: URI
- Timestamp: Date
+ sign() : void
+ signContent() : void
+ encrypt() : void
+ encryptContent() : void
+ decrypt() : void
+ decryptContent() : void
+ verifySignatures() : void
AttributeValuePairAnnotation
- AttributeName: String [1..n]
- AttributeValue: String [1..n]
ArbitraryContentAnnotation
- Content: ArbitraryContent

Figure 5.16: Annotations

Following the above description, an annotation should contain at least the following information (this is graphically illustrated by the UML diagram in figure 5.16):

Author An information that uniquely identifies the service that created the annotation (there should be a single author for each annotation). This information is important if trust between security services is an issue to be taken into consideration. In this case, security services can use this field to verify the source of the information. In situations where trust between services is not an issue, services can only look at the contents of the annotation without verifying the source of the information.

Target An information that will identify the recipients of the annotation, that is, the service or services that should process the annotation. This information is optional and is designed to simplify the processing of annotations. The target does not uniquely identify a security service, but rather announces the kind of information that the message contains, or the purpose of this information. Therefore services can inspect this field and decide whether the contents of the annotation is relevant to them or not.

Contents This is the assertion itself and its presence is mandatory. It should be possible to allow for different kinds of contents: arbitrary contents, attribute pair, etc. These are further described in the next subsections.

ID A unique identifier for the annotation. This permits the annotation to be referenced from within the message (for example by other annotations or by digital signatures) and allows a more efficient processing (services can easily index annotations).

Timestamp The time of creation. This is useful especially in combination with digital signatures to prevent replay attacks.

Digital Signature In complex systems, where internal messages are processed by not totally trusted services, or in those cases where messages travel over unreliable channels, digital signatures should be added to assertions. In this way, a receiver can verify that the issuer of the annotation is the claimed entity, and therefore that the contents of the annotation can be trusted.

Encryption In the same situations as above, there might be the need to encrypt certain annotations either in whole or only the contents field. Encrypting the whole annotation means encrypting both its contents field as well as the other fields (author, target, id and timestamp). If only the contents of the annotation is encrypted, the other fields will remain in clear text.

As far as the contents field is concerned, two models are proposed: arbitrary-content annotations and attribute-value pair annotations. The first one has no semantics defined and has the advantage of being extensible, while the latter one has strong semantics defined, which gives it the advantage of being simple and easy to use. The two are described in the following two subsections.
Depending on specific application needs and the exact realization (encoding) of annotations, other information may be included in annotations or other annotations types may be defined. For this, the arbitrary-content annotation can be used as a starting point, and new annotation types be derived through either extension or sub-typing.

Arbitrary-Content Annotations

In the case of an arbitrary-content annotation, no structure for the contents is defined and no constrains are placed on the contents. It is up to the service creating the annotation and the services consuming it to decide how the contents is structured, encoded and interpreted. An example of arbitrary-content annotation is an annotation that stores an arbitrary XML block.

Attribute-Value Pair Annotations

An attribute pair annotation stores information of the form attribute = value. For performance and flexibility reasons, it should possible to store several attribute-value pairs in a single annotation.
In comparison with the previous annotation type, here the semantics of the information are well defined: both attribute name and the value must be strings. Attributes can be structured, if a corresponding naming schema is used for the attribute name (i.e. "level1.level2.level3.name").

Arbitrary-Content Annotations vs. Attribute-Value Pair Annotations

Attribute-value pair are semantically rich enough to accommodate annotations produced by most security services. A couple of examples are given below:

1. Identification Service

  - user.identity.department = Human Resources
  - user.billing.creditCard.type = VISA

2. Authorization / Policy Information Point (PIP)
   - resource.accessType = read
3. Authorization / Policy Decision Point (PDP)
   - message.authorized = true
4. Accounting
   - message.accountingUnits = 27

However, the attribute pair annotations do not accommodate certain annotations. One example for this is the case of annotations created by PDP services that contain obligations - a richer semantical model is necessary to encode obligations. For such cases, an application should either use arbitrary contents annotations, or define a new annotation type (through extension / sub-typing).

Comparison with Other Models

It was mentioned earlier that annotations should be seen as assertions, and therefore it is worth discussing the similarities / differences between the annotations proposed here and the assertions defined in SAML [22].
Furthermore, because assertions are designed to store intermediate processing results, they share similarities with the SOAP header blocks, which are designed to allow the processing of a SOAP message by multiple services.

Annotations as Custom SOAP Headers Annotations are intended to be processed by other security services in the same way SOAP headers are intended to be processed by certain nodes along the message path. SOAP headers contain a role[7] attribute, that identifies the SOAP nodes that must process the header block. This field is similar to the target element of annotations.
SOAP headers do not have support for the rest of the annotation information fields (author, id, timestamp). The contents of SOAP headers is an arbitrary XML block, with no structure defined, which is similar to the arbitrary-content annotations. No direct support for digital signatures and encryption is built into SOAP headers, however XML Encryption [160] and XML Digital Signature [159] can be used to implement these features.
In ***SOS!e 1.0*** annotations are implemented as SOAP headers (see the Technology Viewpoint, section 5.6 for details).

Annotations as SAML Assertions SAML assertions have a similar structure with the annotations, in that they also contain information about the issuer, the issuance timestamp and an assertion ID. They also have built-in support for digital signatures, but not for encryption.
Because of the use-cases that lie behind the design of SAML, assertions are always made about a subject; a subject is either a human or a software agent. This is not necessarily the case for annotations which contain security information about the message they are bound to. Annotations may contain assertions about a subject such as for example information regarding the sender of the

[7]In SOAP 1.2, [88]

message, but can also contain information that is not bound to an identity - such as context information which needs to be evaluated by a decision making service. Furthermore, SAML assertions do not have a built in support for the target field.
These shortcomings can be overcome by developing a profile of SAML where custom assertions are defined for annotations. This is possible, and several other specifications take advantage of this mechanism (for example the SAML profile of XACML [18]).
Besides the up-mentioned shortcomings, SAML offers a lot of advantages: SAML attribute-pair annotations can be used to implement attribute-value pair annotations and the combination of WS-Security and SAML allows the binding of assertions to SOAP messages in a way that prevents replay attacks. Through WS-Security, SAML assertions are attached to SOAP messages in special SOAP headers - this feature can be used to leverage the advantages of the previous approach. In ***SOS!e 2.0*** annotations are implemented as SAML assertions (see the Technology Viewpoint, section 5.6 for details).

5.4.3 Information Model for Exceptions

In the computational viewpoint it was mentioned that if the security checks fail or if the processing logic cannot be correctly applied, security services can raise exceptions signaling the errors encountered. Exceptions take the form of messages which are distinctly marked as exception messages so that the middleware system and the other security services can recognize them and treat them appropriately.

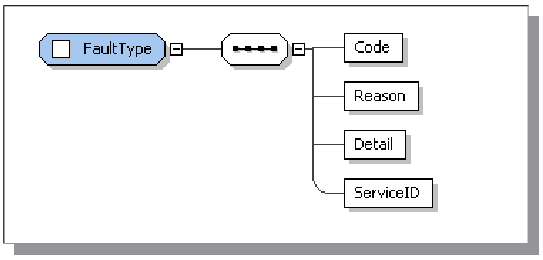

Figure 5.17: Information model for exceptions

An information model similar to the one from the SOAP specification [111, 88, 89] is adopted here. An exception message will contain as payload information detailing the exception (see figure 5.17). The following fields shall be present:

Code (mandatory) An identifier for the error encountered; this field is intended to be processed by a machine;

Reason (mandatory) A textual description of the encountered error; this field is intended to human readable;

Detail (optional) This field carries application specific information regarding the failure;

ServiceID (optional) An identifier (such as the address) of the service that caused the error. This field is important in certain message routing scenarios, if the exception needs to be further processed before being sent back to the service requester.

SOAP further defines a role field which identifies the role the node was operating in at the time the fault occurred. This is defined as optional in SOAP and is not needed here, because the security services do not act on different roles.

5.4.4 Information Model for Service Composition

Web services composition is about proving an approach to connecting several Web services together in order to create a higher-level business-process. In the context of this dissertation, the focus is on assembling together several security services in order to create a security system. Furthermore, the security system is coupled together with one or more protected Web services which are *security unaware*, thus enhancing these services with security capabilities.
Such service compositions need to be described in machine-readable languages. As described in the computational viewpoint, security services are realized as regular Web services. Because of this, existing work in the field of Web service composition can be leveraged for defining service orchestrations, choreographies and message routing rules. Leveraging this work has numerous advantages, including the ability to use existing runtime frameworks, tools and editors (several visual, easy-to-use Web service composition editors are available), and the ability to combine security service compositions with other existing Web services compositions.
In the literature there is a rich amount of information about Web services composition available. [124, 125, 156] present a review of the existing technologies, tools and standards. The two the most important standards for describing Web service compositions (BPEL and WS-CDL) are introduced below. The differences between the two of them are illustrated in figure 5.18, where a simple composition containing three services is illustrated using both approaches.

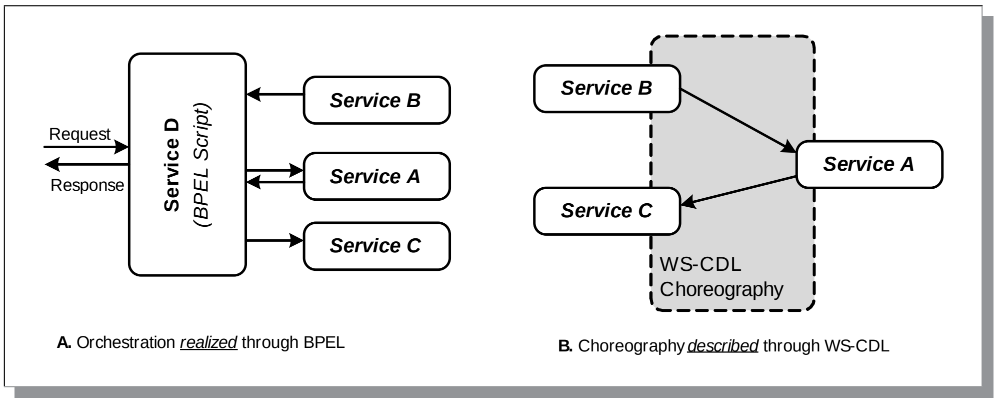

Figure 5.18: BPEL vs. WS-CDL

BPEL4WS [51] Currently a standard from OASIS, BPEL[8] provides an XML grammar to describe the control logic required to coordinate Web services participating in a process flow. BPEL describes how a central entity invokes (or is invoked by) several other Web services - in other words BPEL describes Web service orchestrations. BPEL leverages WSDL: both the services which are invoked and the resulting composed service are described by means of WSDL. As far as the grammar is concerned, BPEL is quite complex allowing conditional branches, loops, parallel as well as sequential executions. Furthermore, BPEL has built-in support for handling transactions and exceptions. Several extensions have been proposed for BPEL, such as BPEL-J [45] - to allow orchestration processes to include code for executing computational work written in Java, and BPEL4People [102] - to allow human people as one possible type of participant in orchestration scripts.

[8]The standard is called BPEL4WS - Business Process Execution Language for Web Services. The terms BPEL and BPEL4WS are used interchangeably throughout this dissertation.

WS-CDL [26] The Web Services Choreography Description Language, which is the successor of WSCI[9] [11], defines an XML-based language for describing peer-to-peer collaborations of participants by defining, from a global viewpoint, their common and complementary observable behavior [26]. As opposed to BPEL which describes orchestrations, WS-CDL describes Web services choreographies. Similar to BPEL, WS-CDL also supports a wide variety of structured activities including sequential and parallel processing, conditional branches, loops and includes mechanisms for business transactions and exception handling. WS-CDL is not an "execution language" (as BPEL is). With WS-CDL Web service choreographies can be described; having this description, it is up to each participant that takes part in the choreography to implement the message exchanges by means of one technology or another (BPEL is one possible choice).

BPEL is a good choice to integrate security services that expose RPC-like interfaces. It is also a good choice for describing and implementing complex processes that are then integrated with the rest by other means (for example message routing techniques). Furthermore, as also described in [56], BPEL can be used to implement the content-based message routing pattern. However, because of its centric view on Web service composition, it is not possible to specify distributed interactions such as the ones that can be implemented by means of itinerary routing.
The disadvantages of BPEL are compensated by WS-CDL. With WS-CDL it is possible to describe all interactions between security services the protected services and the service requesters. However, there are several disadvantages to WS-CDL:

1. WS-CDL currently only has a W3C candidate recommendation status, and not many implementations are available at the time of writing.
2. WS-CDL is useful to describe the choreography as a whole, but these descriptions are not directly executable. Having the WS-CDL description, each participant then needs to map the interactions onto its internal implementation of the choreography.
3. With WS-CDL complex choreographies involving different kinds of messages and different partners playing different roles can be described. For this reason WS-CDL might be too complex for describing the interactions between security services which may be easier modeled by means of message routing.

For these reasons WS-CDL was not chosen when implementing ***SOSA*** . Instead, the more simpler ESB-specific service composition mechanisms have been chosen: service composition through message routing. ESB implementations have built in support for describing message routing rules, including itinerary routers and content-based routers [56, 134]. Some ESB implementations also provide visual tools for creating message routing rules and defining mediations where several services are combined together. These tools can be leveraged by implementations, to allow for a quick and easy definition of the interactions between security services.
One further argument in favor of message routing is that in comparison with BPEL, service aggregation by means of message routing renders a better performance. The most important argument for this is that normally BPEL scripts are persisted to disk during execution in order to be able to implement long asynchronous processes; this is not necessary for message routing, were a send-and-forget strategy can be applied.
Nevertheless, WS-CDL (and BPEL) have applications in the context of ***SOSA*** . Complex security

[9]Web Services Choreography Interface

systems, containing many services and having complex security policies, can be described by means of WS-CDL. This would provide a good way for modeling the system because various graphical tools can be used. Furthermore, automated system-testing can be performed on the WS-CDL representation of the system. In addition to this, specialized tools could be developed that generate runnable representations of the service aggregations (either as message routing rules or as BPEL scripts). These issues have not been further analyzed in this dissertation but are possible a continuation of this work (see 8.4).
In the Technology Viewpoint (section 5.6) two implementations of ***SOSA*** are presented. These implement service composition by means of message routing: ***SOS!e 1.0*** uses a simple XML vocabulary to describe itinerary routing, while ***SOS!e 2.0*** leverages the built-in features of the Mule ESB to implement both itinerary as well as content-based message routing.

5.5 The Engineering Viewpoint

According to the RM-ODP, the Engineering Viewpoint describes the system as a network of computing nodes. If the previous three viewpoints described the purpose, the contents and the functions of the security system, the Engineering Viewpoint will relate these to specific components linked by means of network communication. The main concerns of this viewpoint are communication, computing systems, software processes and the clustering of computational functions at physical nodes of a communications network [14].
The computational nodes in our system are security services. In this section I will describe how these services can be combined, how they communicate with each other and how they can be distributed. At the end of this section I will show a couple of patterns that show how security services can be aggregated to solve common security issues.

5.5.1 Web Services Composition

Web services composition (aggregation) refers to the composition of multiple Web services in a process flow. Such processes are described in terms of exchanged messages, business logic and the order of execution for interactions. They can range from simple ones (such as when one Web service calls another) to very complex ones which span several applications and organizations and result in long lived, transactional, multi-step interactions [125].
As also described in [49, 125], there are two models for aggregating services: choreography and orchestration. The difference between the two is that in orchestration the execution is controlled centrally from a single entity. As opposed to that, choreographies are more collaborative in nature, and each party involved in the process controls a part of the execution.
Because each of the approaches has its advantages, both of them are valid ways for composing security services. Furthermore, one does not exclude the other, so the two approaches can be combined - this is illustrated in figure 5.19.

Choreographies

Are realized by means of message routing. After a security service has processed a message, it will send the message to another service for processing - in figure 5.19 service A sends the message to service B which sends the message to the orchestrator service. The control of the execution is distributed, because each service is responsible for choosing the next service along the chain.

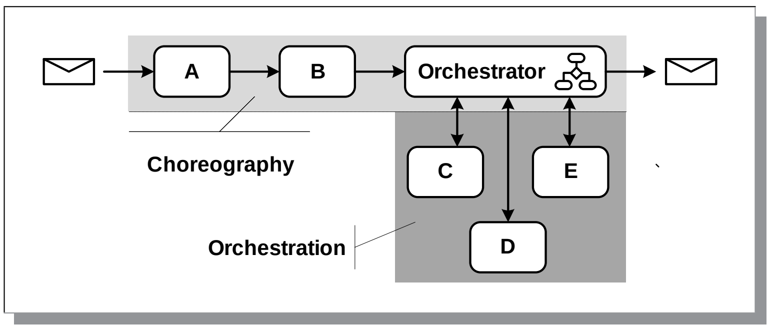

Figure 5.19: Service composition

Orchestrations Are realized by means of orchestration services (described in the Computational Viewpoint) - in figure 5.19, the orchestrator coordinates the activity of services C, D and E. The orchestrated process can be exposed as a single service and can be integrated together with the other services - figure 5.19 shows how the orchestrated process combining services C, D and E is integrated with the other two services (as part of a choreography).

5.5.2 Service Chains

With choreographies realized through message routing in mind, considering a request-reply message exchange, we can define two service chains: forward chain and return chain (figure 5.20). Messages originating from the requester are travelling along the forward chain to the protected service. After processing the request, the service generates a response message which travels along the return chain back to the requester.

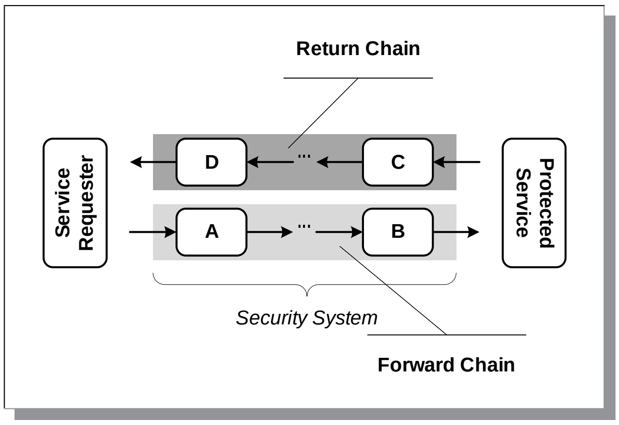

Figure 5.20: Service chains

The forward and return chain may also contain orchestrations which are exposed as services. Furthermore, the split / gather message pattern (described in 2.3) can be applied to execute several tasks in parallel.

5.5.3 Realization of Choreographies

As previously mentioned, the choreographies are realized through message routing. Having the message routing patterns described in section 2.3 in mind, the two patterns that can be used for the realization of choreographies are itinerary routing and content-based routing. The requirements for each of these are described in the next two subsections, together with the advantages and disadvantages.

Itinerary Routing

In itinerary routing a routing slip is attached to the message describing the route that the message should follow. The routing slip fully describes the choreography. In an execution environment, after a service has processed the message, it is the responsibility of the messaging middleware to dispatch the messages according to the routing slip.
In a typical deployment scenario, a gateway service would receive the message from the requester, inspect it and attach a routing slip describing both the forward and the return chains. In this way, the protected service is aggregated together with the security services into one choreography which is centrally described. For more, see the gateway pattern described in section 5.5.7.
The advantages of this approach are that the choreography is centrally described at the gateway service which is the one entity attaching the routing slip. This is very convenient for the management and administration of the whole system.

Content-Based Routing

In the case of content-based routing, after a message is processed by a service, it will be inspected and, depending on its contents, the next service along the chain is determined. In networking, this kind of routing is called next-hop routing, as each node determines the next destination of the message, based on some internal routing table.
As opposed to itinerary routing where the choreography is centrally described and managed, here the choreography is specified through routing tables which are distributed (each router has its local routing table). The choreography is managed in a distributed fashion.

Mixed approaches

Of course mixed approaches are possible. In this case parts of the choreography are centrally described through itineraries, while the rest is specified through routing tables. This is a good way to combine multiple choreographies together or to handle exceptions (see section 5.5.5).

5.5.4 Transactions

Each service implements only a part of the overall functionality. In a similar fashion to database systems, there are be cases in which it is necessary to execute several services as one single logical operation. This is realized by means of transactions.
To make an example of such a situation, let's consider the case of a Web service that involves payment. We assume that the payment model is pay-per-request, so the client must be charged every time he makes a request to the Web service; however, he should not be charged if the request can not be fulfilled. A possible ***SOSA*** implementation could have charging implemented as a security service and deployed on the forward chain as showed in figure 5.21. In this case, it is mandatory that the two services be executed as an atomic operation, otherwise, if the Web service fails, the requester will still be charged.
Atomic operations can be implemented by means of transactions which are supported at the middleware layer. Because security services run on top of the middleware, they can make use of transactions. This dissertation does not further discuss transactions.

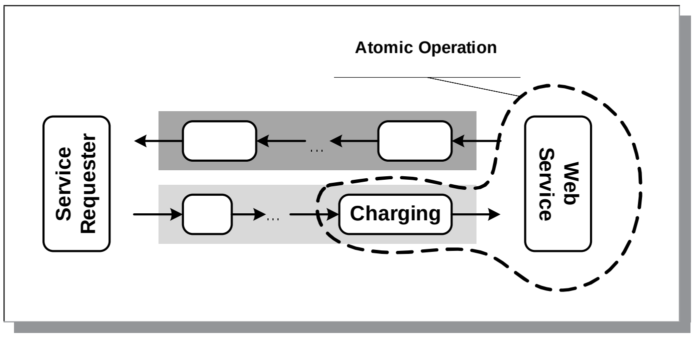

Figure 5.21: Transaction example

Transactions from the middleware perspective are described in [56, 134]. There are also proposals for standardized SOAP specifications for transactions, namely the WS-Coordination framework [27], which describes extension to SOAP messaging for providing protocols that coordinate the action of different applications. The framework contains two specific coordination types: WS-AtomicTransaction for short duration [24], ACID transactions and WS-BusinessActivity for longer running business transactions [25].

5.5.5 Exception Handling

Security services may throw exceptions if they cannot fulfill their tasks. Exceptions, as already described in both information and computational viewpoint, are messages that contain as payload details about the failure (reason, code, description, etc). Because exception messages are distinctly marked, the messaging middleware can take special action. In orchestration scripts (such as BPEL) exception handlers can be declared, defining the actions to be taken in case of failure. In the case of choreographies realized through message routing, content-based routers can be configured to inspect if the result message is an exception message, and in this case, to route the message on a special route.
The simplest strategy for exception handling is to return the exception messages back to the requester. Because they contain information regarding the failure reason and details about the it, the requester can fix the problem and resend its request. However, more complex strategies can be applied such as routing exceptions to a central exception-handling service, that implements exception recovery strategies, releases any resources associated with the request, unrolls any transactions pending and further processes the exception message (through either enrichment or filtering) before sending it back to the service requester.

5.5.6 Service Clustering

Because in ***SOSA*** several services need to cooperatively process one message, the failure of one service results in a system failure. Furthermore, because the security functionality is distributed among services following the principle of separation of concerns and not using the processing load as a main criteria, it is most probable that some services will spend more time processing a message than others. This results in some services being performance bottlenecks for the whole system.
To address these issues, and still profit from the ***SOSA*** model, solutions available for server clustering can be deployed. In such a design, several services are configured to appear as one single logical service (figure 5.22). The technique can be used for enhancing availability, scalability or both.

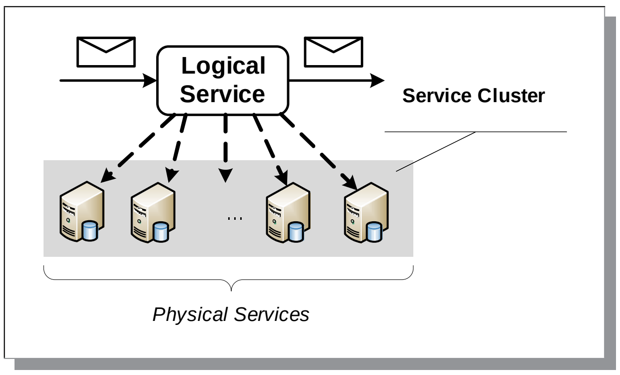

Figure 5.22: Service clustering

Clustering is usually supported both at the networking layer as well as at the middleware layer. The security services can easily take advantage of these features.
The technical aspects involved in clustering are not further discussed here. [145] describes server clustering in the context of Web services in more detail. [56, 2, 100] address clustering in the context of ESB.

5.5.7 Patterns in SOSA

The majority of books addressing Web services architectures, ESB and messaging-based integration such as [56, 95, 145, 94] present their contents in the form of patterns. A pattern[10] describes a recurring problem that occurs in a particular situation and recommends a solution [94]. The solution alone is not enough for transmitting the knowledge to the reader; it must be accompanied by the decisions that must be made, the considerations that go into those decisions and it must discuss possible alternatives [95].
Patterns are a very powerful approach to documenting knowledge in a way that it can be easily understood and readily applied by others. In the following, I will present a couple of patterns related to ***SOSA*** . The patterns address common security problems related to Web services. The purpose of this is on one hand to document how security services can be designed and coupled together in order to build a ***SOSA*** security solution. On the other hand, the purpose is to illustrate the practicability of ***SOSA*** .
The patterns address disparate aspects of security (authentication in federated domains, distributed authorization, session establishment, etc). They show how the security services described in the Computational Viewpoint (section 5.3) can be linked together by means of common messaging patterns - the ones described in section 2.3. Because of the loosely-coupled nature of the system, it is easy to assemble together several patterns and build complex security solutions.
Each pattern has a suggestive *name* and is described in terms of the *context*, *problem*, *solutions*, *variations*, *benefits*, *considerations* and *related patterns*[11]. In addition to this, the relation to approaches taken by others to solving the same problem is discussed where appropriate.

[10]Christopher Alexander notes that *each pattern describes a problem which occurs over and over again in our environment, and then describes the core of the solution to that problem, in such a way that you can use this solution a million times over, without ever doing it the same way twice.* [78]
[11]Similar approaches to describing patterns are taken by [95, 94].

The Gateway Pattern

Context External applications require access to one or more Web services which are deployed inside the private network. The access to these services is restricted to authenticated users. External applications should not be able to access or determine the existence of services and other resources deployed inside the private network. However, it should be possible for services inside the private network to have access to transport level security information (IP address, SSL authentication, etc.) which is sent by the service requester together with his requests.

Problem How to make Web services in the private network available to external applications, without exposing the other resources?

Solution The solution (figure 5.23 A) involves a Gateway service (see section 5.3) located at the perimeter of the network, in a demilitarized zone. This pattern builds on the perimeter security idea, specifically on the Perimeter Security Router described in section 3.3.

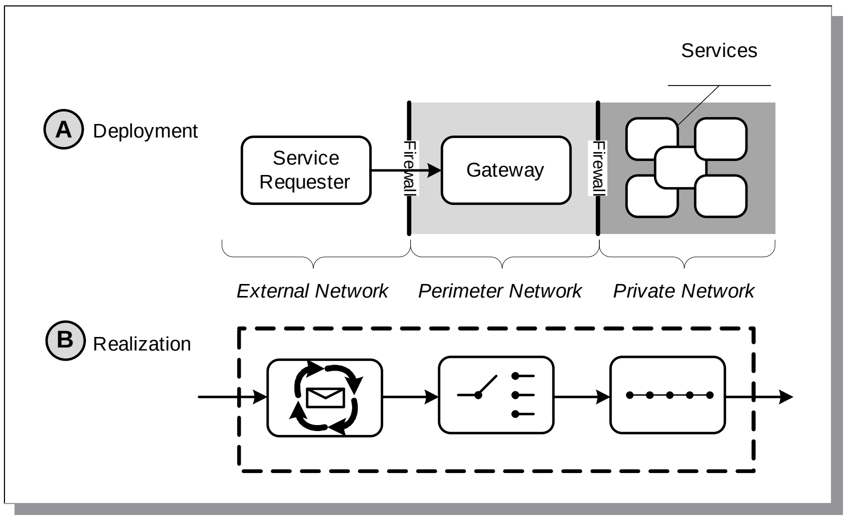

Figure 5.23: Gateway pattern

The Gateway Service is the only entity visible from the external network; the protected service together with the rest of the security services are hidden from applications running in the external network. The tasks of the Gateway service are as follows:

- Transform the message into the canonical message format used by the security services (with support for annotations).
- Extract and, if possible, also verify security relevant information provided by the transport protocol. This step is required because the Gateway service is an intermediary between the service requester and the other services, and as such, it interrupts the connection at the transport layer (i.e. IP, SSL, HTTP, etc).
- Determine the forward and return chains for the given request message. This involves the following steps:
  - If several services are exposed as a single endpoint, or if one service has several operations with possibly different security requirements, the message content must be inspected in order to determine the message itinerary;

  - Append the itinerary to the message and route the message to the first service on the forward chain.

- If the security system is implemented using asynchronous messaging and the protected service requires a request-response message exchange pattern, the Gateway must correlate the request messages with the responses and deliver the response to the requesting party. This follows the *Smart Proxy Pattern* described in [95].

Considerations

- If message validation and / or replay prevention are not implemented as separate security services, the corresponding checks should be implemented by the Gateway service.
- Figure 5.23 B, shows how the Gateway service uses three ESB patterns: message transformer - to transform the message into the canonical message format, content-based router - to determine the itinerary according to the message type and itinerary-based router to route the message on the forward and return chains.

Benefits The major benefits of implementing the Gateway Pattern are the following:

Concentrate Security Administration The rules describing which security services need to be invoked for which type of requests are centrally manged by the Gateway service. The Gateway Service acts as a single point of administration for the security policy.

Multiple Transport Protocols As also remarked in [92], the Gateway service can be implemented so that it provides the same service on a variety of transport protocols. By this, the Web service logic and the security implementation are decoupled from the transport protocol on which the Web service is being offered.

Related Approaches A similar pattern is discussed in [92, 94]. However there, the Gateway service has more responsibilities: it also involves authentication and authorization. Here, the Gateway service relies on the other security services to perform these tasks. [92] additionally describes a *Reverse-Proxy* pattern, which is equivalent to a Perimeter Security Router, but also implementing authentication and authorization. [95] describes a *Smart Proxy* pattern which is a service that correlates request messages with the corresponding response messages.

The Federation Pattern

Context A protected Web service must accept users which are registered in several security domains. Each domain contains a user management system which stores attributes about the registered users. These attributes are required during the security process (for example one of the attributes contains access permissions necessary for the authorization process).
External applications send requests to the protected service that contain some kind of authentication credentials. However, from the credentials, it is not possible to know the domain where the requester is registered ([33] describes this situation as *implicit federation*, because federation is hidden from the requester). One illustrative example is a situation where all users are identified by username and no mapping exists between the username and the registration domain.

Problem How to retrieve the attributes of the requester from the home domain?

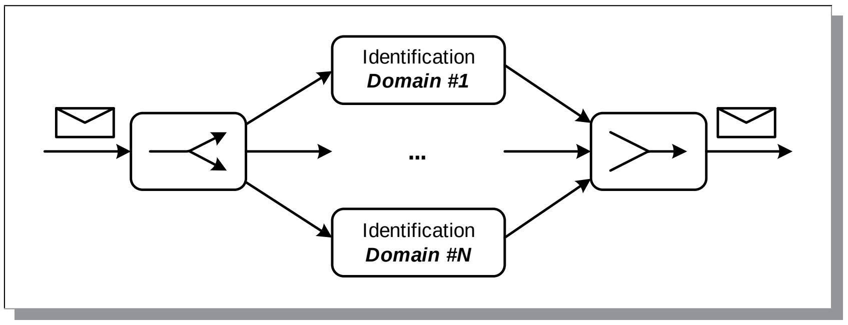

Figure 5.24: Federation pattern - without knowing user's home domain

Solution A service wrapper is built around the local user management system in each domain. This wrapper service resembles the identification service described in 5.3. Its tasks are to identify users based on the credentials provided in the message, to convert their identity to a canonical identity (which is understood by the following services along the chain) and to annotate the message with the attributes found in the user management system (all or a predefined list of them).
Because the user's home domain is not known, all identification services must be invoked. In order to optimize this operation, the splitter-aggregator pattern is used (see figure 5.24). A splitter broadcasts the message to all identification services. The execution of these services is done in parallel, thus optimizing execution time and minimizing failure. We consider the requesters identity to be unique, therefore at most one identification service will execute successfully. An aggregator service then joins the messages together by simply dropping the messages containing failures.

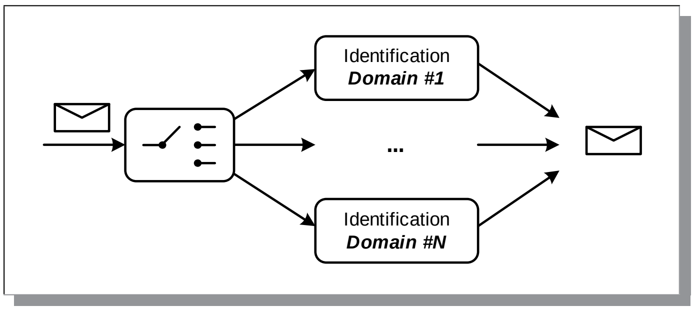

Figure 5.25: Federation pattern - knowing user's home domain

Variations The following four represent variations of the context described above:

Explicit Federation In the case of an *explicit federation*, where it is possible to determine the user's home domain from the provided credentials (the domain is specified besides the authentication credentials or there is some kind of mapping between the authentication credentials and the home domain), a content-based router can be used instead of the splitter (see figure 5.25). The router determines the home domain and routes the message to the associated identification service. Because the message is not split (as in the previous case), there is no need for an aggregator service.

Orchestrator Another variation involves aggregating the identification services by means of an orchestration service (figure 5.26). If the home domain can not be determined, the orchestration will call all identification services in parallel. Otherwise, the orchestrator will first determine the home domain, and then only call the corresponding identification service.

Verification Depending on the type of credentials the user provides, it may also be necessary to verify these credentials in the home domain. In this case, each domain must provide two security services (verification and identification) which are linked together or, alternatively, a single service that implements both tasks.

Distributed Profile Another variation of this pattern is where the user is registered in several domains. The user management systems associated with each domain, contains a user profile storing only some attributes (and not the whole profile). In this case the aggregator will combine the user attributes found in each message received. The services following along the chain will receive the complete user profile.

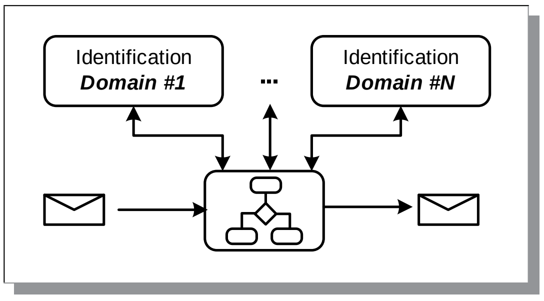

Figure 5.26: Federation pattern - using an orchestration service

Benefits The major benefits of implementing the Federation Pattern are the following:

- Several domains are aggregated together in a manner totally transparent for the user.
- The solution is optimized in terms of time (execution is parallel) and fault tolerance (if the identification service of one domain is down, the users in the other domains are not affected).
- The identification services are built as wrappers around the existing infrastructure. This technique provides a middleware layer between the user repository and the security system. It is thus possible to integrate different user management repositories (like LDAP, proprietary database, etc.) in a single system.
- Several authentication methods (password, digital signature, etc.) can be supported by delegating the verification process to different verification services.
- The identification service receives the whole message, including the service request and the itinerary of the message. It is possible to build identification services that inspect the service request and the itinerary of the message and, based on this information, respond with different attributes or attribute values. Such services are able to enforce privacy on behalf of their users, serving a similar purpose with the pseudonym services described in [33].

Related Approaches WS-Federation [33] also addresses federation in the context of Web services. The specification defines attribute and pseudonym services that can be queried through defined interfaces. These services serve similar purposes with the identification services described here. Security Token Services (defined in WS-Trust [40]) are used to verify the credentials. All these types of services could be integrated by means of an orchestrator, as described in the Orchestrator variation above.

Sequential Decision Making Pattern

Context The decision as to weather or not a given Web service can be accessed in a certain context can not be taken in a single place or by a single entity. One justification for this could be the fact that the access permissions are not all stored in a single place, and in order to grant the access, the complete set of permissions is required. Another scenario could be one where the resource is under the authority of several security domains and, in order to take the access decision, the cooperation of several entities is required.

Problem How to have messages authorized by multiple entities?

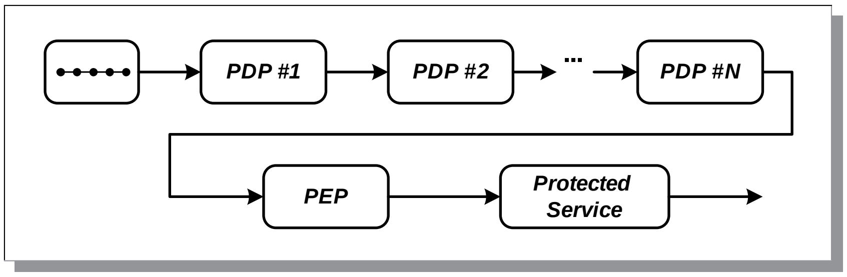

Figure 5.27: Sequential decision making pattern

Solution Several Policy Decision Point (PDP) services are deployed, one for each place where messages need to be authorized. A single Policy Enforcement Point (PEP) is deployed. Because the enforcement should be as close as possible to the resource being protected, this one will be located right in front of the Protected Service (see figure 5.27).
The PDP services, the PEP and the Protected Service are chained together by means of itinerary routing. Messages are routed such that they sequentially pass through each of the PDP services. The PDPs inspect each message they receive and annotate it with their decision and possibly also with obligations. Because messages sequentially visit each PDP, a PDP in the middle of the chain will be able to "see" the decision of the services that previously processed the message and will be able to use this information when making their own decision.
It is the task of the PEP to combine all decisions. If the combined access decision is "access granted" and all the obligations are either satisfied or they can be discharged, only then the message is forwarded to the Protected Service.

Example To illustrate this scenario, consider the situation where PDP#1 annotates a given message with "access granted, but only if (1) PDP#2 agrees and (2) an audit trace is stored". PDP#2 annotates the message with "access granted" and then the message passes through an audit service which annotates it with "audit trace stored". In this case, the combined access decision is "access granted" and, because the two obligations are satisfied, the PEP should forward the message to the Protected Service.

Benefits The major benefits of deploying the Sequential Decision Making Pattern are the following:

- Several entities can collaborate in taking an access control decision;
- Enforcement is done in a single place, close to the resource;

- Through obligations, requirements can be specified. These requirements can refer to decisions taken by other PDPs or the action of other security services.

Variations The following is a variation of the context above:

QoS Negotiation Instead of taking a boolean decision as to whether or not the access is granted, the PDPs can negotiate quality-of-service parameters for the Web service. In this case, each PDP annotates the service with certain values of the QoS parameters (in doing this, the values set by the services that previously processed the message may be taken into consideration). The duty of the PEP is to combine the values of all PDP services. One application scenario for this pattern variation is the bandwidth broker example described in the Enterprise Viewpoint (section 5.2) and [148].

Related Approaches [92] describes chaining of different security services named *Security Enforcement Services* (SES). These services have common features with the security services described in this dissertation, handling encryption/decryption, digital signing / validation, authentication or enforcement. However, the decision services are not combined by means of chaining; instead these are invoked by the SESs.

Related Patterns Linking several security services together, can result in delays that might be unacceptable if several messages are exchanged between the service requester and the protected service as part of a conversation. The *Session Establishment Pattern* addresses this issue.
The *Asynchronous Authorization Pattern* shows how asynchronous decision processes - like for example when human intervention is required - can be integrated in a security system. Such asynchronous decisions makers can be sequentially chained together with other decision makers.

Session Establishment Pattern

Context A Web service is protected using a ***SOSA*** -like security system. The access control system is quite complex and involves several authentication and authorization services which are connected by means of message routing and / or orchestration services. One such example was already presented in the sequential decision making pattern.
Application clients accessing the protected Web service normally make several consecutive requests. The complexity of the security system results in long round-trip times for requests and low throughput values for the service. This seriously affects the performance of the client application.

Problem How to improve round-trip time and throughput if the access control system can not be simplified?

Solution The solution involves the deployment of a Session Management Service (SMS) which is responsible for creating sessions (possibly also for destroying them). The service requester must first invoke a `GetSession` operation on the SMS. This results in the request being routed through the complex authentication and authorization process (see figure 5.28). At the end of this process, the SMS creates a session which is deposited in a persistent repository. At the same time, the SMS returns a session token to the service requester. The session represents an established security context. The session token is a proof that the session has been established and thus, a proof that the process of authentication and authorization was successful and that certain permissions have

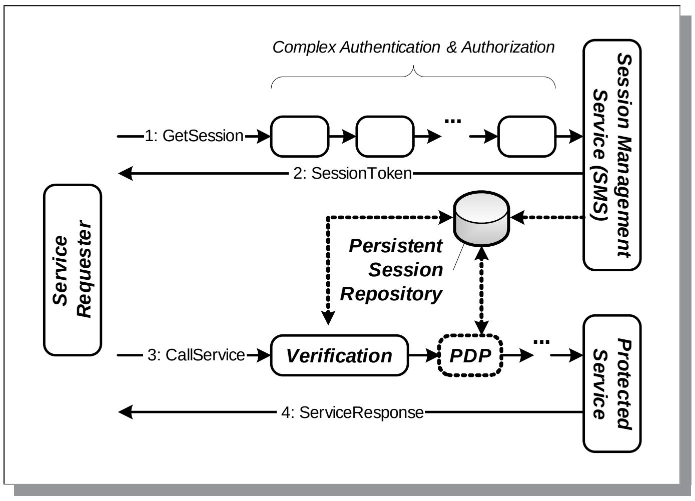

Figure 5.28: Session establishment pattern

been granted.
Having the session token, the service requester accesses the protected service, sending the session token together with his request message. This request bypasses the complex access control system, being routed instead through only one Verification service, which verifies the validity of the session token. From this perspective, the token can be regarded as a bypass ticket for certain security services.
Figure 5.28 shows that the verification service may connect to the session repository in order to verify the token. Weather or not this is required depends on the kind of session token - for example if the token is digitally signed by the SMS its validity can be determined without connecting to the repository.

Benefits The major benefits of deploying the Session Establishment Pattern are the following:

- The number of security services that are invoked for a given message are reduced to a single one;
- The round-trip time per request is lowered while throughput is increased.

Considerations

- This pattern is not applicable if the authorization context changes throughout the lifetime of the session token. In this case, requests to the protected service need to be authorized every time.
- Session tokens must be designed so that they eventually expire. In addition to `GetSession`, a `DestroySession` operation may also be offered. Service requesters will call this operation in order to cancel a given token and instruct the SMS to clear any related information from the persistent repository. This is equivalent to a logout operation.

Variations The following three represent variations of the solution presented above:

With PDP An optional PDP service can be chained after the Verification service. Assuming that the session context contains a list of allowed actions / permissions, the PDP will verify

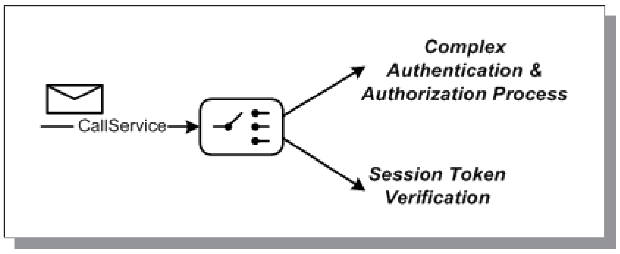
CallService
Complex Authentication & Authorization Process
Session Token Verification

Figure 5.29: Session establishment pattern - the CBR variation

that the request conforms to the permissions associated with the session token. Depending on where these permissions are stored (token or repository), the PDP may also query the session repository.

With CBR Both access ways - the complex authorization and authentication process and the session based authorization - can be offered at the same time. For this, a content-based router is introduced along the `CallService` path. The messages containing session tokens are routed through the ”express-way”, while the others are routed through the complex process. This is depicted in figure 5.29.

Transparent Session Establishment In the presented solution, the session is explicitly initiated by the service requester. A variation of this solution is one where the service is the one initiating the session. In this case, after making a request which is routed through the complex process, the service returns the session token together with the response. This variation simplifies the service definition, as the `GetSession` operation is no longer needed.

Related Solutions Session establishment is common practice. The most basic example are HTTP cookies, which can also be leveraged by Web services if the transport protocol is HTTP. A much more powerful mechanism that is also independent of the transport protocol is specified by WS-SecureConversation [28]. This specification defines the contents and the syntax for a special type of tokens - *Security Context Tokens* (SCTs) - as well as ways to create and exchange such tokens through mechanisms defined in WS-Trust. The WS-SecureConversation model is compatible with the pattern described here, as session tokens can be implemented through SCTs, while the SMS can expose a WS-Trust interface.
WS-SecureConversation further describes token negotiation mechanisms as well as key derivation algorithms for SCT. These can be leveraged by implementations of this pattern.

Related Patterns This pattern can be used to optimize situations where the authentication and authorization processes are complex. Such examples are the *Federation Pattern* and the *Sequential Decision Making Pattern.*

Asynchronous Authorization Pattern

Context The decision whether or not a certain request is accepted or denied is asynchronous by nature. Decisions can not be taken immediately and the response for a decision request can be delayed for very long periods of time. One possible scenario is where every request needs to be approved by a human person (for example, whether or not an email tagged as spam should be posted on a mailing list).

Problem How to integrate decision processes that require very long periods of time in the security system?

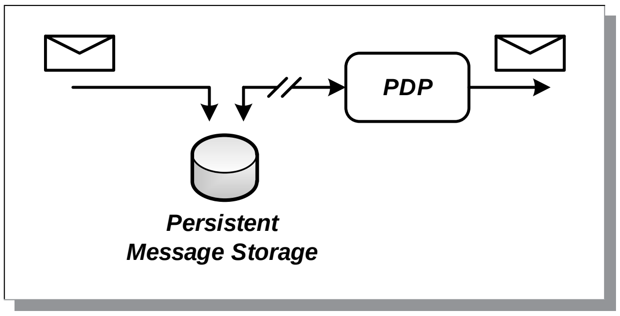

Figure 5.30: Asynchronous authorization pattern

Solution Because the decision processes require very long periods of time to finish, it is impractical to queue the messages at the decision maker. The queue might get full, resulting in messages being dropped.
It is therefore necessary to persistently store messages until the decision maker is able to process them. For this, a message storage service is deployed before the decision maker (see figure 5.30). Both push and pull models can be used. In the push model, the message storage service will constantly try to send messages to the PDP service, until the latter one accepts them. In the pull model, the PDP service connects to the message storage requesting new messages.

Considerations

- In most of the cases, services can be aggregated by means of both orchestration and message routing. In this particular case, because of the asynchronous nature of the communication, aggregation by means of an orchestrator is not feasible.
- Some middleware systems support reliable messaging. This allows the sender to specify delivery assurances which are then met by the middleware. If reliable messaging is used, then there is no need for a message storage service, because the middleware handles message delivery. However, reliable messaging is supported through persistent message stores which assure that in the case of a system failure, messages are not lost [134].

Security Off-Shoring Pattern

Context A Web service is deployed within the internal network of Company A. For certain reasons, not all security functions can be performed within the network of Company A. One motivation for this could be that Company A has decided to off shore some of the security functions to a partner organization - Company B.
In this scenario, the security system is not all located within one organizational boundary. Messages must exit the private network of Company A and enter Company B's network. So that not everybody can use their services, company B may require messages to be authenticated (a proof that they come from Company A should be presented). Furthermore, Company B may perhaps also use some internal authorization mechanisms.
Because of privacy regulations, Company A may be required not to release certain information in clear text outside their network perimeter.

Problem How to integrate security services located in a partner network while respecting the policies of both networks?

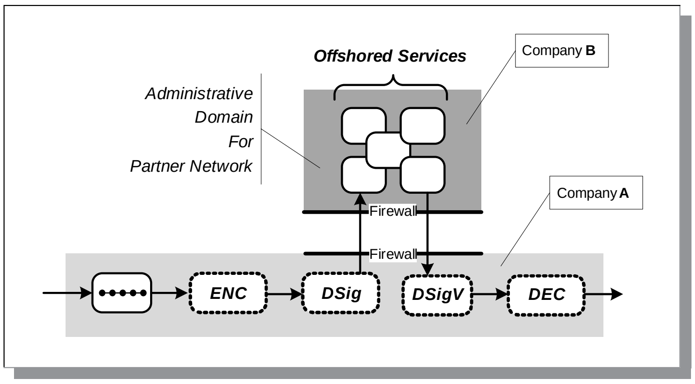

Figure 5.31: Security off-shoring pattern

Solution The solution involves routing the messages such that they exit the network of Company A, enter the network of Company B where the off shored security tasks are performed and then comes back in the network of Company A (figure 5.31).
Because messages may contain information which is not to be released outside the network perimeter, an encryption service (`ENC`) may be introduced on the message route, before exiting the network of Company A. Its task is to encrypt the sensitive information. A corresponding decryption service is introduced on the message path, after the message returns from the partner's network.
Messages may also be signed with Company A's key in order to be authenticated by Company B. For this, messages are routed through a `DSig` service which computes and appends digital signatures. Besides digital signature, other authentication methods (i.e. password, etc.) can be used as well. The message should then be routed through a service which adds the corresponding authentication information.
If the partner is not totally trusted, or if the message travels over untrusted transport protocols, additional signatures can be added to the message. The signatures ensure that certain parts of the message are not altered (the message can not be signed as a whole, because the services hosted by the partner need to modify the message). When the message returns from the partner's network, the validity of the signatures will be verified.

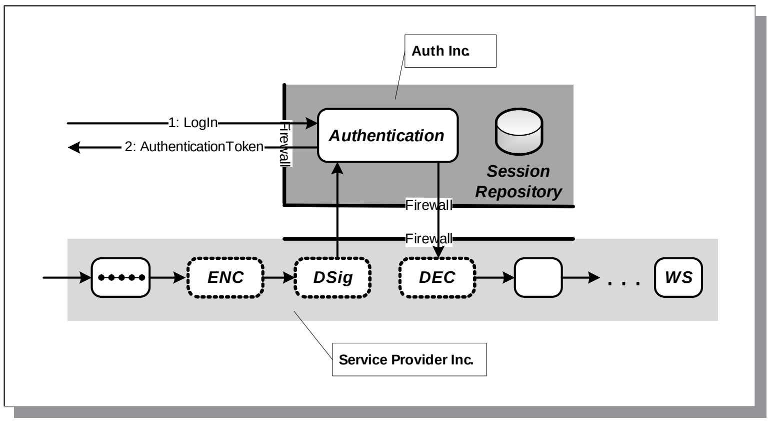

Figure 5.32: SSO with off-shoring of the authentication service

Example Figure 5.32 shows how Single Sign On can be implemented by off shoring the authentication service. In this case, *Auth Inc.* operates an authentication service which exposes a `LogIn` operation. *Auth Inc.* may also host a profile for each user which contains attributes and pseudonyms, together with privacy rules (i.e. which attributes may be released to which service or for which purpose).
A service requester, will first call the `LogIn` operation, receive an authentication token in response, and use this one when calling the `WS` service provided by *Service Provider Inc.* Messages addressed to this service are routed through the Authentication service which verifies the authentication token and annotates the message with the user profile. As previously explained, messages may be encrypted or signed before being sent to *Auth Inc.*

Benefits The following are benefits of implementing the Security Off Shoring Pattern:

- Security functions can be implemented in different networks which are governed by different policies
- Privacy can be achieved through encryption, while integrity can be achieved through digital signatures
- The partner can be fully trusted or only partially trusted. Through digital signatures it is possible to ensure integrity of certain parts of the message.

Considerations

- The partner must always be trusted to perform the tasks it is assigned. It is normally not possible to verify that these tasks were performed correctly.
- The partner may not always be trusted in other matters besides performing its tasks. For these reasons, digital signatures and encryption may be used in order to assure integrity and enforce confidentiality.

Service Sharing Pattern

Context A security system shall be designed which must protect not only one, but several Web services. All these Web services have certain security aspects in common. For both administrative reasons (simplicity, higher understandability) as well as resource allocation reasons (better resource allocation) it would be more profitable to have these security aspects implemented centrally for all Web services.
Any of the security aspects described in section 2.4.2 are subject to being centrally implemented. Figure 5.33 shows several protected Web services sharing authentication. Centralized authentication brings many advantages: on one hand side it allows the clients to use a single identity to access all services (single-sign-on); on the other hand side it makes it easier to manage permissions.

Problem How to implement and enforce certain security aspects centrally for different protected Web services?

Solution The solution assumes that the *Gateway Pattern* is used. Each of the protected Web services is deployed in the private network while a corresponding service proxy is deployed in the perimeter network (figure 5.33). As described in the gateway pattern, the proxy is responsible

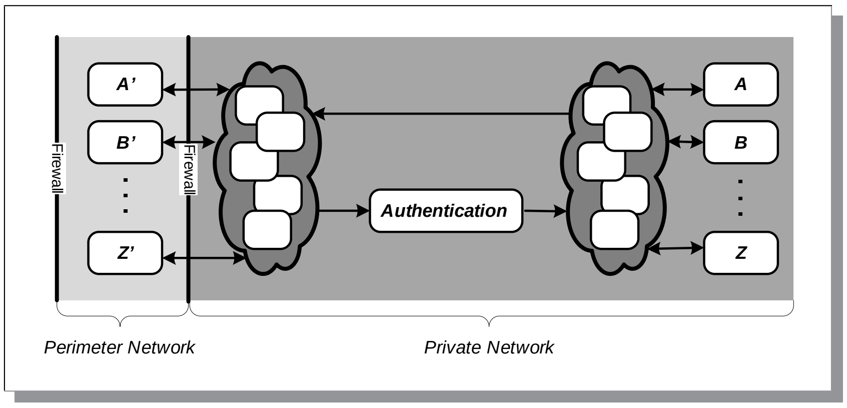

Figure 5.33: Service sharing pattern - centralized authentication

for determining both forward and return chains. In order to have several Web services share one security service, the proxies will have to be configured such that they route messages through the desired security service.
In figure 5.33, proxy services `A'` to `Z'` all include the Authentication service on the forward chain.

Benefits The benefits of implementing the Service Sharing Pattern are the following:

- The overall management overhead is lowered because services are not replicated.
- A better allocation of resources is achieved. Processor intensive tasks can be implemented as services and deployed on more powerful machines. These services need not be replicated for every protected service.

Related Approaches XML firewalls (described in section 3.3) centrally implement different security functions for all Web services inside the private network. However, they tend to concentrate them in a single physical and logical location, on the perimeter network. In the pattern above, no assumption is made regarding the location of the shared services (they can be located in both the private network and the perimeter network) or their number (several services can be shared).

5.6 The Technology Viewpoint

According to the RM-ODP [7], the Technology Viewpoint focuses on the technology aspects related to the system and its environment. It describes the hardware and software components used in the distributed system together with the infrastructure which allows the distributed components to communicate [14]. [129] states that the technology viewpoint should describe the implementation of the system together with information required for testing.
Because ***SOSA*** is a security system for Web services (these have been designed to address a multitude of platforms and implementations) and because it is an architectural design (providing a higher level of abstraction), several implementations are possible. The purpose of the Technology Viewpoint is therefore not to present the implementation of the system, but rather to exemplify how such a system can be built and to show how the most important aspects regarding ***SOSA*** can be realized by means of Web services technologies. The aspects discussed here concern the realization of the system as a whole and not the implementation of specific security services; concrete implementation of security services and the deployment of the framework in real-world scenarios is

showed in chapter 6.
In the following, two implementations of ***SOSA*** will be introduced. The implementations are named ***SOS!e*** (SOSIE)[12]: **S**ervice **O**riented **S**ecurity - an **I**mplementation **E**xperiment. Both implementations are designed for SOAP Web services and take advantage of the SOAP processing model and the SOAP extensions.
SOS!e 1.0 was the first prototype implementation developed; it is based on AXIS 1.2 and other open-source libraries. Later, after analyzing the shortcomings of this implementation, a better implementation was developed - ***SOS!e 2.0*** ; this is more aligned with standards and also takes advantage of the ESB model.
The aspects presented for each implementation concern the realization of choreographies and orchestrations, service definition and communication through annotated messages. In order to show how the framework is deployed and how the security services are connected, a wiring diagram is presented and the most important services together with the required infrastructure are described.

5.6.1 Standards

There are several possible realizations for SOA, several possible realizations for messaging systems and also several possible realizations for Web services (these have been discussed in chapter 2). However, the concepts described in the previous four viewpoints have been only implemented for SOAP Web services and related technologies. An implementation for SOAP messaging was considered desirable as it offers interoperability with other platforms.
The following lists the most important standards that are leveraged by the implementations together with a brief description of how they are used:

SOAP All messages, including the communication with the service requester, the service provider and the communication internal to the security system are SOAP messages. Furthermore, message routing is implemented based on the processing model defined by SOAP (both ***SOS!e*** 1.0 and 2.0).

WS-Security Security tokens are encoded and attached to messages by means of WS-Security. Furthermore, the same specification is leveraged for encrypting parts of messages and attaching digital signatures (both ***SOS!e*** 1.0 and 2.0). In ***SOS!e 2.0*** , SAML security tokens are used as annotations; these are encoded and attached to SOAP messages by means of WS-Security.

BPEL Service orchestrations are described in BPEL and executed through BPEL runtime-engines (both ***SOS!e 1.0*** and ***SOS!e 2.0***).

SAML Message annotations are realized by means of SAML assertions (only in ***SOS!e 2.0***).

WS-Addressing This specification provides enhances SOAP messaging with addressing capabilities that are independent of the transport protocol. These are leveraged by the ESB middleware in ***SOS!e 2.0*** to deliver messages to services accessed through different transport protocols.

5.6.2 SOS!e 1.0

SOS!e 1.0 was the first implementation of the concepts described in this dissertation. It was developed as part of several diploma thesis [128, 75, 90] and student assignments [74, 81, 144]

[12]In French, *sosie* means a person who physically resembles another person to an extent that it is hard to distinguish the two of them. Besides being an acronym, the name was chosen because the implementation follows the gateway pattern described in the engineering viewpoint (section 5.5): a proxy service is deployed in the perimeter network which exposes the same interface as the protected service. A service requester located in the public network will only see the *sosie* of the protected service and not the protected service itself.

and was applied in several projects (OWS4 [68], OWS3 [105]) during the course of which it was improved and extended with new services. For detailed information regarding the implementation please refer to the bibliographic citations.

Software Platform, Libraries and Tools

The security framework was implemented in Java using open-source software. Apache AXIS 1.x was chosen as Web services container as it was the state-of-the-art implementation at the time. Subsequent versions of this software were used: 1.2, 1.3 and finally 1.4. This had several drawbacks which are discussed at the end of this section, among them the fact that AXIS 1.x has limited support for asynchronous messaging.
Several other open-source libraries are also used by the implementation, the most important to mention being the following: *Xerces 1.4.4* parser (`http://xerces.apache.org/xerces-j`), *Xalan 2.7.0* for XPath evaluations (`http://xml.apache.org/xalan-j`), *WSS4J 1.5.1* as the WS-Security implementation (`http://ws.apache.org/wss4j`), *OpenSAML 1.1* as the SAML implementation (`http://www.opensaml.org`), *Log4J 1.2.9* and *Commons-HttpClient 3.0*.

Wiring Diagram

Figure 5.34 shows the most important services in the implementation together with the communication paths between them. The protected service is located in the private network and is not visible to service requesters. The only endpoint that the requesters are able to access is the Front-Side Proxy, which is a service that exposes the same interface as the protected service and follows the gateway pattern presented earlier in the Engineering Viewpoint.

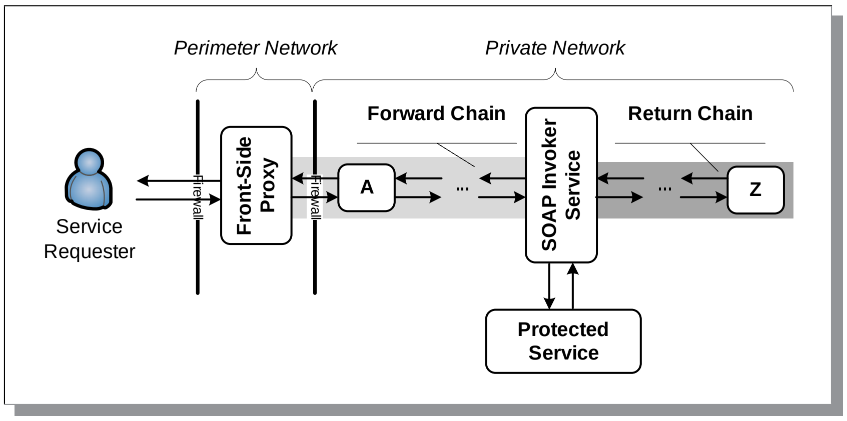

Figure 5.34: ***SOS!e 1.0*** wiring diagram

The task of the Front-Side Proxy is to determine the itinerary for the request to follow and attach an itinerary slip to the message. The request is then forwarded by the Front-Side Proxy to the first security service along the forward chain - in figure 5.34, service A. Each of the security services then implements itinerary routing, and thus the request travels according to the itinerary slip.
The message eventually reaches a special service called SOAPInvoker. This service invokes the protected service and then passes the response further along the return chain. The response then travels along the return chain, until it reaches the last service - in figure 5.34, service Z.

Remarks

- Because AXIS 1 has no support for asynchronous messaging, services in ***SOS!e 1.0*** follow the request-reply pattern. Each service calls the next one along the chain and then waits for the answer to come.

Security Services Realization and Deployment

The implementation provides a middleware layer upon which the security services can be built. This is depicted in figure 5.35 where it is showed how the security services are running inside the ***SOS!e*** middleware, which is deployed as a Web service inside Apache AXIS, which instead runs as a servlet inside the Tomcat servlet container.

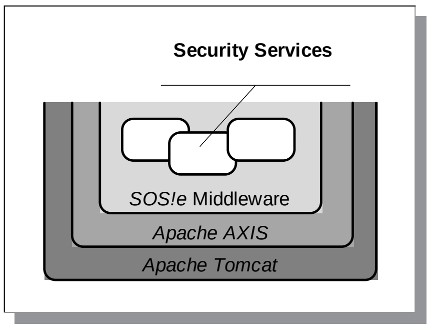

Figure 5.35: Realization and deployment of security services

For optimization reasons, a message request can be processed by several security services which are running inside a single AXIS Web service. Exactly how this is realized will be showed later in this section, when message routing is described.
As middleware, ***SOS!e*** is responsible for invoking the security services and for the realization of the choreography. In other words, when a message request is received, AXIS will invoke the ***SOS!e*** middleware, which will invoke the necessary security services and in the end, will make sure that the message follows its predetermined itinerary.

WSDL Interface for Security Services

Each of the security services along both chains will receive a message, process it and then execute the choreography. In the implementation, all security services use the SOAP document / literal operational mode (listing 5.1 shows the WSDL definition for a security service). The services define a single operation (in the WSDL definition called invoke) which accepts an arbitrary XML document as request and provides an arbitrary XML document as response. The reason for not further defining the request and response messages is twofold: (1) in this way the interface of the security services matches any interface the protected service might have and (2) most of the security services are only concerned with the header of the message and not the body (WSDL documents only define the syntax of the body section of messages).

```
<?xml version="1.0" encoding="UTF-8"?>
<wsdl:definitions
 [... Namespace definitions are omitted ...]
 <wsdl:types>
  <schema targetNamespace="http://framework.services.s3f.unibw.de"
  xmlns="http://www.w3.org/2001/XMLSchema">
   <element name="invoke" type="xsd:anyType" />
  </schema>
  <schema targetNamespace="http://localhost:8080/axis/services/RoutingService"
  xmlns="http://www.w3.org/2001/XMLSchema">
   <element name="invokeReturn" type="xsd:anyType" />
  </schema>
 </wsdl:types>

 <wsdl:message name="invokeRequest">
  <wsdl:part element="tns1:invoke" name="part" />
 </wsdl:message>

 <wsdl:message name="invokeResponse">
  <wsdl:part element="impl:invokeReturn" name="invokeReturn" />
 </wsdl:message>

 <wsdl:portType name="RoutingService">
  <wsdl:operation name="invoke">
   <wsdl:input message="impl:invokeRequest" name="invokeRequest" />
   <wsdl:output message="impl:invokeResponse" name="invokeResponse" />
  </wsdl:operation>
  </wsdl:portType>

 <wsdl:binding name="RoutingServiceSoapBinding" type="impl:RoutingService">
  <wsdlsoap:binding style="document"
  transport="http://schemas.xmlsoap.org/soap/http"/>
  <wsdl:operation name="invoke">
  <wsdlsoap:operation soapAction="" />
   <wsdl:input name="invokeRequest">
    <wsdlsoap:body use="literal" />
   </wsdl:input>

   <wsdl:output name="invokeResponse">
    <wsdlsoap:body use="literal" />
   </wsdl:output>
  </wsdl:operation>
 </wsdl:binding>

 <wsdl:service name="RoutingServiceService">
  <wsdl:port binding="impl:RoutingServiceSoapBinding" name="RoutingService">
   <wsdlsoap:address location="http://host:8080/axis/RoutingService" />
  </wsdl:port>
 </wsdl:service>
</wsdl:definitions>
```

Listing 5.1: WSDL definition for a security service

Message Routing

As previously explained, the most important task of the Front-Side Proxy is to attach the itinerary to incoming request messages. In ***SOS!e 1.0*** , the itinerary consists of a series of hops, each hop representing an AXIS Web service. The information model for itineraries is showed in figure 5.36.

Hops contain an address - the Web service endpoint URL, and a list of security services to be invoked. The services are described through a ClassName and zero or more parameters. When a request message is received, the ***SOS!e*** middleware will sequentially invoke each of the services specified for the respective hop. This is done by instantiating the class and passing the parameters as initialization arguments.

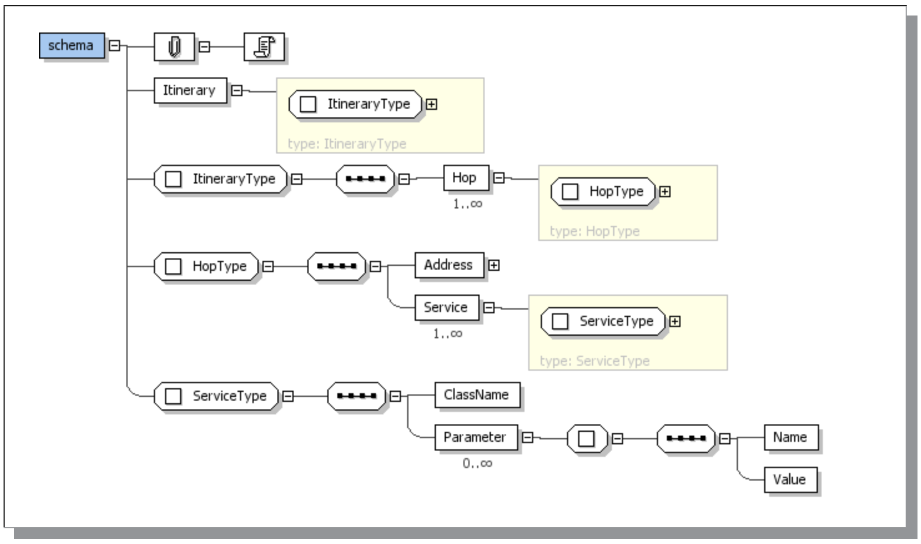

Figure 5.36: Information model for itineraries in ***SOS!e 1.0***

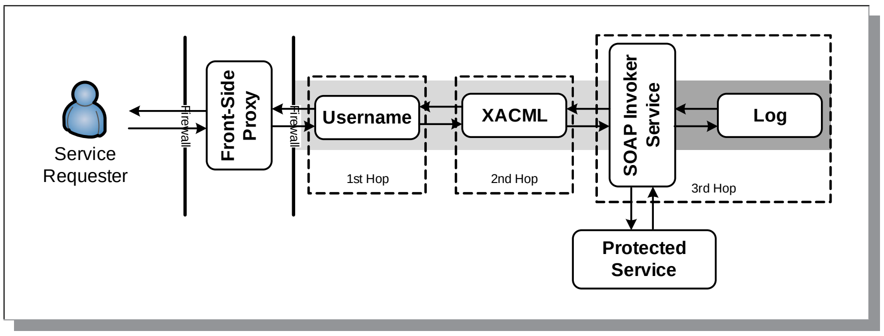

Figure 5.37: Itinerary example

Example Listing 5.2 shows the example of an itinerary as XML code; the same itinerary is showed as a wiring diagram in figure 5.37. The itinerary contains three hops:

1. First, the message is processed by a service which verifies the identity of the requester, based on information stored in a file. The class name is Username and has one single configuration parameter which is the name of the authentication file.

2. Then, the message is processed by an XACML Policy Enforcement Point service. There are two configuration parameters for this service: the URL of the PDP service and its type.

3. The final hop has two services: the first one is the SOAPInvokerService which invokes the protected service (the URL of the protected service is passed as parameter), while the second one is an audit service which logs the response before sending it to the requester - this service requires no parameters.

```
<Itinerary xmlns="http://www.unibw.de/s3f">
 <Hop>
  <Address>http://some.host/some/service/1</Address>
  <Service>
   <ClassName>de.unibw.s3f.services.authentication.Username</ClassName>
   <Parameter>
    <Name>authenticationFile</Name>
    <Value>authentication.properties</Value>
   </Parameter>
  </Service>
 </Hop>
 <Hop>
  <Address>http://some.host/some/service/2</Address>
  <Service>
  <ClassName>de.unibw.s3f.services.authorization.XACML</ClassName>
   <Parameter>
    <Name>pdp.url</Name>
    <Value>http://iisdemo.informatik.unibw-muenchen.de/GeoPDP/service</Value>
   </Parameter>
   <Parameter>
    <Name>pdp.type</Name>
    <Value>geopdp</Value>
   </Parameter>
  </Service>
 </Hop>
 <Hop>
  <Address>http://some.host/some/service/3</Address>
  <Service>
   <ClassName>de.unibw.s3f.services.framework.SOAPInvokerService</ClassName>
   <Parameter>
    <Name>url</Name>
    <Value>http://127.0.0.1:8080/axis/services/MSD3ProxyOWS4</Value>
   </Parameter>
  </Service>
  <Service>
   <ClassName>de.unibw.s3f.services.audit.LogService</ClassName>
  </Service>
 </Hop>
</Itinerary>
```

Listing 5.2: Itinerary example

Message Routing Implementation The itinerary, encoded in XML as exemplified before, is inserted in the message as a special SOAP header element (this is showed in listing 5.3). The only difference is that here, each hop has an additional boolean field - processed; its value is either true, if the corresponding hop has already been visited or false, otherwise. Upon receiving a request, the ***SOS!e*** middleware (1) locates the SOAP header corresponding to the itinerary, then (2) locates the hop block corresponding to itself (this is done by finding the first entry in the itinerary where the processed field is marked false), then (3) invokes the services specified in the block, after that (4) modifies the processed field to true and finally (5) sends the message to the address specified in the next hop block.

```
 <SOAP-ENV:Header>
   [...]
  <proxy:Itinerary xmlns:proxy="..." SOAP-ENV:actor="..."
    SOAP-ENV:mustUnderstand="0">
    <proxy:Hop>
      <proxy:Address>http://some.url/some/service/1</proxy:Address>
       [... The rest of the Hop element contents is omitted ...]
      <proxy:Processed>true</proxy:Processed>
    </proxy:Hop>
    <proxy:Hop>
       [... The rest of the Hop element contents is omitted ...]
      <proxy:Processed>false</proxy:Processed>
    </proxy:Hop>
   [...]
```

Listing 5.3: Itinerary in the header of a SOAP message

Annotations

SOS!e 1.0 does not implement the whole information model presented in the Information Viewpoint. Annotations incorporate only the following information (the information model is showed in figure 5.38): author, target, ID, timestamp and contents. Because the implementation is only a prototype, encryption and digital signatures are not implemented. However, these are implementable through XML ENC [160] and XMD DSig [159]. Furthermore, for the same reasons, only one type of annotations is supported - attribute pair.

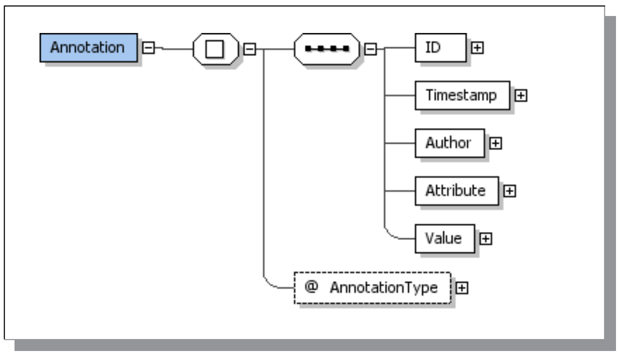

Figure 5.38: Information model for annotation

Annotations are encoded in XML and are implemented as SOAP header elements. According to the SOAP specification [111], header elements contain information which can individually be targeted at SOAP nodes that might be encountered in the path of a message from a sender to an ultimate receiver. This is the case for annotations, which are targeted to specific security services that the message encounters along its path.
For this reason, the actor[13] SOAP attribute is used to represent the target security services. The rest of the information fields are realized as XML elements.

Example Listing 5.4 shows an example annotation. As it can be seen, the annotation is authored by an authentication service and asserts that the userid of the requester is alice27.

```
<SOAP-ENV:Envelope xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/">
 <SOAP-ENV:Header>
   [...]
  <s3f:Annotation xmlns:s3f="http://www.unibw.de/s3f"
    s3f:AnnotationType="AttributePair"
   SOAP-ENV:actor="http://www.unibw.de/s3f"
   SOAP-ENV:mustUnderstand="0" >

   <s3f:Id>15903241-6d7b-40ac-853c-72b20685204c</s3f:Id>
   <s3f:Timestamp>2006-10-31T10:52:53.703Z</s3f:Timestamp>
   <s3f:Author>http://www.unibw.de/s3f/authentication</s3f:Author>
   <s3f:Attribute>userid</s3f:Attribute>
   <s3f:Value>alice27</s3f:Value>
  </s3f:Annotation>
   [...]
```

Listing 5.4: Annotation in a SOAP message

[13]In SOAP 1.1 this attribute is named actor. In SOAP 1.2 the attribute is named role. However the semantics of the attribute are the same in both versions.

Developing Custom Security Services in SOS!e 1.0

SOS!e 1.0 has several security services already implemented. These implement common security tasks such as authentication by means of user-name - password or X509 certificates, simple authorization, logging, mail alert, encryption and digital signing. However, ***SOS!e*** is designed as a framework, and it therefore provides the necessary support to build custom security services.
To do this, the AbstractService class must be extended (figure 5.39). This class provides basic support for retrieving configuration and runtime parameters, processing annotations and altering the message itinerary. Derived classes must implement the abstract method serve; this is the place where the message mediation is realized. The framework then takes care of the rest the aspects (routing, message dispatching) and invokes this method when necessary.

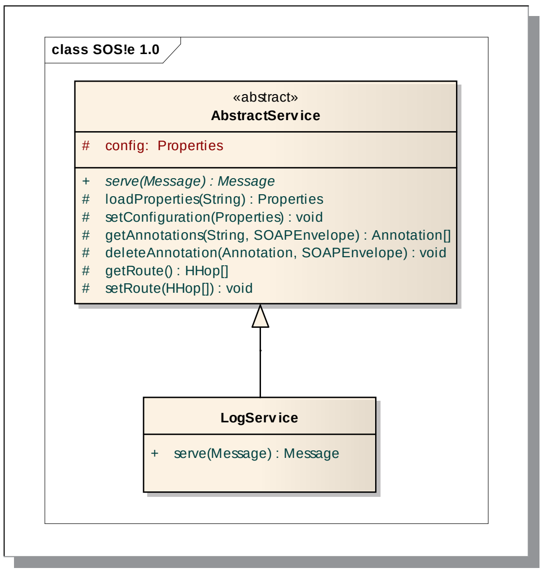
class SOS!e 1.0
«abstract»
AbstractService
# config: Properties
+ serve(Message) : Message
# loadProperties(String) : Properties
# setConfiguration(Properties) : void
# getAnnotations(String, SOAPEnvelope) : Annotation[]
# deleteAnnotation(Annotation, SOAPEnvelope) : void
# getRoute() : HHop[]
# setRoute(HHop[]) : void
LogService
+ serve(Message) : Message

Figure 5.39: Implementation of security services in ***SOS!e 1.0***

Listing 5.5 illustrates the extension mechanisms by showing how a very simple logging service can be implemented. The service simply appends the whole content of the SOAP message at the end of a log file.

```
// Some content is omitted for clarity reasons
public class LogService extends AbstractService {
 private String logFile;
 public Message serve(Message msg) throws ServiceException {
   try {
     PrintWriter log = new PrintWriter(new FileWriter(logFile, true), true);
     log.println(msg.getSOAPPartAsString());
     log.close();
   } catch (Exception e) {
     e.printStackTrace();
     throw new ServiceException(e.getMessage(), e);
   }
   return msg;
 }
}
```

Listing 5.5: Simple implementation of a logging service

Analysis

SOS!e 1.0 was deployed in two projects (OWS3 and OWS4) during the course of which feedback was gathered. The ***SOSA*** concepts were found to be satisfying, but shortcomings were found in several aspects of the implementation. These are discussed in the following paragraphs.

Choreographies
Several aspects regarding the realization of choreographies were found dissatisfying and need improvement:

Custom Implementation Perhaps the most important shortcoming is that choreographies are implemented from scratch in a custom way without leveraging related work. The reason for this was that ESB systems were only in development at the time ***SOS!e 1.0*** was developed, and most of the existing implementations were not open-source. At the time of writing, several open-source ESB systems are in development; integrating security services by means of an ESB would bring a lot of advantages.

SOAP Only The framework can only integrate SOAP Web services.

Message Routing Only itinerary routing is supported in the current implementation. Content-based routing is not supported.

Asynchronous Messaging Because it is implemented on top of AXIS 1.x, all services expose a request-reply interface. The routing of messages is implemented on top of this request-reply pattern which leads to lower performance and higher memory consumption because services block and wait for the rest of the queue to finish.

Annotations
There is no support for encryption and digital signatures.

Speed and Memory
Several factors lead to poor performance and high memory requirements. One of them was already mentioned - the use of synchronous messaging. Another factor is the use of the DOM XML parsing model: even though most security services only process a small portion of the message, the parser will first build the whole DOM tree consuming both memory and processor time. Other parsing models, such as the newer XPP used by AXIS2 [126], are expected to visibly improve performance.

5.6.3 SOS!e 2.0

After experimenting with ***SOS!e 1.0*** , it was decided that a complete rewriting of the framework is necessary in order to test some new concepts and implementation possibilities. The major driver for this was the development of Enterprise Services Bus (ESB) software - several implementations, both commercial and open-source, made their way into the market.
SOS!e 2.0 was developed as part of a diploma thesis [82]. The framework was used in another diploma thesis [161] where accounting and charging services have been developed. An article describing the concept, implementation and some preliminary testing results is also pending publication [119]. For detailed information regarding the implementation please refer to the bibliographic citations.

Software Platform, Libraries and Tools

As its first version, ***SOS!e 2.0*** is implemented in Java and uses several open-source libraries, most notably the *Xerces 1.4.4* parser (`http://xerces.apache.org/xerces-j`), *Xalan 2.7.0* for XPath evaluations (`http://xml.apache.org/xalan-j`), *WSS4J 1.5.2* as WS-Security implementation (`http://ws.apache.org/wss4j`), *OpenSAML 1.1* as SAML implementation (`http://www.opensaml.org`), *Log4J 1.2.9* and *Commons-HttpClient 3.0.1*. *Ant 1.7* was used to automate build tasks (`http://ant.apache.org`).
The major difference to the first version is that the *Mule ESB 1.4.0* is used to aggregate the services together (`http://mule.codehaus.org`). Although the original plan was to use AXIS2 as container for Web services, this was not possible because the up-mentioned version of Mule only supports AXIS 1.x services. Integration with AXIS2 remains an item for future work (see 8.4).

Wiring Diagram

Figure 5.40 shows the deployment of the ***SOS!e 2.0*** framework. Several security services, in the figure named A, B, ..., Z, are plugged into the Mule ESB. They are stand alone Web services and can each be deployed in a separate Web service container. In order to be able to communicate with one another, all security services use the ***SOS!e 2.0*** annotation library which contains the main functions necessary to read, create, modify and delete annotations from a SOAP message.

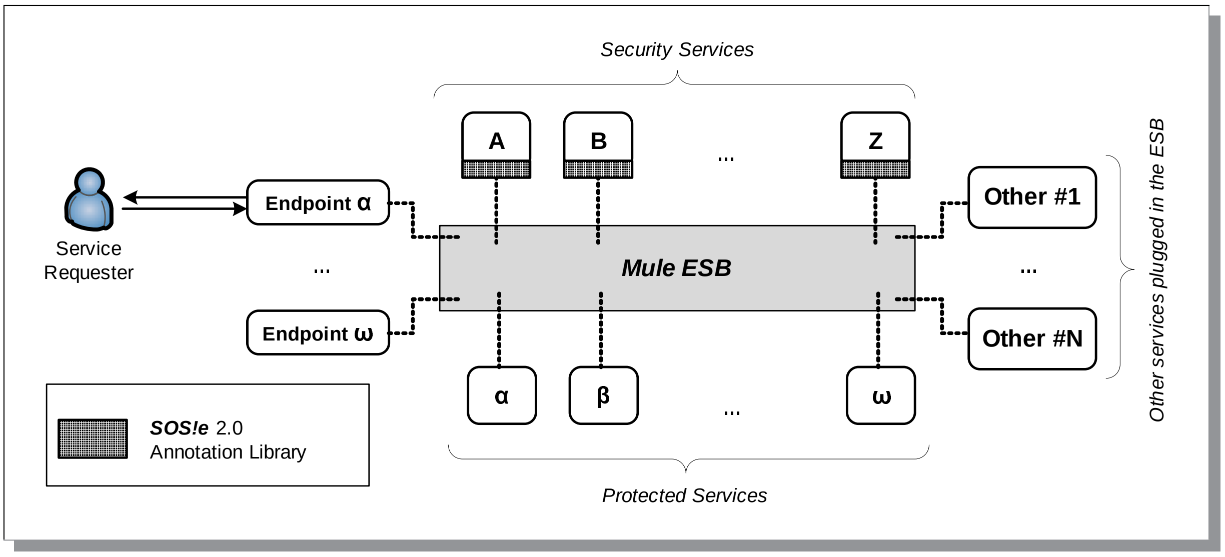

Figure 5.40: Wiring diagram for ***SOS!e 2.0***

Also connected to the ESB are several protected services, in figure 5.40 these are named with Greek letters (α, β, ..., ω). For each of these protected services, an endpoint is exposed (Endpoint α ... Endpoint ω). The ESB is configured to route the request messages that it receives on each of the endpoints through a chain of security services before sending them to the protected service. In a similar fashion, response messages are routed through several security services, before being sent back to the requester. Furthermore, the ESB can be configured to take advantage of other services already existing in the infrastructure (in the figure named Other #1, ..., Other #N).

Remarks

- The proxy pattern is also applied here: the endpoints mentioned above are proxies for the respective protected services. They expose the same functionality, but the interface is enhanced to support security features.
- In comparison with the initial version of ***SOS!e*** , here the aspects related to message routing and orchestrations are completely left to the ESB software. This is a powerful integration middleware which supports a variety of routing patterns, message exchange patterns and transport protocols. This brings more flexibility in architecting security systems and decouples the aspects related to service orchestration (realized through the ESB) from the communication aspects specific to security services (implemented in the ***SOS!e*** annotation library).

Message Routing and Orchestrations

The Mule ESB was used for the realization of message routing and orchestrations. It supports most of the messaging patterns described in [95], including itinerary routing and content-based routing. There are various possibilities as to how the ESB can be configured and deployed - the details of this are described in the Mule User Guide [2]. To illustrate the concepts, a sample configuration file is presented in listing 5.6 and briefly described in the following.

```
<mule-configuration id="MuleChainingSample" version="1.0">
 <description>Example SOS!e 2.0 configuration</description>
 <mule-environment-properties synchronous="true"/>
 <transformers>
  <transformer name="SOAP2String" className="org.mule.samples.myApp.SOAPMessageToString"/>
 </transformers>

 <model name="MuleConfig">
  <mule-descriptor name="sosie" implementation="org.mule.components.simple.BridgeComponent">
   <inbound-router>
    <endpoint address="http://localhost:8081/demo" remoteSync="true"/>
   </inbound-router>

   <outbound-router>
    <router className="org.mule.routing.outbound.ChainingRouter">
     <endpoint address="axis:http://some.host/Authentication" synchronous="true">
      <properties>
       <property name="style" value="document"/><property name="use" value="literal"/>
      </properties>
     </endpoint>
     <endpoint address="axis:http://some.host/Authorization" synchronous="true">
      <properties>
       <property name="style" value="document"/><property name="use" value="literal"/>
      </properties>
     </endpoint>
     <endpoint address="axis:http://some.host/Echo" synchronous="true">
      <properties>
       <property name="style" value="document"/><property name="use" value="literal"/>
      </properties>
     </endpoint>
    </router>
   </outbound-router>
  </mule-descriptor>
 </model>
</mule-configuration>
```

Listing 5.6: Example configuration for the Mule ESB

In this example, a single mule-descriptor is defined which contains an inbound-router as well as an outbound-router. As a simple rule, messages received on the endpoints specified in the inbound-router, are processed by the Mule middleware (here a BridgeComponent is used), and then forwarded according to the outbound-router.
In the example above, all messages received on http://localhost:8081/demo are processed by the BridgeComponent (this leaves the messages unchanged), and then given to the outbound-router. The one used here (ChainingRouter) implements itinerary routing and will first forward messages to an authentication service, then to an authorization service and finally to the protected service (http://some.host/Echo). The axis prefix in front of the URLs tells the ESB that AXIS Web service invocations shall be used, while the properties defined for each endpoint specify that the document / literal SOAP style shall be used.

Remarks

- In this example the security functionality is implemented in stand-alone Web services deployed in an AXIS container. However it is also possible to implement the same functionality directly into the Mule ESB. For this purpose Mule defines two possibilities: interceptors and transformers. This approach renders better performance (instead of converting messages into XML and sending them over the wire, Java objects are passed from one class to the other inside the same JVM) at the cost of having the implementation dependable on the middleware (less portability).
- Mule also defines filters which can be used in specifying content-based routing rules. The current version contains several very useful implementations that filter messages based on the payload type, regular expressions, XPath expressions, etc.
- The inbound-router as well as the outbound-router can also be used to aggregate / split messages (see ESB patterns, section 2.3.5).

Annotations

In ***SOS!e 2.0*** annotations are implemented using SAML attribute assertions. Most of the information elements for annotations are already present in SAML assertions:

Author The issuer of the SAML assertion;

ID The ID of the SAML assertion;

Timestamp The issue instant of the SAML assertion;

Content Several attribute-value pairs can be included in an attribute assertion.

The only field that is not present in SAML assertions is target. This has been implemented as a special attribute (name collision is avoided through the use of different attribute namespaces).

Example Listing 5.7 shows an annotation attached to a SOAP message. The annotation is issued by an AuthenticationService and asserts that the userid of the requester is johndoe27. The annotation is digitally signed, however the body of the signature is omitted in the listing.

```
<soapenv:Envelope xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/">
<soapenv:Header>
 <wsse:Security soapenv:actor="http://unibw.de/sosie/dummy" soapenv:mustUnderstand="1">
  <Assertion xmlns="..." AssertionID="_198bdd0d5201265b1d19e8cdc367c58e"
    IssueInstant="2007-07-29T08:07:41.218Z" Issuer="AuthenticationService"
    MajorVersion="1" MinorVersion="1">

  <Conditions NotBefore="2007-07-29T08:07:41.218Z"
    NotOnOrAfter="2007-07-29T08:08:41.218Z"/>

  <AttributeStatement>
   <Subject>
    <NameIdentifier>AuthenticationService</NameIdentifier>
     <SubjectConfirmation>
      <ConfirmationMethod>urn:oasis:names:tc:SAML:1.0:cm:sender-vouches</ConfirmationMethod>
    </SubjectConfirmation>
   </Subject>
   <Attribute AttributeName="userid" AttributeNamespace="[...] annotation/attributes">
    <AttributeValue>johndoe27</AttributeValue>
   </Attribute>
   <Attribute AttributeName="TARGET" AttributeNamespace="[...] annotation/framework">
    <AttributeValue>http://unibw.de/sosie/authentication</AttributeValue>
   </Attribute>
  </AttributeStatement>

  <ds:Signature xmlns:ds="http://www.w3.org/2000/09/xmldsig#">
     [ ... ]
  </ds:Signature>
 </Assertion>
```

Listing 5.7: Annotation in a SOAP message (as SAML assertion)

Developing Custom Security Services in SOS!e 2.0

Several security services were implemented on top of the ***SOS!e 2.0*** framework. These implement common security tasks such as authentication by means of user-name password against an LDAP directory, simple authorization, audit, accounting or charging by means of PayPal. More details can be found in [82, 161] and in the next chapter.
As far as developing custom security services, the mechanisms built in ***SOS!e 2.0*** are similar to the ones from the first version of the framework. An AbstractService class is defined which must be extended by all security services. This class implements the functions for processing annotations (retrieval, creation, deletion, modification). The UML diagram in figure 5.41 shows how a LogService can be defined by extending the AbstractService and implementing the serve method.

Remarks

- From the UML diagram one can see that here the security services are not handling the orchestration aspects. As opposed to ***SOS!e 1.0*** where the API exposed functions for the modification of the message itinerary, here no such functions are exposed. The reason for this is that the orchestration aspects are entirely left to the ESB software.

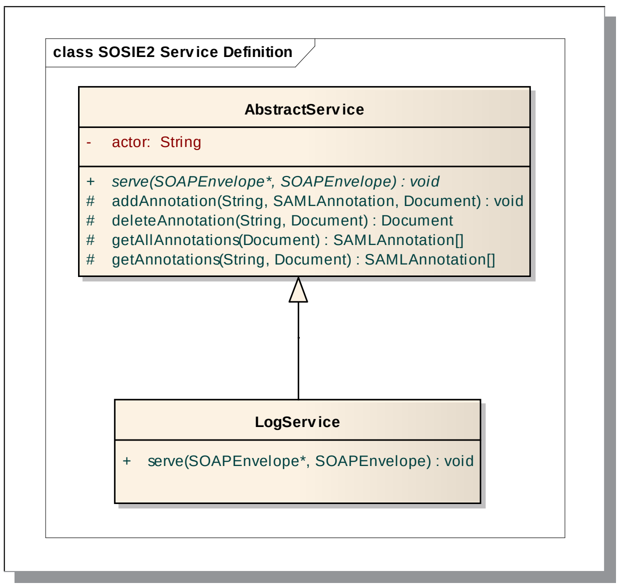
class SOSIE2 Service Definition
AbstractService
- actor: String
+ serve(SOAPEnvelope*, SOAPEnvelope) : void
# addAnnotation(String, SAMLAnnotation, Document) : void
# deleteAnnotation(String, Document) : Document
# getAllAnnotations(Document) : SAMLAnnotation[]
# getAnnotations(String, Document) : SAMLAnnotation[]
LogService
+ serve(SOAPEnvelope*, SOAPEnvelope) : void

Figure 5.41: Implementation of custom security services in ***SOS!e 2.0***

Analysis

SOS!e 2.0 was built based on feedback received for the first version of the framework. The following represent the major changes in this version of the framework:

Choreographies These are realized by means of an off-the-shelf ESB software (Mule). This brings several advantages: support for more message routing patterns, more transport protocols, more message exchange patterns (including asynchronous messaging) and more deployment configurations.

Annotations These are built on top of the SAML specification. This increases the level of interoperability (SAML is a well established standard for exchanging security information) and allows annotations to be digitally signed.

As a drawback, this implementation is not able to work with AXIS2 Web services because the current version of Mule does not support this feature. However, because the framework is separated from the ESB software, it will be easy to upgrade the framework when the AXIS2 support will be implemented in Mule.

Chapter 6

Practical Experiences with SOSA

The concepts described in the previous chapter have been applied in several projects, where the topic of securing geospatial Web services was addressed.
This chapter opens with a brief introduction to the problematic of securing geospatial Web services, and then several projects are presented. Each project is described in terms of *context*, *problem statement and objectives*, *requirements* and *design and implementation.* After this, three independently developed security services are briefly described. Finally, conclusions are drawn and the practical experiences are summarized.

6.1 Security for Geospatial Web Services

6.1.1 Context

Geospatial information is the key data for enabling decision making in many business sectors. Because Web services offer a very good data distribution channel, both governments and private organizations are currently building Spatial Data Infrastructures (SDIs). Several interfaces for geospatial services are defined by the Open Geospatial Consortium - these have been introduced in section 2.2.
An important obstacle in the adoption of this technology are aspects related to security. [91] remarks that because it is expensive to create and maintain, many organizations involved in the production and trading of geodata find the need to protect their Intellectual Property (IP) assets through the digital distribution value chain; further motivation for considering the security aspects can be found in [64]. However, little guidance is provided as to how these aspects are to be handled: the current versions of the OGC service specifications do not address these issues and the OGC provides no best practices documents for this.
There are nevertheless approaches to this both from the industry as well as from the academia, examples for this are described in [118, 107, 152]. In the following I will show how the concepts described in this dissertation have been applied in several projects where different security solutions for geospatial Web services were prototypically implemented.

6.1.2 Approach

The general approach is to separate the security aspects from the geo-processing functionality as much as possible at the technical level. The purpose of this is to treat the geo-processing services as *security unaware* and to implement the security externally, independent of the geo-processing functionality. The principle together with the advantages for this approach have been discussed in section 3.2.

6.1.3 SOAP Communication

The basis for this approach is to use SOAP as the communication protocol. SOAP permits a clear separation between the payload (which is geo-specific) and the security information (licenses, authentication information, etc). The security information is carried in the SOAP header part, as specified in WS-Security and the related standards. The geo-data specific requests and answers are carried in the SOAP body and need not be modified normally.

6.1.4 Proxy Components

In order to take advantage of the existing implementations for OGC Web services and clients, proxies (façades) were used. These are deployed in front of a geo-processing service or client in order to enhance their functionality (see figure 6.1). This architectural design facilitates the separation between the geo-processing functionality and the security aspects at the implementation level and has been applied several times:

- On the client-side the proxy enhances the client capabilities by implementing the security aspects.
- Because only few OGC service implementations supporting SOAP were available, several server-side proxies were used as HTTP-SOAP protocol converters.
- A server-proxy was used to implement a Gateway service (see Computational Viewpoint, section 5.3).

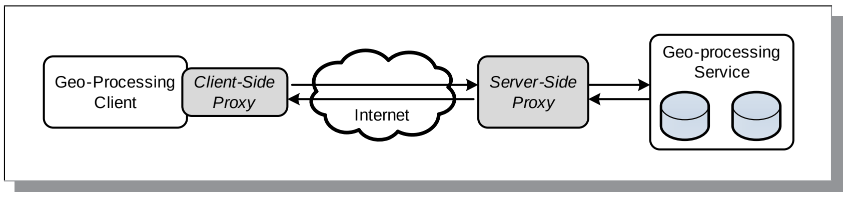

Figure 6.1: Proxies for geo-processing services

6.1.5 The SOS!e Framework

Having a SOAP communication which separates the geospatial aspects from the security aspects at the information level, the ***SOS!e*** framework was leveraged to implement various security aspects. For this, different security services were developed according to the requirements of each project. Figure 6.2 illustrates this.

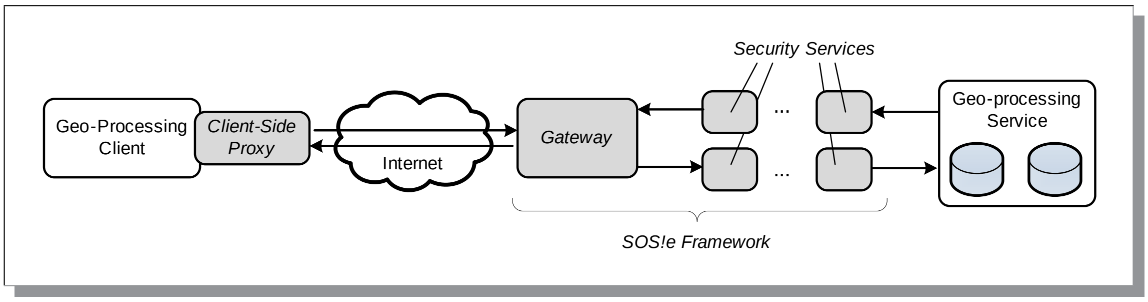

Figure 6.2: ***SOS!e*** protecting geo-processing services

6.2 Click-through Licensing for OGC Web Services

6.2.1 Context

As part of the GeoDRM thread of the OGC Web Services Testbed 3 activity, the University of the German Armed Forces has participated in the design and implementation of click-through licensing for the Web Map Service and Web Feature Service interfaces. The results of this project are documented in [105], have been presented within the 55th OGC TC Meeting and are available online, both on the OGC website[1] and on the UniBw's website[2]. The concepts developed here were further improved in a joint research project with the Bavarian Surveying Office[3] - the results of the latter one are documented in [120].

In the following, I will briefly describe different aspects of the project. The emphasis is here on how ***SOSA*** was applied in the solution design. Other issues, such as proposed extensions to the WMS/WFS specifications or details about the implementation are intentionally left out. For these, please refer to the references provided above.

Even though the security requirements for this project are very basic (in its simplest case, only some basic form of authorization is necessary), this project illustrates how two of the ***SOSA*** patterns described in the Engineering Viewpoint (section 5.5) can be applied in practice: the gateway pattern and the session establishment pattern.

6.2.2 Problem Statement and Objectives

The exact objectives of the project together with detailed requirements, use cases, deliverables and component diagrams are detailed in [122]. This subsection only presents an overview.

The objective of the GeoDRM thread within OWS-3 was to extend the "click-through" licensing concept which is widely deployed on web sites to geospatial Web services. In particular, click-through licensing techniques needed to be developed for the Web Map Service (WMS) [63] and Web Feature Service (WFS) [154]. Furthermore, the activity should be coordinated with the Feature Portrayal Service (FPS) [157] specification so that the same techniques developed for the WMS and WFS could be readily applied to the FPS.

The *click-through license* (also known as *clickwrap license* or *clickwrap agreement*) is a common type of agreement which is often found on the Internet. The content and form of these agreements vary widely, however most of them require the end user to manifest its acceptance of a certain textual license by clicking on an OK button. Typical uses of click-though licensing on the Internet

[1]`http://www.opengeospatial.org/projects/initiatives/ows-3`
[2]`http://iisdemo.informatik.unibw-muenchen.de/ows3`
[3]Landesamt für Vermessung und Geoinformation - `http://www.geodaten.bayern.de`

involve disclaimer of warranty and limitation of liability.
In the OWS-3 project it was required that before a certain user could make use of a WMS or WFS service, he or she would need to first read and accept a certain license agreement which is presented in textual form. In addition to this, the service needs to implement some sort of session mechanism, so that the user is only presented once with the click-through license and not before every request (imagine a user requesting an initial map and then navigating by performing pan / zoom operations). The licenses are enforced on a per-layer basis in the case of WMS (each layer can have its own license) and on a per-feature-class basis in the case of WFS (each feature class can have its own license). A further requirement are audit traces: the server should maintain a log documenting the accesses for each service, and the acceptance of licenses.

6.2.3 Requirements Analysis

Use-Cases and Agreement Workflow

In the analysis phase, several use-cases were developed. Two of these use-cases which were also implemented during the project, are described below. The workflows involved in the two use-case are illustrated in figure 6.3.

Named-User In this use-case users are known by the service. We assume that they register with the service before requesting geospatial information. During this process some form of authentication information is generated by the service and shared with the user. The interactions are showed in figure 6.3A. After registration, the users attach their identity information to subsequent service requests. The responses contain either the requested information or an exception with an URL to a website where the click through license can be read and accepted.

Annonymous-User In this use-case the service does not keep track of the identity of the users. Because of this, an additional step is required, namely the establishment of a session. This is done explicitly by the client which calls the GetSession operation and receives a session token in return. The rest of the process is the same as in the previous case. The interactions are showed in figure 6.3B.

Security Requirements

From the use-cases described above and the project objectives described in 6.2.2, considering the security requirements described in 2.4, the following three security issues render as important and should be taken into consideration in the system design:

Authentication In the Named User use-case, the identity of the clients must be verified. In the Anonymous User use-case, the validity of the session token must be verified for every request received. Therefore authentication is important in both use-cases.

Authorization Knowing the identity of the requester (the session token), the service needs to verify if the requester has agreed with the licenses corresponding to the requested layers / feature classes.

Audit The service should maintain audit trails documenting for each user the layers viewed, the applicable licenses and the date and time when the user acknowledged those licenses.

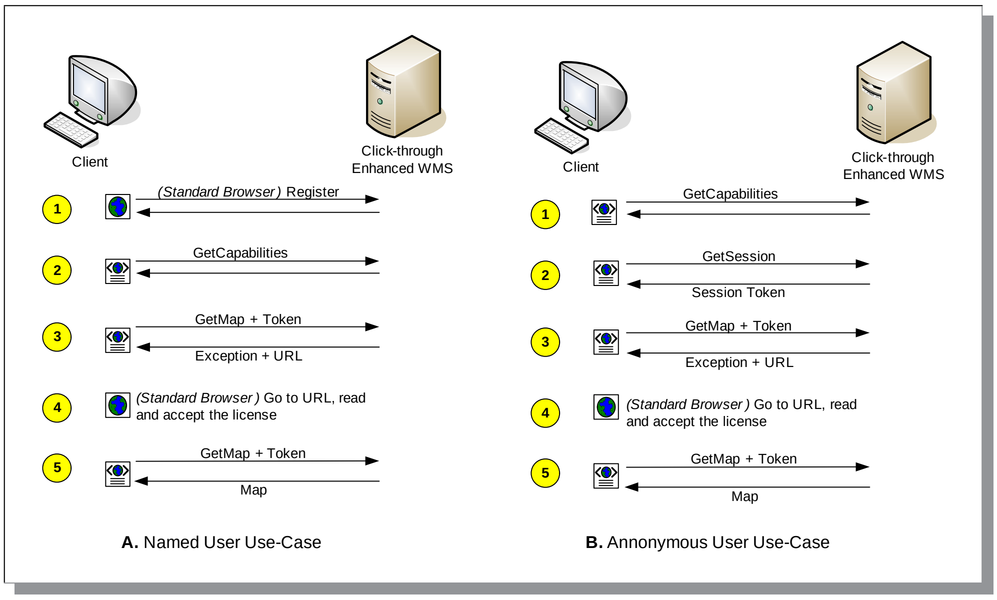
Client
Click-through Enhanced WMS
1
(Standard Browser) Register
2
GetCapabilities
3
GetMap + Token
Exception + URL
4
(Standard Browser) Go to URL, read and accept the license
5
GetMap + Token
Map
A. Named User Use-Case
Client
Click-through Enhanced WMS
1
GetCapabilities
2
GetSession
Session Token
3
GetMap + Token
Exception + URL
4
(Standard Browser) Go to URL, read and accept the license
5
GetMap + Token
Map
B. Annonymous User Use-Case

Figure 6.3: Click-through licensing for OGC Web services: Use-cases

6.2.4 Design and Implementation

Session Management

In the Named-User use-case, the click-through agreements are associated with the user name. This one is sent together with every request, in the header of the SOAP message.
In the Anonymous-User use-case, the click-through agreements are associated with a session ID. In order to have a unified design, sessions are implemented on top of username-password authentication tokens. The GetSession operation returns a username-password pair that is valid for a single session (an example is given in listing 6.1 below).

```
<SOAP-ENV:Envelope xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/">
<SOAP-ENV:Header/>
<SOAP-ENV:Body>
  <ows3:UserNameToken xmlns:ows3="http://iis.unibw-muenchen.de/ows3">
    <ows3:UserName>sessionID:550e8400-e29b-41d4-a716-446655440000</ows3:UserName>
    <ows3:Password>7263db23</ows3:Password>
  </ows3:UserNameToken>
</SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

Listing 6.1: Example response for the GetSession operation

Wiring Diagram

The system design is presented in figure 6.4. In this figure the main components, data stores and inter-component communications are showed. The description of the components can be found below.

Gateway This is an implementation of the Gateway pattern presented in the Engineering Viewpoint. This component acts as an endpoint for the service (clients from the external network

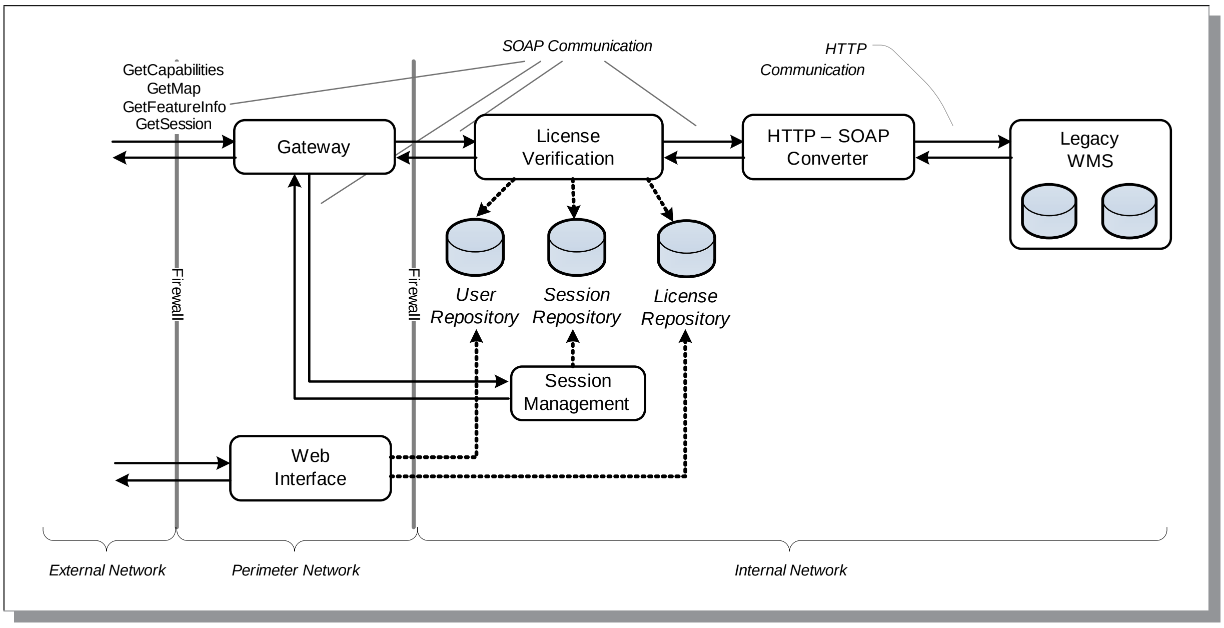

Figure 6.4: Click-through licensing for OGC Web services: Wiring diagram

can only access this Web service endpoint). The responsibilities of this component are listed below[4]:

- Inspects the incoming messages and determines their itineraries: the GetSession calls will be sent to the Session Management component, while the other calls will be routed through the License Verification component.
- Attach the itinerary to the request.
- Forward the request to the first service from the itinerary.

Session Management This component implements the GetSession operation. For this, it generates session IDs (username-password pairs) and stores them in the Session Repository.

License Verification This component implements all three security functions determined in the analysis: it verifies that the supplied username-password pairs are valid (by inspecting the User Repository and the Session Repository), checks if the appropriate click-through agreements have been accepted by the requester (by inspecting the License Repository) and also leaves audit trails.

HTTP-SOAP Converter Because no WMS/WFS services implementing SOAP were available at the time of the implementation, a protocol converter was introduced. The converter is an implementation of the Message Transformation Pattern described in [95]. Such converters are typical examples for mediation services in an ESB.

Legacy WMS This is the Web Map Service itself, an of-the-shelf component. It does not need modifications or special configuration.

Web Interface Through the web interface users can register with the service (as required in the Named-User use-case) and can read and accept license agreements. For this, the interface

[4]For a detailed description of this component refer to section 5.6, where ***SOS!e 1.0*** is described (this component corresponds to the FrontSideProxy described there).

interacts with the User Repository - in order to introduce new users and the License Repository - in order to update the license agreements. This component is not a web service: it consists of dynamic web pages and can be implemented with technologies such as JSP, ASP, PHP, etc.

Design Analysis

The design of the security system adopts the ***SOSA*** principles: the security system is external to the protected Web service and is composed of several Web services which are aggregated by means of message routing. However, the security services are not developed according to the taxonomy presented in the Computational Viewpoint (section 5.3): instead of implementing three different services performing authentication, authorization and audit, a single security service (the License Verification service) is proposed which implements all three functions. The motivation for this decision is that the tasks for all three functions are very simple and separating them into independent services would only introduce delays in the response time without any further benefits. Because the services process the requests independently (there is no need to exchange information among services), annotations are not used.

6.3 Biometric Authentication for OGC Web Services

6.3.1 Context

As a research project started in 2004 at the University of the German Armed Forces and further developed at the University of Zürich, BioLANCC (*Biometric Local Area Network Control Center*) is a system for managing physical access that uses biometric data (fingerprint, iris scans, etc.) to identify users. The system aims at providing an extensible solution which is operating system independent, database independent and is able to integrate different types of biometric devices from different manufacturers.

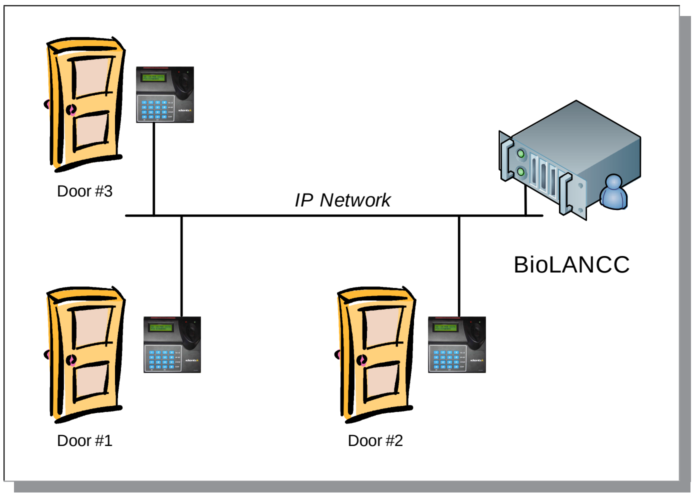

Figure 6.5: Typical deployment for BioLANCC

In a typical deployment, each door is fitted with a biometric access control device. The devices allow users to authenticate themselves, perform access control and can be connected to relays that open the doors (see figure 6.5). The communication between the biometric devices and the BioLANCC is done be means of TCP/IP, and allows the BioLANCC server to configure access restrictions on the devices and the devices to report events to the server (successful / failed authentications, etc).

In the following the BioLANCC will not be further detailed - for more information about this topic please refer to [90, 127, 135, 142]. Instead, I will show how the BioLANCC system was integrated by means of ***SOS!e 1.0*** so that it provides authentication to OGC Web services. This integration was developed as part of several diploma thesis at the University of the German Armed Forces. The results are described in [90, 128] - please refer to these for details. Furthermore, this project was presented at several industry fairs: Systems 2005, RTGIS Fortbildungsseminar 2006, Systems 2006. The rest of this section presents in short the problem, analysis elements and shows the system design (which follows the ***SOSA*** principles).

6.3.2 Problem Statement and Objectives

The objective was to integrate the BioLANCC system with OGC Web services (WMS and WFS) so that it is possible to authenticate users through BioLANCC. This will allow users to authenticate themselves by using one of the biometric readers which are connected to BioLANCC. Furthermore, because BioLANCC also stores user attributes, these can be used in the authorization process - i.e. access restrictions can be developed based on these attributes.

6.3.3 Requirements Analysis

This project is a typical example for a security integration project - some existing security infrastructure (the BioLANCC system) needs to be integrated with another system which was developed independent of the previous one - the OGC Web services.
The following list enumerates the most important requirements that were taken into consideration when designing the security system:

- Because it is an existing system which is installed and running (the deployment at the Informatics Department, UniBw consists of more than 15 fingerprint readers and a database with more than 100 users), the modifications to the system should be minimal and a clear separation between the BioLANCC and the servers hosting the OGC Web Services should be kept (the two are deployed in separate networks).

- The biometric readers used were fingerprint scanners equipped with a numeric keypad. In order to authenticate to such a device, a user needs to first introduce its PIN and then perform a fingerprint scan.

- Besides authentication, the integration should also take into consideration authorization and audit. It should be possible to define simple authorization rules and the security system should document the accesses to the protected Web service.

- As already mentioned, the outcome of this experiment was to be able to use the biometric readers when authenticating users accessing Web services. Because these terminals are usually fitted in the wall, near access doors, the access control can only be provided for users which are physically located near one of these terminals (such as for example users accessing a Web service from within a room which is equipped with a biometric reader). Several possible deployment scenarios are possible, however these will not be discussed here - for more information about deployment scenarios as well as a security analysis refer to [90].

6.3.4 Design and Implementation

Authentication Process

In order to be authenticated and to access the Web service, several steps are required. The communication between the different components involved in the authentication process is described below and illustrated in figure 6.6.

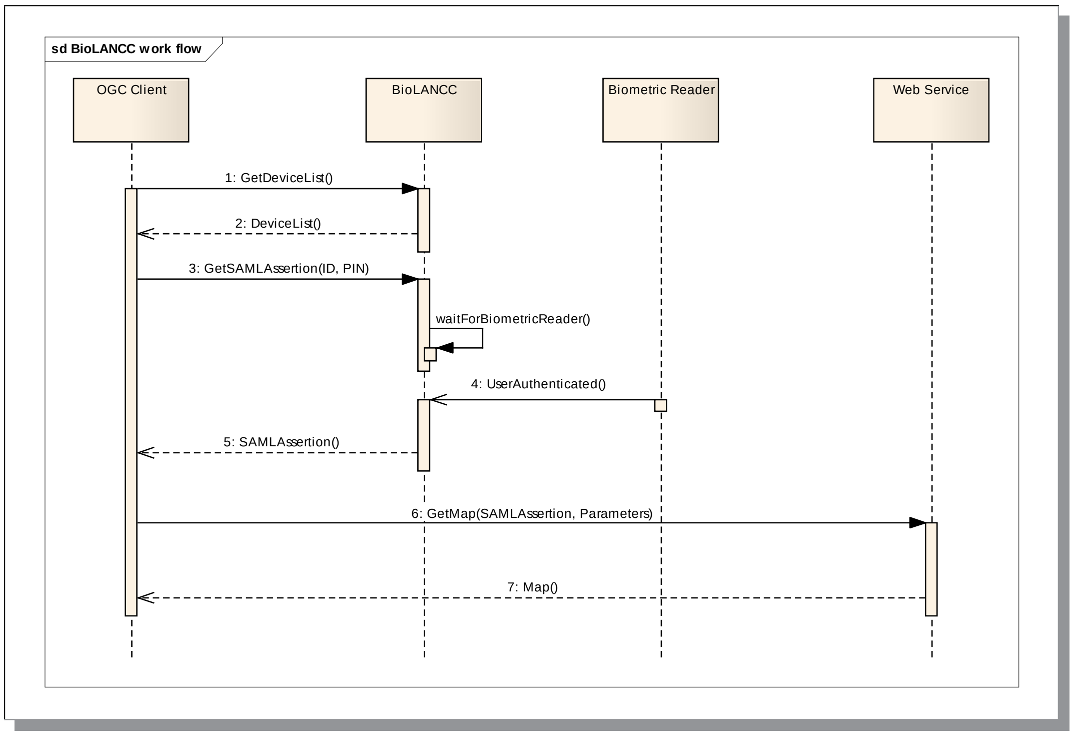

Figure 6.6: Biometric authentication for OGC Web services: Authentication process

1. The client requests the list of biometric devices from the BioLANCC server.
2. The service returns the requested list.
3. With this list, he chooses one of the devices form the list (perhaps the one closest to him), introduces its PIN and requests a SAML Authentication Assertion from the BioLANCC server.
4. The user then authenticates to the biometric reader, by typing its PIN and scanning its finger. Biometric readers report all events, including successful / unsuccessful authentications, to the BioLANCC. Upon receieving a request for an assertion, BioLANCC will wait a certain amount of time to receive a successful authentication event from the specified fingerprint reader.
5. After the event is received, BioLANCC will generate a SAML authentication assertion and send this one to the client in response to its request. The assertion is signed with a private key which is installed on the BioLANCC server.
6. The client then makes a request to the OGC service, attaching the SAML assertion to its request. The service can authenticate the client based on the information from the SAML assertion. Furthermore, having the corresponding public key, it is also possible to verify the authenticity of the SAML assertion.

Extensions to Existing software

The following extensions to existing software were developed (these are also showed in figure 6.7):

OGC Client A client-side proxy was developed which is able to connect to the BioLANCC server and request SAML authentication assertions. When sending OGC requests, the client includes the retrieved assertions in the WS-Security SOAP header.

BioLANCC A Web service interface was implemented for the BioLANCC. The following two operations were exposed:

- **GetNodeList()** Returns a list with all the biometric devices which are managed by the system.
- **GetSAMLAssertion(String ID, String PIN)** Receives the ID of a biometric reader and a PIN identifying the requester. The system then waits for the user to authenticate to the specified biometric reader and, if the authentication is successful, a SAML Authentication Assertion is returned. This method implements the SAML protocol for querying authentication statements, as described in [12]. Example request and response messages are showed in listings 6.2 and 6.3 respectively.

OGC Services The OGC services were enhanced by means of the ***SOS!e 1.0*** framework. Several services were developed; these are presented in the next section together with the wiring diagram.

```
<Request IssueInstant="..." MajorVersion="1" MinorVersion="1" RequestID="...">
  <RespondWith>saml:AuthenticationStatement</RespondWith>
  <AuthenticationQuery AuthenticationMethod="urn:oasis:names:tc:SAML:1.0:am:HardwareToken">
    <Subject xmlns="urn:oasis:names:tc:SAML:1.0:assertion">
      <NameIdentifier Format="urn:oasis:names:tc:SAML:1.1:nameid-format:unspecified"
        NameQualifier="UserID">3008</NameIdentifier>
      <SubjectConfirmation>
        <ConfirmationMethod>NodeID:15</ConfirmationMethod>
      </SubjectConfirmation>
    </Subject>
  </AuthenticationQuery>
</Request>
```

Listing 6.2: GetSAMLAssertion request

```
<Response InResponseTo="..." IssueInstant="..." MajorVersion="1" MinorVersion="1"
 Recipient="pc63-163.informatik.unibw-muenchen.de/137.193.63.163" ResponseID="...">
 <Status>
  <StatusCode Value="samlp:Success"/>
 </Status>
 <Assertion xmlns="urn:oasis:names:tc:SAML:1.0:assertion"
  AssertionID="..." IssueInstant="2005-06-01T13:39:56.312Z"
  Issuer="iisdemo.informatik.unibw-muenchen.de" MajorVersion="1" MinorVersion="1">
  <ds:Signature xmlns:ds="http://www.w3.org/2000/09/xmldsig#">Re2A...ak=</ds:Signature>
  <Conditions NotBefore="2005-06-01T13:39:56.312Z" NotOnOrAfter="2005-06-02T15:39:56.312Z"/>
  <AuthenticationStatement AuthenticationInstant="2005-06-01T13:39:56.312Z"
  AuthenticationMethod="urn:oasis:names:tc:SAML:1.0:am:HardwareToken">
   <Subject>
    <NameIdentifier Format="urn:oasis:names:tc:SAML:1.1:nameid-format:unspecified"
    NameQualifier="UserID">3008</NameIdentifier>
     <SubjectConfirmation>
      <ConfirmationMethod>NodeID:15</ConfirmationMethod>
     </SubjectConfirmation>
    </Subject>
   <SubjectLocality DNSAddress="pc63-163.informatik.unibw-muenchen.de"
```

```
      IPAddress="137.193.63.163"/>
    </AuthenticationStatement>
  </Assertion>
</Response>
```

Listing 6.3: Response to a GetSAMLAssertion request

Wiring Diagram

The system design is presented in figure 6.7. In this figure the main components, data stores and inter-component communications are showed. The description of the components can be found below. Figure 6.8 shows how the contents of a request to the OGC service is altered as the message passes through the security framework.

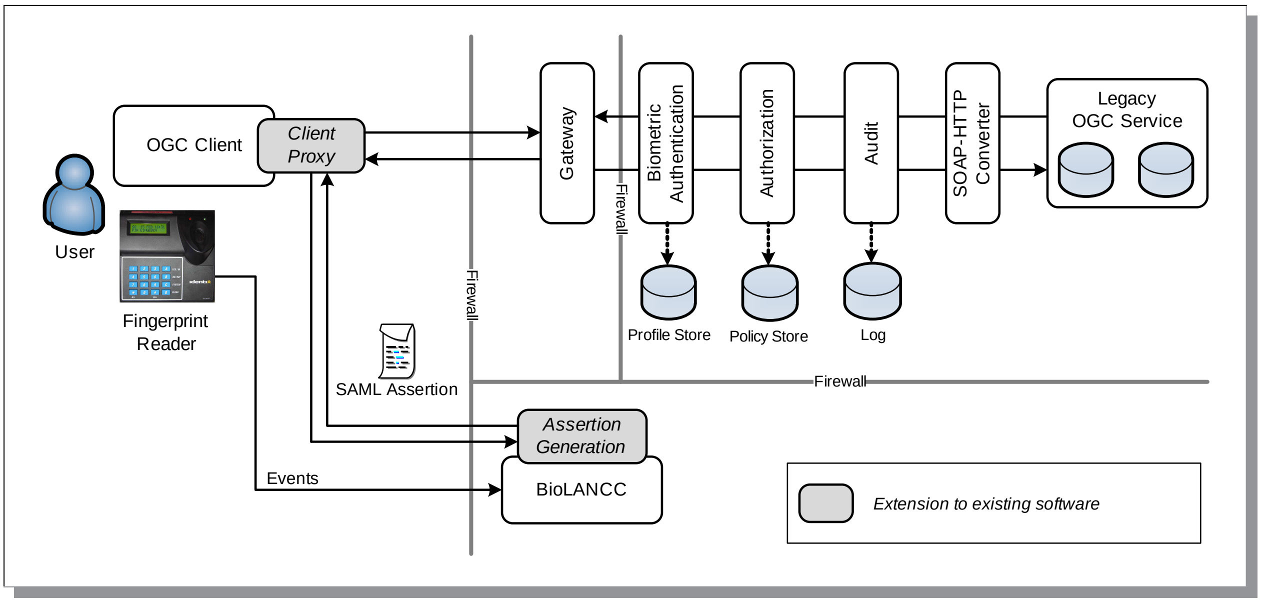

Figure 6.7: Biometric authentication for OGC Web services: Security system design

Gateway Similar to the design approach taken in the previous project, this component is an implementation of the Gateway pattern. It acts as an endpoint for the service (clients can only access the OGC service through the gateway). Its responsibility is to build the itinerary for the request, to attach this itinerary to the message and to forward the message according to the itinerary. Refer to the Technology Viewpoint, section 5.6, for details. The Gateway is deployed in a DMZ.

Biometric Authentication This component extracts the SAML Authentication Assertion from the header of the SOAP message and checks the digital signature of the assertion. If the signature is valid, the user's profile is retrieved from a database and the associated information is attached to the message in the form of annotations. Because in this project only a prototype was developed, the only information which is passed by means of annotations is the username. This service acts both a verification service (it verifies the user's credentials) and as an identification service - it supplies attributes related to the user's identity.

Authorization Because authorization was not the primal focus of this project, a simple authorization service was implemented, that only supports rules based on the username (i.e. "user Bob is

/ isn't authorized"). The implementation reads the annotation generated by the authentication service, extracts the username and takes the authorization decision based on the information stored in the policy store.

Audit The message contains enough information to generate proper audit trails (see figure 6.8). However, only a prototype implementation was realized for the audit service which stores the messages, as received, in a local log file. More advanced implementations could further process the message and only log relevant information.

SOAP-HTTP Converter Because the OGC services used do not implement SOAP messaging, a SOAP-HTTP protocol converter was required. Because multiple profiles are defined for each OGC specification and because most implementations only implement some of the defined profiles, it is not possible to implement a general purpose converter. In the project a custom converter was developed which converts the WMS SOAP profile to WMS HTTP GET.

Legacy OGC Service The implementation was only tested with an OGC WMS service.

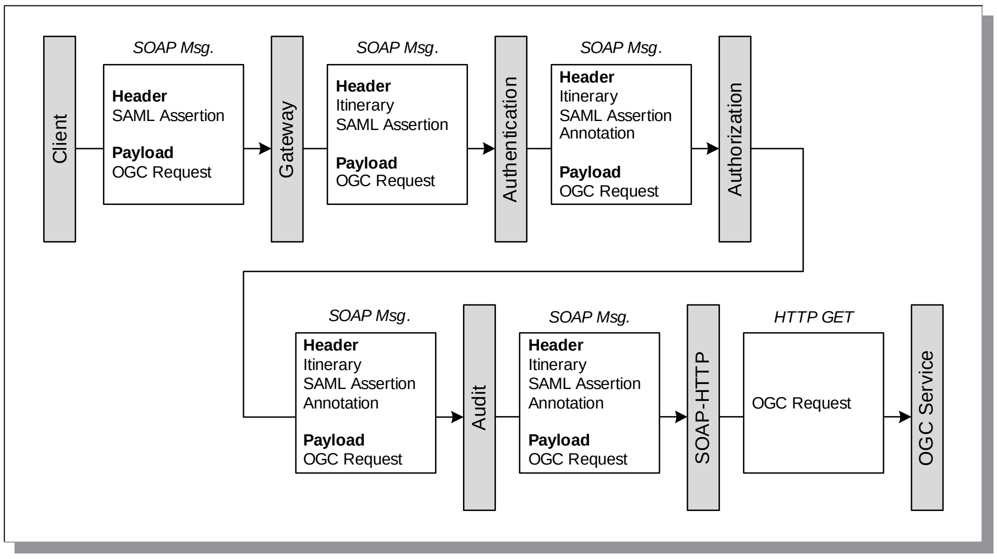

Figure 6.8: Biometric authentication for OGC Web services: Message propagation in the framework

Design Analysis

The design of the security system adopts the ***SOSA*** principles: the security system is external to the protected Web service and is composed of several Web services which are aggregated by means of message routing. The security services resemble the ones described in the taxonomy presented in the Computational Viewpoint and the security tasks are realized into services according to the principles of separation of concerns: authentication, authorization and audit are realized as different services. This leads to a clear and understandable design and has the obvious advantage that services can be replaced with little configuration effort (for example instead of the simple audit service, a more complex audit implementation can be deployed).

Because only very simple access control rules are supported, the authorization task was implemented into a single service. Dividing this task into PDP, PEP and PIP services (as described in the Computational Viewpoint) was not feasible in this case. A more complex authorization

implementation is described in section 6.5.
Finally, it should be remarked that this project illustrates very good how the Authentication-Authorization-Audit security services chain can be realized by means of ***SOS!e*** .

6.4 Cryptography for OGC Web Services

6.4.1 Context

Encryption and digital signatures are common security tasks. For XML communication, as is the case of SOAP Web services, these operations are supported through the XML Encryption [160], XML Digital Signatures [159] and the WS-Security [35] specifications. For messages containing non-XML content they are supported through the SOAP with Attachment (SwA) [36] profile of WS-Security.
As described in 2.4, through encryption and digital signatures integrity, confidentiality and non-repudiation can be achieved. There are various application scenarios where OGC services are required to implement these functions; one of them, involving a land register information system, will be described in this section.
OGC Web service interfaces do not have built in support for cryptographic operations. A simple and easy to deploy solution to address this problem is though SSL, but this has several disadvantages, most of them related to the fact that SSL works at a lower layer of the network stack: SSL implements only encryption (it is not possible to have the content digitally signed), SSL only works for point-to-point message transfers, not end-to-end, and SSL does not allow selective encryption of the content (the whole payload must be encrypted).
In this project, two security services were realized that allow SOAP messages to be digitally signed and encrypted by leveraging the specifications mentioned in the beginning of this section. The implementation is documented in detail in [144]. One possible application example for these two components in the context of security for geospatial Web services involves land register information (because of data protection laws). This scenario is described in the next section.

6.4.2 Problem Statement and Objectives

Scenario

A land register information system, containing information about land parcels and ownership, must be deployed in the Internet. The system exposes a Web Feature Service (WFS) interface through which users can query for information about the land parcels and their owners. The WFS returns vector information encoded in GML. It is assumed that the service is *read-only* (not WFS-T), in other words, the database can not be modified through the exposed service interface.
Because it contains personal information, land register information is subject to privacy regulations in many countries ([64] elaborates on this topic). Such information may only be accessed by authorized persons and the information should be protected against unauthorized access (both read and modification) while in transit over the network. Furthermore, if the data is stored or cached on the client machine, the same security measures should be applied.

Objectives

The main objective of this project are to extend the ***SOS!e 1.0*** framework by developing two security services that implement XML encryption and XML digital signatures. The two services should accommodate general purpose SOAP messaging and should have a high degree of reusability. In the following, I will describe the most important aspects related to the two cryptographic services and will also show how a security system based on the ***SOS!e 1.0*** framework can be deployed and configured to handle the various security aspects involved in the land register information system scenario described above.

6.4.3 Requirements Analysis

Requirements for the Two Cryptographic Security Services

Because the two security services should have a high degree of reusability, the following requirements render as important:

- The security services should be able to process general purpose XML messages. Because the application scenario involves a WFS service, these services should be tested in combination with WFS response messages.
- The services should permit that only some parts of the message content be encrypted / digitally signed. Because messages are XML, the desired parts can be specified by means of XPath expressions.
- Because these services may be used by more than one protected service it should be possible to specify the cryptographic keys and the content to be encrypted through parameters. This will allow the two services to be shared, without the need to replicate them (as in the case of the Service Sharing Pattern - Technology Viewpoint, section 5.6).
- The services should support several encryption and hashing algorithms. Both symmetric and asymmetric encryption should be supported.

Security Requirements for the Land Register Information System Scenario

From the scenario above, considering the security requirements described in section 2.4, the following security issues render as important and should be taken into consideration in the system design:

Authentication Because the service is not available for everybody, the identity of service requesters must be verified. The identity of the requester is important for both authorization and audit.

Authorization Because not all users should have access to the whole database, some authorization process must be implemented in the system. The exact authorization process is out of the scope of this dissertation; usually access to land register information is regulated by national laws.

Audit The system should maintain audit trails documenting for each user both what the user has requested as well as the information generated by the WFS.

Integrity The service must guarantee that the information arrives at the client without being altered (accidentally or maliciously) when travelling over the network. One way to enforce this is by digitally signing the content of messages before dispatching them.

Confidentiality Because the ownership information is subject to privacy laws, this information should be encrypted while in transit over the external network. However, other kinds of data that the service may return, such as for example information about the land parcels, need not be encrypted because this information is not subject to privacy regulations.

Non-Repudiation Proof of origin is important in this scenario. This can be realized through digital signatures: the service signs the content of response messages before dispatching them.

6.4.4 Design and Implementation

Implementation

The two services were implemented on top of the ***SOS!e 1.0*** framework. The implementation is based on two open source libraries from Apache: *XML Security* - the Java implementation of XMLENC and XMLDS, and *WSS4J* - the Java implementation of WS-Security.

Configuration

In order to avoid service replication, the two services are built such that parameters are expected in the message to be processed, leveraging the parametrization features of the ***SOS!e 1.0*** framework (for details refer to section 5.6.2). In this way the Gateway service can specify the service parameters when building the message itinerary.
Listing 6.4 illustrates the configuration parameters for an encryption service (the digital signature service is configured in a similar fashion). The first parameter contains a list of XPath expressions that identify the nodes to be encrypted, the second parameter specifies the configuration file containing the encryption parameters and the key store location, while the last parameter indicates the key to be used in the encryption process.

```
<Hop>
  <Address>http://some.url/services/enc</Address>
  <Service>
   <ClassName>de.unibw.s3f.services.encryption.EncryptionService</ClassName>
   <Parameter>
    <Name>ElementsToEncrypt</Name>
    <Value>Xpath1;Xpath2;...      </Value>
   </Parameter>
   <Parameter>
    <Name>CryptoPropertiesFileName</Name>
    <Value>crypto_test.properties</Value>
   </Parameter>
   <Parameter>
    <Name>UserName</Name>
    <Value>tom_test</Value>
   </Parameter>
  </Service>
</Hop>
```

Listing 6.4: Service configuration example

Listing 6.5 shows an example XPath expression that matches features of type Kataster[5] in a WFS SOAP message response. Specifying this in the configuration would result in the land register features being encrypted.

[5]Kataster is the German word for land register.

```
/soapenv:Envelope/soapenv:Body/gmgml:featureCollection/gml:featureMember/gmgml:Kataster
```

Listing 6.5: Example XPath expression matching Kataster feature types in a SOAP message

Wiring Diagram for the Land Register Information System Scenario

The wiring diagram for the land register information system scenario is presented in figure 6.9. It is similar to the wiring diagrams presented in the previous sections of this chapter. Some of the components - Gateway, Authorization, Audit and SOAP-HTTP Converter are the same components as in the case of the biometric authentication system - this demonstrates the reusability of the ***SOS!e*** security services.
The different components are the authentication service - here, instead of the biometric authentication a service supporting username - password authentication is used, and the two cryptographic services.

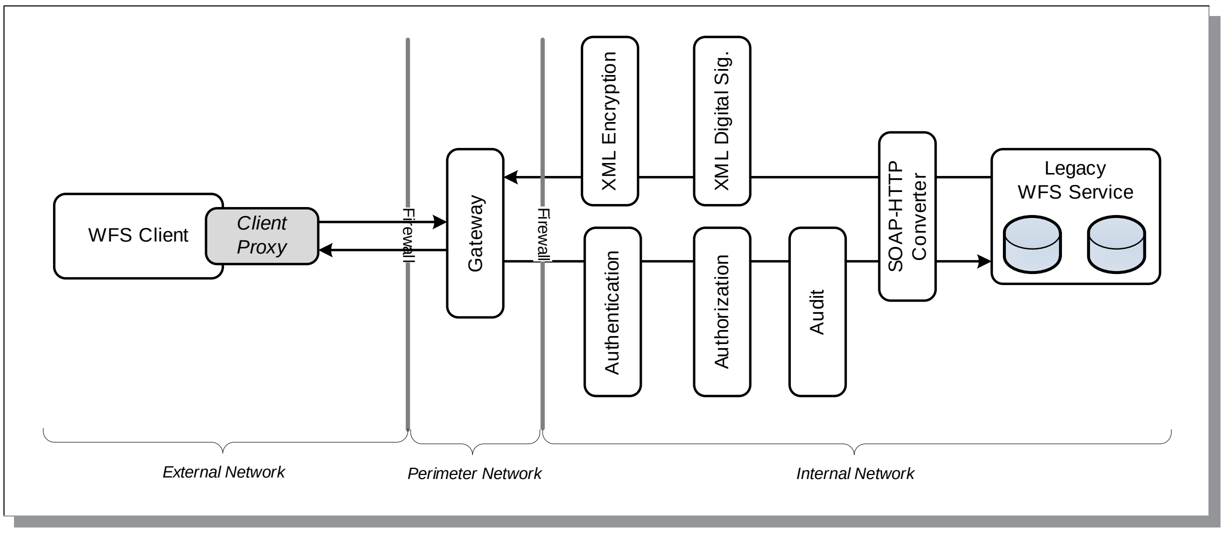

Figure 6.9: Cryptography for OGC Web services: Security system design

The message flow is similar to the one showed in the previous section. However, here several services are introduced on the message return chain: the request is first authenticated, authorized and audited, then the corresponding response is generated. After that, the response message is digitally signed and finally encrypted. For digital signing the private key of the service will be used, while for encryption the public key of the requester will be used.
Example SOAP messages illustrating a GetFeature request containing the authentication credentials and two responses containing GML feature collections where only the Kataster features are encrypted / digitally signed are showed in appendix A.1.

Design Variations

Because of the up-mentioned data privacy regulations, it might not always be possible to perform the authorization process without the intervention of a human person. For this, the *Asynchronous Authorization Pattern* described in the Engineering Viewpoint (section 5.5) may provide a solution. In this case, the authorization service takes the form of an interface where a human person inspects the requests and then manually accepts or rejects them. The messages are persistently stored at the service until the human person performs the authorization task, and only then forwarded along the processing chain.

6.5 Authentication and Licensing for WFS-T

6.5.1 Context

As part of the GeoDRM thread of the OGC Web Services Testbed 4 activity, the University of the German Armed Forces has participated in the design and implementation of licensed access to geospatial information. The results of this project are documented in [68, 65, 79], have been presented within the 59th OGC TC Meeting and are available online, both on the OGC's website[6] and on the UniBw's website[7].

In the following I will briefly describe different aspects of the project. Similar to the previous sections of this chapter, the emphasis will be on how ***SOSA*** was applied in the solution design and how the ***SOS!e 1.0*** framework was leveraged for the implementation. Other issues, such as proposed extensions to the WMS/WFS specifications or details about the implementation are intentionally left out. For these, please refer to the references provided above.

6.5.2 Problem Statement and Objectives

The exact objectives of the project together with detailed requirements, use cases, deliverables and component diagrams are detailed in [123]. This subsection only presents an overview.

The objective of the GeoDRM thread within OWS-4 was to develop the engineering concepts and a prototype implementation for licensing of geospatial information. The GeoDRM Reference Model [149] lays the foundation in this field by specifying the required infrastructure from the enterprise, computational and informational perspectives. In OWS-4, a corresponding engineering viewpoint had to be developed and, based on this one, prototype implementations for both the WMS and the WFS had to be built. Furthermore, a corresponding trust model had to be built that addresses the service protocol, service request, chaining with other services and service response required to form a complete trusted services chain.

In the following only one of the four given use-cases will be discussed[8]. This use-case involves a client making a request to a transactional feature service (WFS-T). In order for the transaction to be executed, the identity of the requester needs to be verified and the request needs to be authorized based on the requester's identity and its permissions (it is assumed that a license containing the requester's permissions is presented to the service). The service must keep audit trails documenting the requesters and the operations executed. Furthermore, in the case of transactions (insert, update or delete operations), the service should send notifications announcing that modifications were performed on the dataset.

6.5.3 Requirements Analysis

Interactions and Trust Model

The interactions between the OGC client and the other entities are described in [68] and illustrated in figure 6.10. As it can be seen here, the client is acquiring an identity token from an Identity Provider and one or more license tokens from the License Broker. For the acquisition of these tokens

[6]http://www.opengeospatial.org/projects/initiatives/ows-4
[7]http://iisdemo.informatik.unibw-muenchen.de/ows4
[8]This was the use-case implemented by UniBw; the implementation was based on the ***SOS!e*** framework.

the mechanisms described in WS-Trust [40] can be used. However this process is besides the scope of this dissertation - we will only focus on how the ***SOS!e*** framework was leveraged in order to enforce the security policy of the GeoDRM Enabled WFS-T.
Having the tokens described above, the client is now able to make requests to the GeoDRM Enabled WFS-T. For this, he will create a request message containing the WFS-T operation request as payload, attach the tokens to the message (one identity token asserting its identity and one or more license tokens asserting its permissions) and then send it to the service.

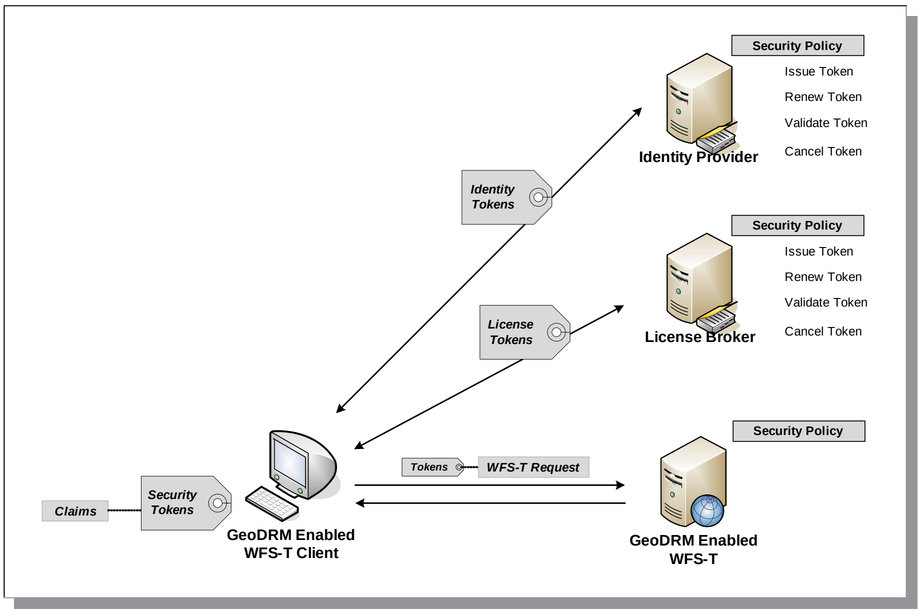

Figure 6.10: Authentication and licensing for WFS-T: Trust model [68]

The two types of tokens contain the following information:

Identity Tokens Contain information asserting identity together with information about the issuing authority (Identity Provider). Examples of such tokens are X.509 certificates and SAML assertions.

License Tokens Contain information asserting the permissions of a certain subject and information about the issuing authority. The subject must prove its identity when presenting a license token. In OWS-4, license tokens were implemented as SAML assertions.

Verification Procedure and Security Requirements

Once the request message reaches the service, this one will have to perform the authentication and authorization steps. The procedure is graphically displayed in figure 6.11. Looking at this figure, we notice that the procedure can be split into two distinct activities:

Verification The validity of the tokens is verified - this corresponds to steps 1 - 5 in figure 6.11.

Authorization The permissions asserted by the license tokens are evaluated against the WFS-T request - step 6 in figure 6.11.

In addition to these two activities, the service should also perform audit tasks, as follows:

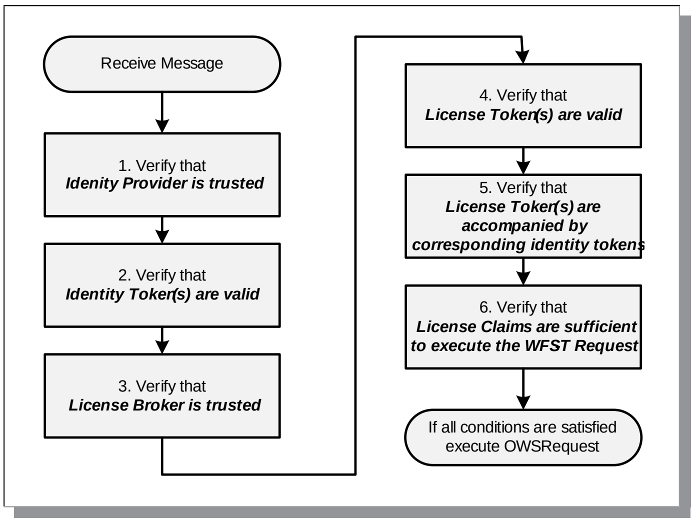
Receive Message
1. Verify that Idenity Provider is trusted
2. Verify that Identity Token(s) are valid
3. Verify that License Broker is trusted
4. Verify that License Token(s) are valid
5. Verify that License Token(s) are accompanied by corresponding identity tokens
6. Verify that License Claims are sufficient to execute the WFST Request
If all conditions are satisfied execute OWSRequest

Figure 6.11: Authentication and licensing for WFS-T: Verification process [68]

Logging Audit trails should be left documenting, for each user both the requests executed and the information returned. The latter is required because the WFS-T data store may change over time, therefore from the request alone it is not possible to determine after-the-fact the corresponding response.

Alerting If one of the operations executed was a transaction, an alert must be generated. In this project, the alerts were email messages sent to an administrator; upon receiving an email, this one would manually inspect the changes.

6.5.4 Design and Implementation

The security system for the GeoDRM Enabled WFS-T was designed as a list of components that were assembled together by means of the ***SOS!e 1.0*** framework. The principle of separation of concerns was applied and the functionality of the components reflects the previously identified requirements.

Wiring Diagram

The system design is presented in figure 6.12. Here, the main components, data stores and inter-component communications are showed. The relevant components are described below.

Verification Verifies both the identity tokens and the license tokens. The OWS-4 implementation was able to verify two types of identity tokens: username-password tokens and X.509 certificates accompanied by a corresponding digital signature; for the latter, an extension to the client was developed that allowed the use of smartcards and cryptographic tokens (for more, refer to the demo website). The Verification service was also able to verify license tokens encoded as SAML assertions. If the verifications are successful, this component also annotates messages with a unique user identifier that is used as a *canonical requester identifier* (see section 5.3, Identification Service). This identifier is used by the rest of the security services (PEP, Logging and MailAlert) to identify the requester.

PEP For the authorization process the GeoXACML implementation developed in [107] was leveraged. For every request, the enforcement point communicates with the GeoPDP to determine

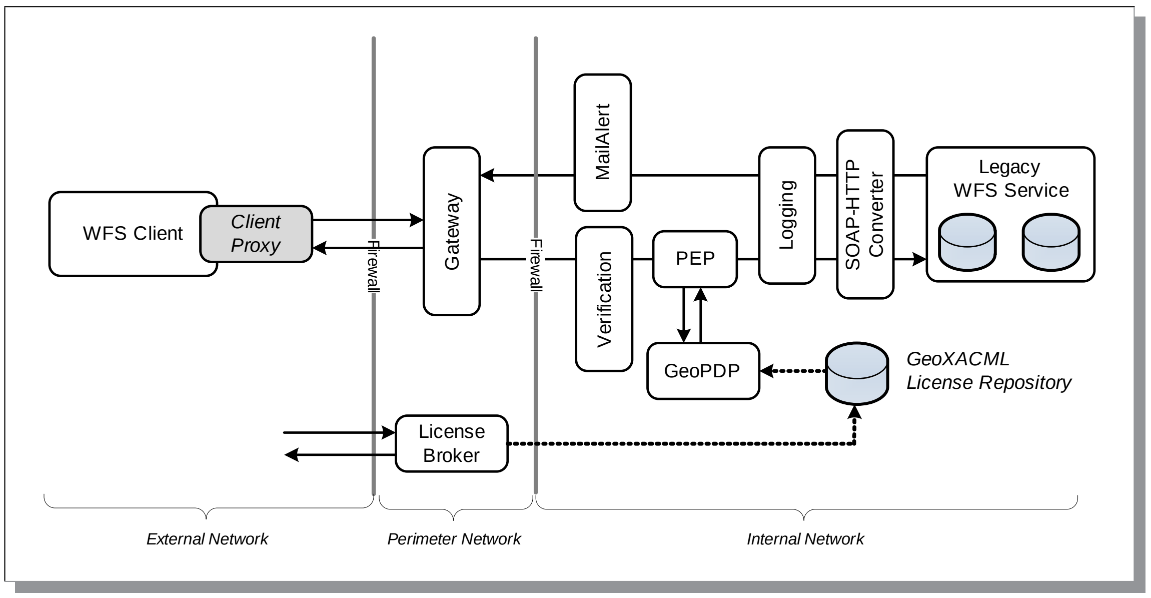

Figure 6.12: Authentication and licensing for WFS-T: System design

if the request should be permitted or denied. For this, the PEP implements the XACML request-response protocol, as described in [13].

GeoPDP This component is part of the GeoXACML implementation (see [107]) and only minor implementation modifications were performed. It is able to perform authorization based on different criteria, including the WFS-T request type and its geographical boundaries.

Logging Logs both requests and response messages. Because the message is annotated with the canonical requester identifier, both the request and the corresponding response can be traced back to the requester.

MailAlert A general purpose alert service was developed for the notification task. The alerts are specified as XPath expressions which are evaluated for all messages received. In this project, the MailAlert service was configured to send emails if a transaction operation was performed (for WFS-T, this can also be determined by investigating the response message).

License Broker This component was realized as a web interface. By accessing it, new licenses can be generated; the corresponding license token is returned to the client and the GeoXACML License Repository is updated.

Design Analysis

In contrast to the previous three projects where the authorization was realized as a single service, here it is divided into PDP and PEP. However, the PDP is not realized as a mediation services, but rather as a service which is invoked by the PEP. The reason for this choice, is that the existing GeoXACML implementation was leveraged, and this implementation conforms to the XACML service model where the PDP is implemented as an invocation service.
Because it leverages existing security capabilities, this design is a good example of how existing security services can be integrated in the ***SOS!e*** framework and leveraged by means of ***SOSA*** .

Design Variations

The Federation Pattern The verification service used here is able to verify all the different kinds of security tokens. This fits the requirements of the OWS-4 project. However, in other cases it may not be possible to have a single service which verifies all types of tokens. For such situations the Federation Pattern (section 5.5) may provide solutions.

Notifications through Message Routing Figure 6.13 shows another possible design for implementing the notification workflow (only the return chain is showed in the figure). After the logging service, a message splitter is introduced in the chain: one route leads to the Gateway service and then back to the requester and the other to the Notification service. This latter one is implemented as a message filter[9]: the messages that do not fit the alert criteria are simply dropped. The other ones are sent to a persistent store which is connected asynchronously with an Application for Inspecting WFS-T Transactions.

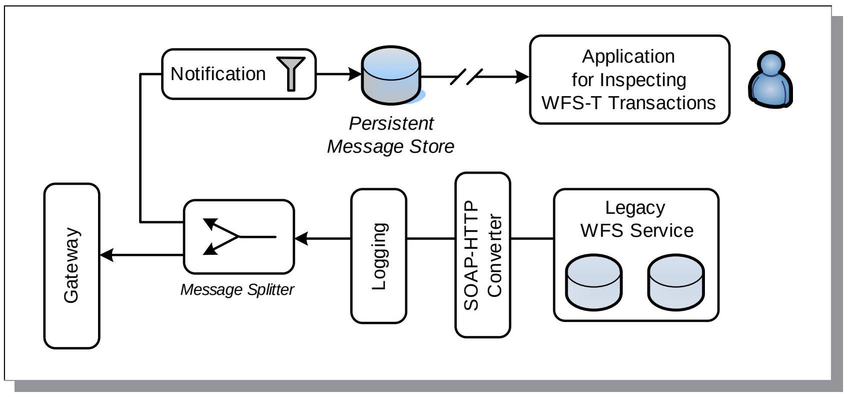

Figure 6.13: Authentication and licensing for WFS-T: System design variation

In ***SOS!e 1.0*** this is rather complicated to realize. However in ***SOS!e 2.0*** , which is based on an ESB, this is very easy to realize and does not require a single line of code: both message filters and splitters are supported through configuration scripts.

6.6 Further Practical Experiences

6.6.1 Kerberos Authentication Service

The purpose of this experiment was to develop a security service that is able to perform user authentication based on a provided Kerberos token. Because Kerberos is the default authentication protocol used in current versions of Microsoft Windows, such a security service is very useful as it leverages the authentication infrastructure already existing in Microsoft Networks. The experiment was performed as a study assignment at the University of the German Armed Forces and the results are described in [81].
In the following this experiment will be briefly introduced by presenting an application scenario and a couple of relevant aspects about the implementation. For further information about Kerberos and authentication in Windows networks, please refer to [61].

[9]This is one of the EAI patterns discussed in [95].

Application Scenario

When a user logs in a domain on a machine running Windows, the Kerberos authentication protocol is used and the machine will acquire a TGT[10] from the domain controller. This TGT can be retrieved by a client software, and a session ticket (ST) can be requested. With this ST, the client can make requests to a Web service, attaching the ticket to the request and using the ticket key to sign the message. On the server side, the developed Kerberos authentication service will test the validity of the token and the authenticity of the signature and, by connecting to the Active Directory service, retrieve different user attributes which are then passed to other security services through annotations.
Kerberos provides a simple authentication procedure on the client side and by means of the developed Kerberos authentication service, the ***SOS!e*** framework can be integrated with the existing Windows infrastructure for authentication and access control.

Implementation

SOAP communication was used and the Kerberos tokens were encoded by means of WS-Security. Because Microsoft .NET offers a very good API for Kerberos, both the Kerberos security service and the client were developed using .NET. The rest of the ***SOS!e*** framework, as described in section 5.6, is developed in Java. However, because SOAP and WS-Security are independent of both operating system and development environment, it is possible to integrate .NET services in the Java-based framework. Figure 6.14 illustrates the deployment.

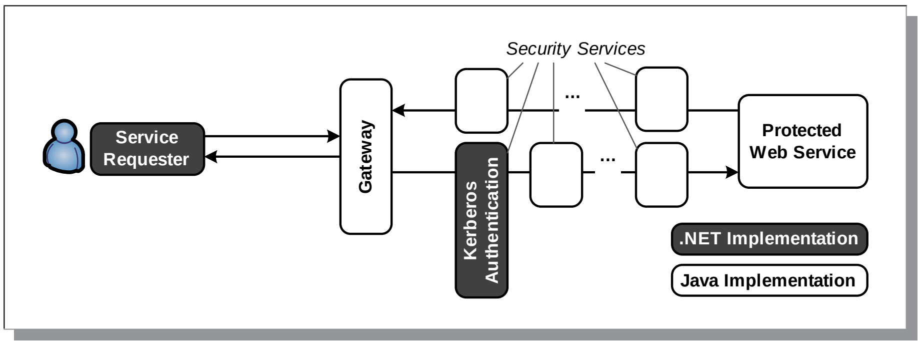

Figure 6.14: Deployment of the Kerberos authentication service

This demonstrates another application scenario for ***SOSA*** : instead of developing a monolithic security system where the developer is forced to use a single technology, different security aspects can be implemented as different security services. Most importantly, the developers are free to choose the *programming language* to implement these services and the target *deployment platform.*

6.6.2 Accounting and Charging with PayPal

In this experiment the objectives were to develop two security services: one accounting service implementing different accounting models for the WMS and WFS, and one charging service that is able to charge users by means of PayPal[11]. Putting together these two security services with the rest of the security services presented earlier in this chapter, would result in a complete A4C[12]

[10]Ticket Granting Ticket
[11]http://www.paypal.com
[12]A4C is an acronym for *Authentication, Authorization, Audit, Accounting and Charging*

security system, able to support complex eBusiness scenarios. These two services are developed as part of a diploma thesis and the results are described in [161]. In the following I will only present some relevant aspects of the two services.

The Accounting Service

The accounting service meters the usage of the service and produces information for the charging service. The metering is done independently for each user, and the metering data is both persistently stored as well as transmitted by means of annotations to the charging service.
Because a generic implementation can not be realized for service metering, it was decided to support two OGC services: WMS and WFS. A separate accounting service is developed for each service interface. The following accounting models are supported by each service (the models are inspired from [76, 152]):

- WMS Accounting Service

  Pay-Per-Pixel A fixed price is charged for each pixel of the requested image. The model can be refined to include different prices for different layers offered by the WMS.

  Pay-Per-Request A fixed price is charged for each request; this model is sometimes also referred to as pay-per-click.

- WFS Accounting Service

  Pay-Per-Feature A fixed price is charged for each feature returned; different prices can be set for different feature classes.

  Pay-Per-Request A fixed price is charged for each request (pay-per-click).

It is worth to remark that, in case of the WMS the information required to perform accounting is found in the request message, while in the case of the WFS this information is found in the response message. This has implications on how the security services are deployed: the WMS accounting service must be deployed on the forward chain, while the WFS accounting service must be deployed on the return chain.

The Charging Service

As discussed in the Computational Viewpoint (section 5.3), charging can be either done on a periodical basis (monthly, yearly, etc.) or immediately. The scope here was to realize a service that implements the latter case.
In order for the charging service to be able to perform its task it requires the following information:

- Information necessary to retrieve the charged amount (i.e. credit card) together with information regarding the address where the invoice shall be sent. This information is provided by the identification service (it is assumed that the charging information is saved together with the rest of the requester's profile).
- The amount of money that must be charged (and possibly also additional information that should be printed on the invoice). This is provided by the accounting service.

The implementation uses the PayPal SOAP API [98]. This permits developers to build custom applications and services that make use of corresponding services to the ones PayPal offers on its website.

6.7 Conclusions

In this chapter I showed how the ***SOS!e*** framework was used in several projects in order to enhance OGC Web services with different security features. This demonstrated the feasibility of the Service Oriented Security Architecture - ***SOSA*** - described in this dissertation. Before going to the next chapter, where the architecture is analyzed and evaluated, the following remarks are made:

Service Type	**Implemented Services**
Authentication	Username-Password Authentication X.509 (Smartcard) Authentication Biometric Authentication (BioLANCC) Kerberos Token Authentication
Authorization	Simple Authorization XACML Authorization
Audit	Logging Mail Alert
Cryptographic Service	XML Encryption XML Digital Signing
Accounting	WMS Accounting WFS Accounting
Charging	PayPal Charging

Table 6.1: Security services implemented for the ***SOS!e*** framework

- In the projects illustrated here, there were cases where the security services were not built strictly as described in the taxonomy presented in the Computational Viewpoint (section 5.3). In many cases, the functionality of several services was implemented as a single service. This shows that when designing a security system, the exact requirements should be taken into consideration. It does not always make sense to fragment the security implementation because as the route followed by a message gets longer, the round-trip-time increases (see next section). Furthermore, sometimes it is more complicated to implement certain security functions as independent services. Nevertheless the security services taxonomy provides a good reference model, as it clearly illustrates how the most common security functions can be implemented as services.
- The projects revealed different reasons for using ***SOSA*** :
  - The ability to have a modular, component based implementation that facilitates reusability; because reusability occurs at the service-level, no replication is necessary (see 6.3, 6.4, 6.5);
  - The ability to implement components in different programming languages and deploy them on different platforms (see 6.6.1);
  - The ability to quickly integrate already existing security infrastructure with newly developed Web services (see 6.3, 6.6.1, 6.6.2).
- The projects illustrated how some of the patterns described in the Engineering Viewpoint (section 5.5) can be applied in practice. In addition to what was already presented, having in mind the particular case of geospatial Web services, the following two patterns can be applied to all four projects (sections 6.2, 6.3, 6.4, 6.5):

  The Security Off-Shoring Pattern For most data providers, dealing with the security aspects related to their services is a hard job, because it requires highly qualified personnel.

By applying this pattern, some of the security aspects can be off-shored to specialized companies, permitting the data providers to focus on producing, maintaining and serving the data.

The Service Sharing Pattern Data providers usually deploy several services hosting different types of data (for scalability reasons), but having similar security policies. For such cases, the service sharing pattern provides an elegant solution - the management effort is lowered because of the centralization.

- The security services developed for the ***SOS!e*** framework are showed in table 6.1. As it can be seen, by combining them together complex security solutions can be developed. This is achieved with little effort as it only requires the configuration of an itinerary that incoming messages should follow and the configuration of the respective service instances.

Chapter 7

Analysis and Evaluation

In this chapter an evaluation of the proposed architecture is performed. First, the architecture is analyzed according to the properties defined in 3.2. Then a detailed performance evaluation is performed with the purpose of determining the impact that a ***SOSA*** based security system has on the performance of the service provider. The results are discussed at the end of the section.

7.1 Architectural Evaluation

In section 3.1, several properties that are used in the evaluation of architectural designs have been presented: performance, scalability, simplicity, changeability, portability and reliability. In section 3.2, three different design approaches to implementing security for Web services have been introduced and evaluated against the aforementioned properties: security system embedded in the application, security system embedded in the middleware and security system external to the application - the last one is the approach taken by the architecture proposed in this dissertation. In the following, I will evaluate ***SOSA*** against the up mentioned architectural properties.

Performance

In section 3.2 a preliminary evaluation of the performance of external security approaches was presented. This evaluation was not exact because it only took into consideration the design without measuring a concrete implementation. It was showed there that the performance of applications where the security system is implemented externally is lower than that of comparable applications where the security system is embedded in the application or in the middleware.
A more accurate measurement of the performance of the proposed architecture is presented in section 7.2. There, the ***SOS!e 2.0*** framework is evaluated in order to determine the effects of different service chaining configurations on network performance parameters (round-trip-time and throughput). At the end of section 7.2 I'm discussing whether ***SOSA*** is appropriate or not for different classes of applications.

Scalability

Scalability is the ability of a system to handle a growing amount of work in a graceful manner or to be readily enlarged. In ***SOSA*** the different security tasks that a security system must perform are split in different components that are implemented as services. These are independent from one another and communicate by means of messages. Therefore the same techniques that are applied in scaling Web services can be applied individually to each security service. The following are the most important aspects that ensure the scalability of the proposed architecture:

Load Balancing As described in the Engineering Viewpoint, section 5.5, service clustering techniques can be applied to the security services. As such, several services can be configured to appear as one single logical service which is capable of handling an increased load.

Specialized Hardware Some security functions (mostly cryptographic functions) are CPU-intensive. Specialized hardware is available that usually provides performance boosts of 10 to 100 times in the number of messages per second that can be processed. However, because they are relatively expensive, it is difficult to equip each server which such hardware. By means of ***SOSA*** these CPU-intensive tasks can be distributed on machines with specialized hardware, and be used by services located on other machines.

Simplicity

Simplicity is the property of being simple. The factors that contribute to simplicity are *Complexity*, *Understandability* and *Verifiability*.

Complexity *is a measure of the number of inter-related parts of a system.* There are both positive and negative aspects of this property in relation with ***SOSA*** :

Negative On one side, the fact that the security tasks are divided into components (these are by their nature inter-related), increases the complexity of the system to some degree. This has effects also on the understandability and the verifiability of the system, as there is some overhead in understanding how the components are coupled together and some overhead in verifying that the components are correctly coupled together.

Positive Nevertheless, because there is a clear separation between the components (they communicated thought a messaging interface) the inter-connections are actually loosely-couplings. Furthermore, as showed in chapter 6, in normal cases it does not make sense to introduce a large number of security services.

In conclusion, complexity is increased to some degree, but this is not significant. Other approaches, as showed in section 3.2, although they may have less well-defined components, the inter-connections are a lot stronger.

Understandability *is a measure of how easy a person can understand a system.* This is a strong point of the proposed architecture. The following two factors contribute to understandability:

- The different security tasks are divided into well-defined components that are implemented as different services. These components are therefore separated at the implementation level - they only communicate by means of a messaging interface.
- Security systems are designed by applying the principle of separation of concerns. This separates the components at a conceptual level: different tasks are implemented into different components.

As previously mentioned, there is some overhead due to the aggregation of the different security services. This is low if graphical tools for the definition and visualization of orchestrations are used.

Verifiability *is a measure of how easy it is possible to prove or disprove the correctness of a system.* The system is a sum of well defined components, which can be individually tested. Security services have clearly defined interfaces (though WSDL or similar technologies) and usually do not implement complex functions. For this reasons it is relatively easy to develop automatic tests for the them. However it is not so trivial to develop automatic tests for the whole system - these tests should automatically prove or disprove the correctness of the choreography. For this, WS-CDL could provide some solutions, as tools could be developed that perform automatic testing of the WS-CDL representation of the service choreography (for more, see 5.4 and 8.4).

Changeability

Changeability represents the capability of a system to be changed without a negative impact. The factors that contribute to this property are: *Configurability*, *Extensibility* and *Reusability.*

Configurability *Represents the ability of a system to support post-deployment changes in behavior through configuration changes.* Configuration can be programed in the security system at both service level (each service can have its own configuration) as well as at the system level (the service choreography).

Individual Services Configuration of the individual services is possible via the management interface (this was described in the Computational Viewpoint, section 5.3). There are various possibilities to implement this depending on the platform and middleware on top of which the security services are implemented. Examples in the context of Java are JMX[1], deployment descriptors, configuration files, etc. The services developed on top of the ***SOS!e*** framework (see chapter 6) allow for different parameters to be specified through configuration files.

Service Orchestration The orchestration of services is described by message routing rules. These are also configurable (they are usually stored in files or other types of persistent stores).

In conclusion, the architecture permits the design of configurable systems. However, the degree of configurability depends on the concrete design and the implementation.

Extensibility *Refers to the ability to add functionality to an existing system without having to make major changes to the system infrastructure.* Because the system is composed from several services that communicate through messages, it is easy to extend a system by simply adding new services or by replacing (upgrading) existing ones. Furthermore, it is possible to do this both offline (by temporarily stopping the system) as well as online (without stopping the system). For the latter one, techniques similar to the ones used when upgrading Web servers can be applied, where traffic is temporarily redirected.

Reusability *Refers to the ability to use components of a software system into other applications, without modifying them.* This aspect represents another strong point of the proposed architecture: security services can be reused both through redeployment (the security service is deployed in a different network and reconfigured) or by sharing a single service instance between multiple

[1]Java Management Extensions, see http://java.sun.com/javase/technologies/core/mntr-mgmt/javamanagement.

applications. The latter was illustrated in the Service Sharing Pattern and the Security Off-Shoring Pattern (Engineering Viewpoint, section 5.5).

Portability

Portability is the property of a system of being able to run on multiple platforms. Web services are designed to be platform-independent in the sense that the service requester and service provider can run on different platforms. Web services are nevertheless often bound to the middleware system.
SOSA applies the external security approach, dividing the protected service and the security system. This gives more flexibility with regard to portability as the two can be deployed on different platforms. Moreover, in ***SOSA*** the security system is realized as a series of security services, each of which can run on independent platforms.

Reliability

Reliability is the ability of a system or component to perform its required functions under stated conditions for a specified period of time. In the context of distributed systems, this is often seen as the degree to which a system is susceptible to failure at the system level due to partial failure of its components.
The proposed architecture introduces several interdependent components - the failure of one of them leads to a failure of the system because the security policy can not be applied. However, as showed in the Engineering Viewpoint, section 5.5, service clustering techniques can be applied which increase the availability of individual services and through this the availability of the system as a whole. Furthermore, because services communicate via messages, upgrades and maintenance for individual services can be dynamically executed (with the system running) by redirecting the traffic.

7.2 Performance Evaluation

In this section a performance evaluation of the proposed architecture is presented. As reference implementation the ***SOS!e 2.0*** framework is used. The goal is to determine the effects of different deployments of the ***SOS!e*** framework on the performance of the service provider. Two performance parameters are measured: the *throughput* and the *message round-trip-time* (message delay). The results are commented at the end of the section.

7.2.1 Performance Parameters

As already discussed in section 3.1, performance parameters can be divided into two distinct categories:

Network Performance Is a measure of the communication between the client and the Web service. The most important aspects that influence network performance are the network QoS parameters and the implementation of the service components.

User-Perceived Performance Represents the impact that the network performance has on the user. This type of performance is very much dependent on how the client application is built.

Because the proposed architecture refers to securing Web services (it does not have implications on the architecture of the client applications), only network performance will be analyzed here.

According to [143], the most important parameters for network performance in the context of Web services are the following:

Throughput Represents the number of Web service requests served in a given period of time.

Latency Also known as *message delay* or *round-trip-time*[2] or , is the time elapsed between sending a request and receiving the response.

Execution Time Is the time taken by a Web service to process its sequence of activities.

Transaction Time Represents the period of time that passes while the Web service is completing one complete transaction.

As a rule, higher throughput, lower latency, lower execution times and faster transaction times represent good performing Web services.
As far as the performance evaluation for ***SOSA*** is concerned, only the first two parameters are taken into consideration in the tests. The reason for this choice is that the *execution time* and the *transaction time* are included in the *latency*. Furthermore, *Latency* and *Throughput* give a good indication of the performance, and are used in most Web service performance analysis [155, 146, 58].

7.2.2 Performance Model for Web Services

In a Web service interaction, the service requester and the service provider exchange information using SOAP. In the following subsections, I will first present a model where the ***SOS!e*** framework is not considered (the service requester communicates directly with the service provider). This model is inspired from [155]. Afterward, this model will be extended to include the ***SOS!e*** framework as an intermediary between the service requester and the service provider.
The goal is to first understand where time is spent in a typical Web service communication and then to understand the delays introduced by the ***SOS!e*** framework. Knowing these, a couple of simple scenarios are proposed, which are representative and can be used in the performance evaluation.

Service Requester Communicates Directly with the Service Provider

In a request-reply interaction, the requester will first construct the request message (converting the parameters from its own in-memory representation into XML) and send the request to the service provider. This one will parse the request, convert the parameters into its native in-memory representation, process the request, then construct the response message and return this one to the requester. The requester will receive the response and deserialize it. Therefore, the factors that influence the RTT are the following (they are graphically illustrated in figure 7.1):

1. Time spent by the Web Service Client (WSC) to retrieve the values from its internal data structures.
2. Time spent by the WSC to construct a message by embedding the previously retrieved values.
3. Time spent by the WSC to dispatch the message.
4. Time spent by the message to arrive at the Web Service Server (WSS).
5. Time spent by the WSS to read the incoming message.
6. Time spent by the WSS to parse the message and retrieve the required data.

[2]Round-Trip-Time is abbreviated RTT in the following.

7. Time spent by the WSS to construct appropriate data structures that can be manipulated in memory.

8. Time spent by the WSS to complete the execution of the request.

9. Time spent by the WSS to retrieve the values for the response message from its internal data structures.

10. Time spent by the WSS to construct the response message by embedding the previously retrieved values.

11. Time spent by the WSS to dispatch the response message.

12. Time spent by the response message to arrive at the client.

13. Time spent by the WSC to read the incoming message.

14. Time spent by the WSC to parse the response message and retrieve the required data.

15. Time spent by the WSC to construct suitable data structures and populate them so that they can be used in execution.

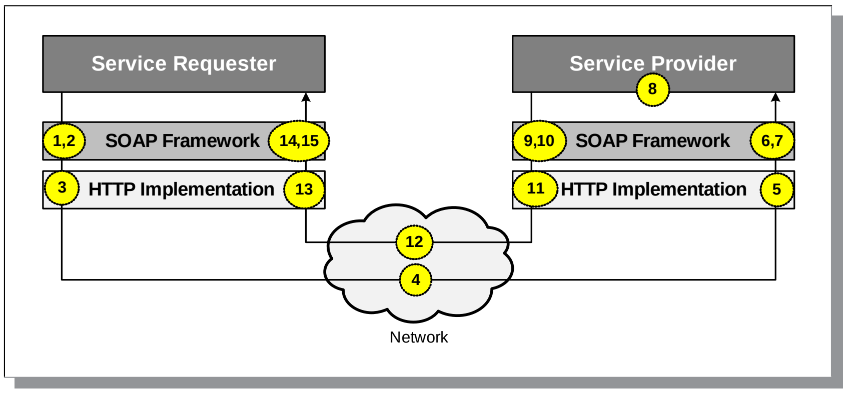

Figure 7.1: Web services performance model

Of these fifteen factors, factors 1, 2, 3 and 13, 14, 15 depend on the implementation of the client. Factors 4 and 12 depend on the performance of the network. Factors 5 and 11 depend on the HTTP protocol implementation (SOAP-over-HTTP is considered in this analysis). Factors 6, 7 and 9, 10 depend on the implementation of the SOAP framework. Finally, 8 represents the time taken by the Web service to complete execution, which is dependent on the implementation of the Web service.

Service Requester, SOS!e and Service Provider

If the ***SOS!e*** framework is deployed between the service requester and the service provider (figure 7.2), steps 4 - 12 are repeated for each intermediary security service introduced along the message itinerary. However, for each security service, step 8 described above can be split into the following two sub-steps:

8a. Time spent by the security service to execute the security operations.

8b. Time spent by the security service because of the imposed processing model. This represents the time spent by processing annotations (creation, deletion, modification).

If the security policy is to be enforced (i.e. there is a security system in place that is realized either embedded in the application or embedded in the middleware), step 8a is anyway executed (i.e. parts of the message are encrypted, authentication tokens must be verified, etc). Therefore, the overhead introduced by the ***SOS!e*** framework is caused by factors 4 - 7, 8b and 9 - 12.

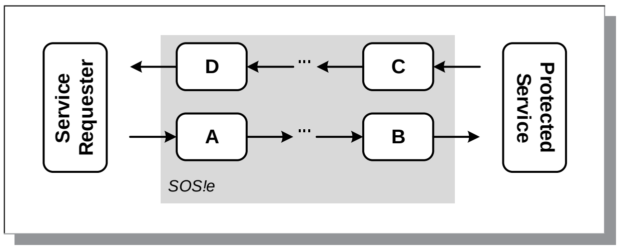

Figure 7.2: ***SOS!e*** deployment

Use-cases for Evaluation

In order to accurately measure the overhead introduced by the ***SOS!e*** framework, time spent in step 8a must be null. We therefore consider "empty" services in the evaluation, which only implement framework specific operations. The following types of services will be used in the evaluation:

Simple A service that does nothing (it simply sends the message to the next service along the chain). This service simulates the framework-specific overhead in services that do not need to communicate with other services. Examples of such security services are: logging, encryption, PEP.

Simple with Annotation A service that reads the annotations found in the message and appends a new annotation to the message before forwarding it. This service simulates the framework-specific overhead in services that do communicate with other services. Examples of such services are: identification, PDP, PIP.

Simple with Digitally Signed Annotation ***SOS!e 2.0*** supports annotations containing digital signatures. This service is similar with the previous one, just that it verifies the signature of the annotations found in the message and signs the newly created annotation.

7.2.3 Performance Testing with a Lightweight Environment

There are various approaches to performance testing [155, 146, 108, 58]. The approach taken in the evaluation of ***SOSA*** was to build a lightweight environment that can be realized in a laboratory. The software used was open-source and the servers were, as it will later be seen, equivalent to medium-size PCs. This kind of environment is inexpensive and can be realized by anybody that is willing to deploy a ***SOSA*** solution, and is interested to do an early capacity testing. The approach is inspired from [146].

The test environment must be built in such a manner that the load is distributed mostly on the security system (after all, this is the component whose performance needs to be determined). If other components represent bottlenecks it is not possible to accurately determine the performance of the security system.

Figure 7.3 presents a typical deployment, where a service requester connects though the ***SOS!e*** framework to a service provider. As is often the case, the service provider relies on one or more back-end databases. Furthermore, the ***SOS!e*** framework also connects to databases where security information is persistently stored (examples are LDAP repositories, XACML PAPs, etc).

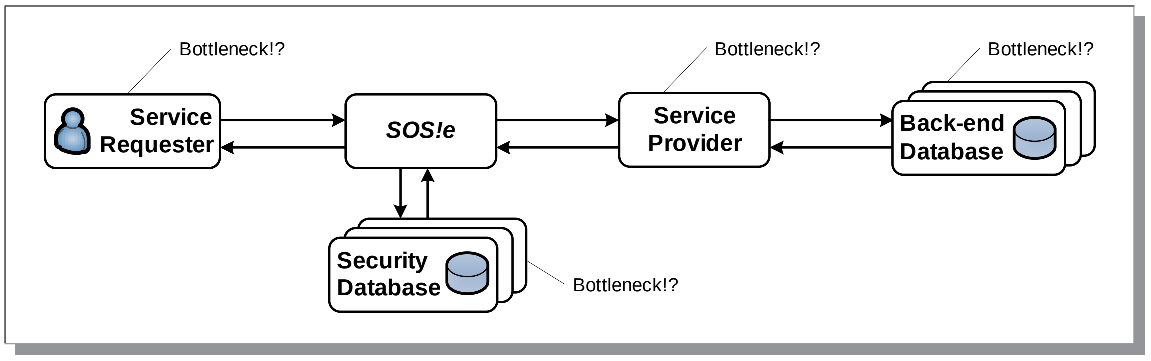

Figure 7.3: Possible bottlenecks in the testing environment

The testing environment must be built in such a way that the only bottleneck is the ***SOS!e*** framework. If any of the other components represent bottlenecks or if significant time is consumed by those components, the measurements would be inaccurate. In order to minimize the effects that these components have on the performance parameters (message delay and throughput), lightweight versions of these components were built:

Service Requester The task of this component is to generate requests and send them to the service provider. This component was realized as a workload generator. In order to provide a good load on the service provider, the generator can be configured to use several parallel threads, simulating several requesters concurrently accessing the service.

Security Database The security services used in the testing environment have been presented in section 7.2.2. They do not connect to external databases.

Service Provider A minimal implementation of the service provider was realized. This resembles a "Hello World" application: it waits for a request message, it modifies a tag from the body of the message and responds with this modified version of the original message. In this way the request and the response messages have comparable complexities. No back-end database is accessed.

7.2.4 Realization of the Environment

Service Requester

The service requester was realized as a workload generator. Apache JMeter[3] [99] was chosen for this purpose because it is an open-source tool with multiple configuration possibilities. JMeter is also able to measure both throughput and RTT and to compute several types of statistics.
In order to boost performance (JMeter consumes a lot of CPU, especially if it is configured to simulate large numbers of clients), the requests were not generated on the fly. Instead, the request messages were prepared in advanced as files.

Service Provider

The service provider was implemented in the same way as the other Web services. It was developed in Java and deployed in Apache AXIS (this one runs inside Apache Tomcat).

[3]`http://jakarta.apache.org/jmeter`

The SOS!e Framework

Different configurations of the ***SOS!e*** framework were tested. The following parameters were varied:

Deployment Type Different deployments of the security services and different configurations of Mule ESB were tested. These are detailed in the next section.

Security Services Type The types of security services used in the testing environment have already been presented in section 7.2.2. These are: Simple, Simple with Annotations, Simple with Digitally Signed Annotations.

Number of Security Services The number of security services that a message encounters on its itinerary ranged from one to five. This simulates different levels of complexity for the security system.

Configuration Parameters

In the testing environment, the following parameters could be configured:

Number of Threads This is a measure of the number of clients simultaneously accessing the service provider. In order to have an optimal load for the service provider, three different values were tested: 3, 30 and 100. We expected this to impact on both the RTT and throughput. A lower number of concurrent clients is expected to result in a lower message RTT because messages wait less in order to be processed. However, a higher number of concurrent clients is expected to produce more accurate throughput results as the service is better solicited.

Message As far as the transacted messages are concerned, the goal was to use real-world messages: OGC service requests and responses were used in the tests. The size of the messages ranged from 1KB (a WMS GetCapabilites request) to 64KB (a WFS GetFeature response containing 38 features).

Number of Messages As also described in [108], it is important that the number of messages is high enough so that the tested system reaches a stable state. Between 15.000 and 50.000 message requests were used for each test - the number was roughly calculated so that each test takes between 3 and 15 minutes. In most cases, the system reached a stable state after one minute.

Monitored Parameters

It is common practice [146, 108, 99] that all the machines which are part of the testing environment be monitored during the performance test. In order to obtain accurate results it is important that load on both the service requester machine and the service provider machine be low and the load on the machines running the ***SOS!e*** framework be high. In order to ensure that this is the case, the CPU usage was monitored on all machines during the tests.

Hardware

The tests were executed on eight Dell PowerEdge 750 [4] machines (2.8GHz Intel Pentium 4 Processor, 2GB RAM, Gigabit NIC). As far as the network is concerned, the eight servers were interconnected by means of a Dell PowerConnect 2324 switch providing 100MBit connectivity.

[4]For the exact description of the machines as well as the installed software, see appendix D.

The Testing Environment

The testing environment is graphically illustrated in figure 7.4. The service requester and the service provider were deployed each on one machine. The machine hosting the service requester was at the same time monitoring the CPU usage on all eight servers. The ***SOS!e*** framework was running on one or more machines, depending on the use-case; details will be provided in the next section.

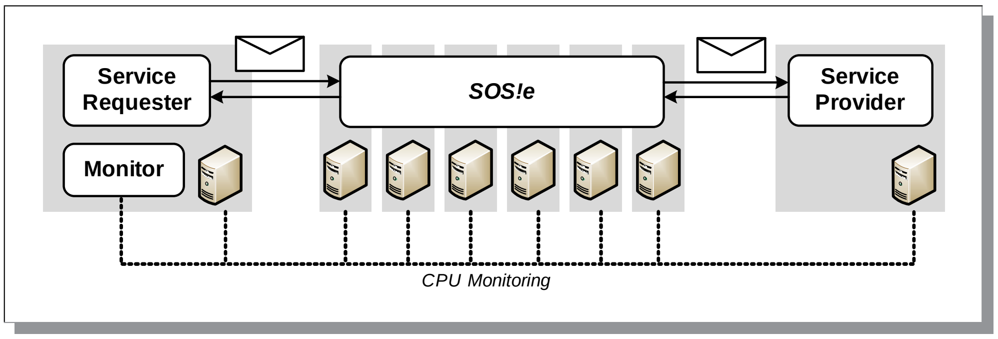

Figure 7.4: The realization of the testing environment

7.2.5 Experimental Results

Goals

The first goal of the performance tests was to determine the variation in performance when messages are routed through a different number of security services. For this purpose several use-cases were considered. Each use-case represents a possible deployment of the ***SOS!e*** framework. The use-cases together with the results of the performance tests are presented in the next subsection.
A further goal of the performance tests was to determine the variation in performance depending on the size of the messages. Because the parsing of XML messages is expected to introduce a significant overhead, it was important to determine how much the performance degrades with the increase of the message size.

Performance Variations with Different SOS!e Deployments

As it happens with most distributed systems, also in the case of ***SOS!e*** there are multiple deployment possibilities. Four different use-cases were taken into consideration for the performance testing - see below. The use-cases illustrate typical usage scenarios. The number of security services ranged from a single one to five (in the following figures these are noted S1 ... S5).

Use-Case #1 In this use-case (figure 7.5), the Mule ESB and the security services were all hosted on a single physical machine. This is a simple scenario, however not a very practical one, as in practice it is expected that the security services are deployed on different machines.
The results of the performance tests for this use-case are showed in figure 7.6. What can be seen here is that when the ***SOS!e*** framework is introduced, there is a significant impact on throughput: from 392 requests/s without ***SOS!e*** , the throughput decreases to 136 requests/s when a single security service is introduced. The throughput then decreases linearly when additional security services are introduced. A justification for this initial performance shift is the overhead introduced by the Mule ESB.

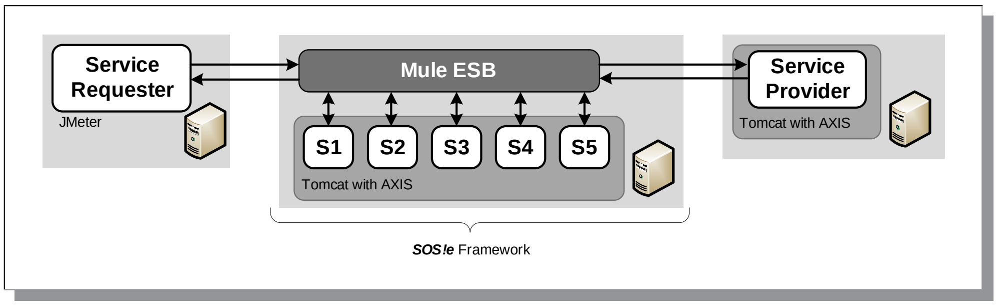

Figure 7.5: Use-case #1: Deployment diagram

The initial shift is less visible for the message delay: this parameter increases from 6 ms without ***SOS!e*** to 21 ms when a single security service is introduced and then to 35, 54, 82, 109 ms when additional security services are introduced. The variation of message delay is linear with the number of security services.

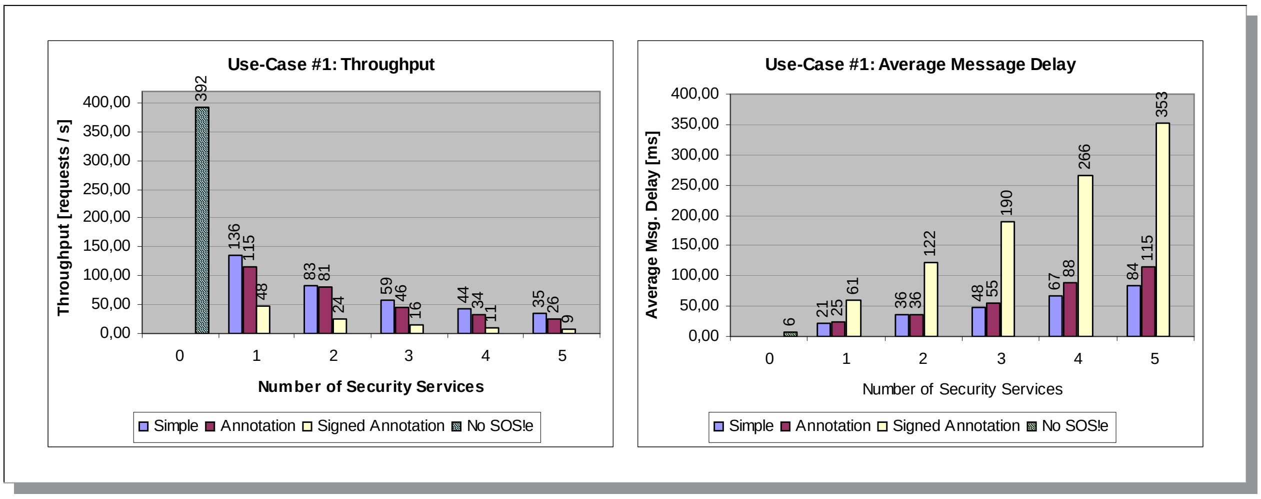

Figure 7.6: Performance results for use-case #1

As far as the impact of the type of security service used on performance, what we can see is that simple services perform slightly better than the annotations services (both in terms of throughput and message delay). Furthermore, the security services processing digitally signed annotations perform significantly worse than the other two. This fact is justified by the fact that cryptographic operations are CPU-intensive.

Use-Case #2 In this use-case (figure 7.7), there is a central Mule ESB which coordinates the activity of all security services. One of the security services is deployed on the same machine with the Mule ESB, while the other security services are deployed on remote machines.
The results of the performance tests for this use-case are showed in figure 7.8. They are very similar to the ones from the previous use-case. The difference here is that both throughput and message delay have slightly better values. This is justified by the fact that the security services are deployed on different physical machines and thus more processing power is available for the framework. An illustrative example is the message delay for a scenario with 5 security services using digitally signed annotations: 353 ms in the previous use-case, compared to just 200 ms in this use-case.

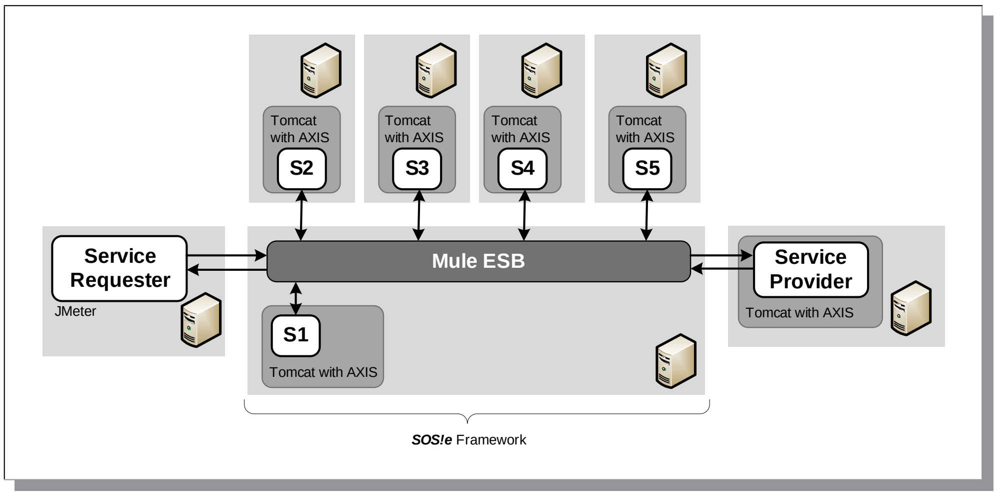

Figure 7.7: Use-case #2: Deployment diagram

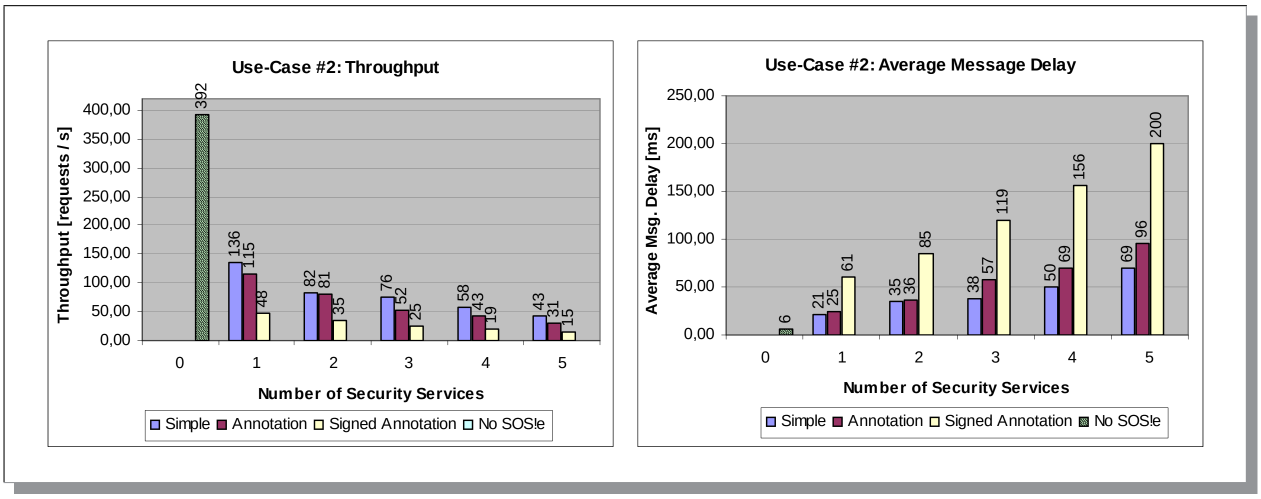

Figure 7.8: Performance results for use-case #2

Use-Case #3 In this use-case (figure 7.9), there are several Mule ESB instances, each deployed on a different physical machine. The Mule ESB instances communicate with one another and each of them invokes one security service which is hosted on the same machine with the ESB.

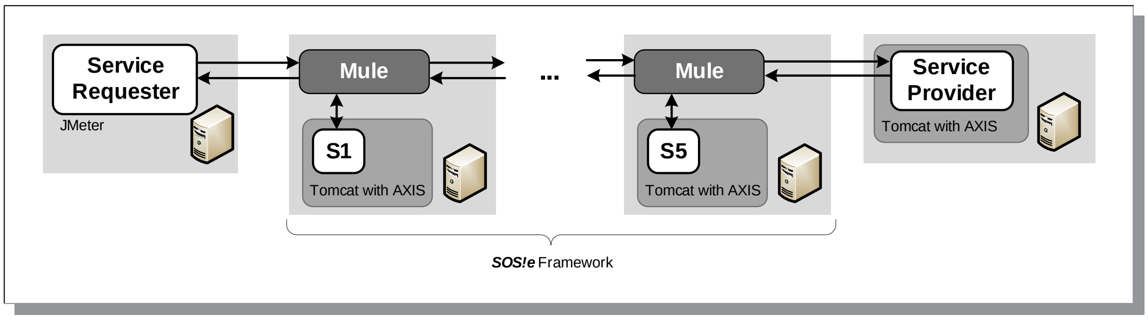

Figure 7.9: Use-case #3: Deployment diagram

The results of the performance tests for this use-case are similar to the first use-cases (figure 7.10). Compared to use-case #2, the results are slightly worse, the justification for this being the overhead in XML parsing introduced by the Mule ESBs: each message is parsed twice for each intermediary service (once by the Mule ESB and once by AXIS).

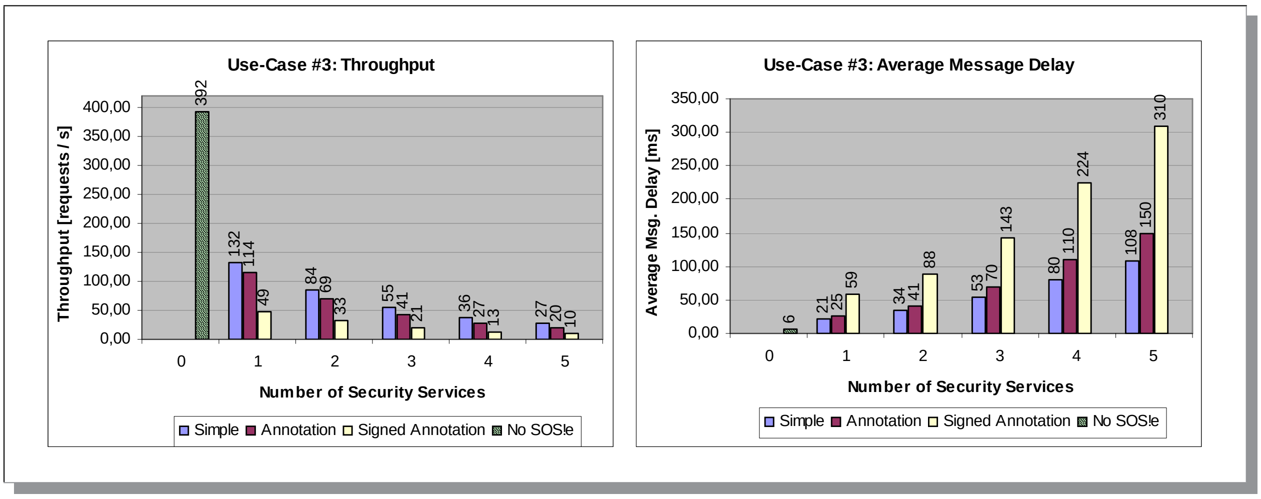

Figure 7.10: Performance results for use-case #3

Use-Case #4 The deployment configuration in this use-case (figure 7.11) is similar with the previous one, the difference being that the security services are not realized as Web services (and hosted in AXIS), but are instead realized as Mule components (see 5.6.3). This use-case is expected to render better performance as it saves the overhead of converting messages into SOAP for each security service invocation.

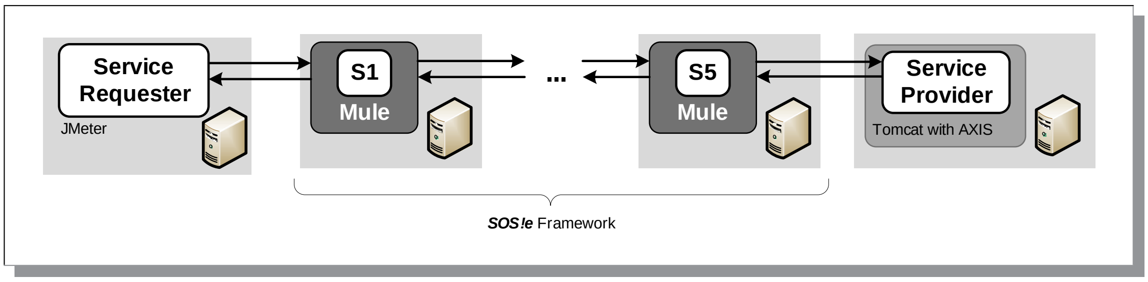

Figure 7.11: Use-case #4: Deployment diagram

The results of the performance tests for this use-case are showed in figure 7.12. The same linear variations of the throughput and message delay can also be observed here. However, compared to the previous use-case the performance is significantly better - for the simple security services, throughput is almost double while the average message delay is almost half. The justification for this is the fact that messages are only parsed once (by the Mule ESB).

Performance Variations with the Increase of the Message Size

In order to determine how the performance varies with the increase of the message size, a typical Authentication-Authorization-Audit scenario was investigated. As in the previous use-cases, because it was desired that only the overhead introduced by the security framework be considered, the security services were simulated by dummy services. Two simple services performing annotations were used (simulating the authentication and the authorization services) together with one simple

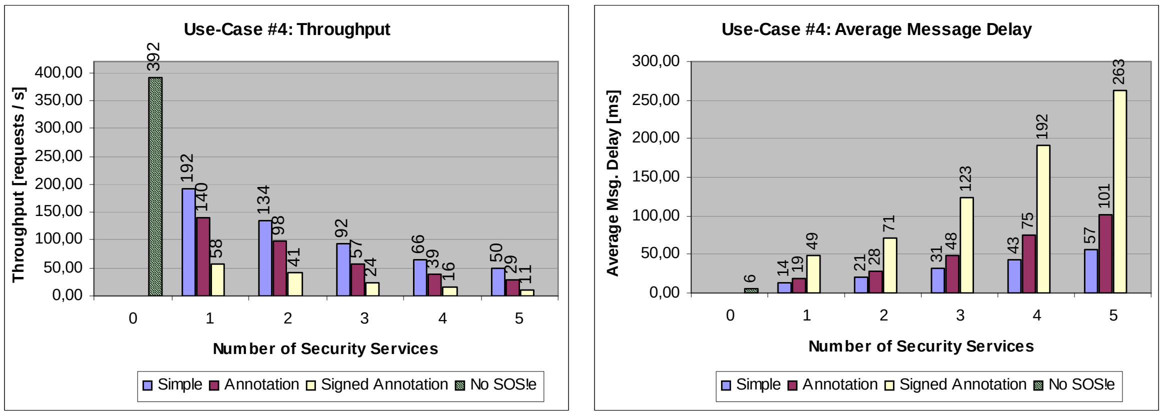

Figure 7.12: Performance results for use-case #4

service simulating the audit service. The deployment was the one described in the previous section in use-case #4.

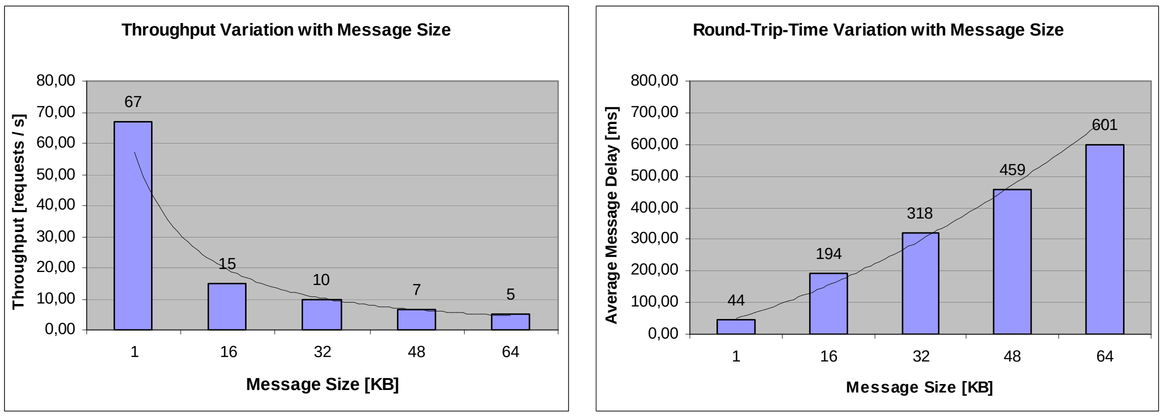

Figure 7.13: Throughput and delay variation with message size

The results are showed in figure 7.13. As expected, as the message size increases, the performance degrades:

Throughput The throughput is highly affected by the increase in message size: from 67 requests/s for 1KB messages this one drops to only 5 requests/s when the message size is increased to 64KB. The reason for this is the overhead caused by XML parsing which is both CPU consuming and memory consuming.

Message Delay As illustrated by the trend line, the average message delay increases almost linearly with the size of the message. For larger messages, high delays are to be expected: in our case more than half of a second for a 64KB message.

7.2.6 Discussion

As seen in this section, the ***SOS!e*** framework has a significant influence on both throughput and message delay. In this conditions, in order to determine whether or not it is appropriate to deploy a ***SOSA*** based solution for a concrete application, the following issues should be taken into consideration:

Performance of the Application If the performance parameters (average message delay and throughput) of the service provider are worse compared to the values showed in the previous section, than the impact of deploying a ***SOSA*** based approach will not be significant. However, if the service provider is performing very good (low latency and high throughput), the introduction of a ***SOSA*** based approach will lead to the security system being a bottleneck for the system.

Service Requester's Expectations If the service requesters have high expectation with regard to throughput and latency, then probably ***SOSA*** is not the right answer. However, there are several types of applications where the performance expectations are not so strict. Examples are given below:

- *B2B transactions* where there is no human person waiting for the response to come back from the service provider.
- *Asynchronous Web service invocations* where the requester does not block waiting for the response to come.
- *Request-only Web service invocations* were there is no response at all.

Performance Evaluation for OGC Web Services

Because most of the practical work related to ***SOSA*** was applied to OGC Web services (see also chapter 6), the performance implications of protecting WMS and WFS services with a ***SOSA*** based approach will be discussed briefly.
Figures 7.14 and 7.15 show the results of performance tests carried out on the deegree[5] open-source framework. The tests are described in [83] and were executed on different databases ranging from 3.000 objects (DB1) to 768.000 objects (DB9).

Throughput As indicated by the graphs, the throughput in the case of WMS ranges from 10.000 maps/hour to about 1.000 maps/hour. This translates to a range between 2.7 maps/s to 0.27 maps/s. The throughput for the WFS ranges from 0.2 feature collections/s to 0.11 feature collections/s. This values are a very low in comparison with the results obtained for the ***SOS!e*** framework which ranged from 136 requests/s to about 5 requests/s depending on deployment and message size.
Because the throughput of WMS and WFS services is so low, it is expected that the deployment of a ***SOSA*** based security solution will have little influence on the overall application throughput.

Average Message Delay The average message delay ranged from 600ms to more than 9s for the WMS and from 400ms to more than 4s in the case of the WFS. The tests performed on the ***SOS!e*** framework showed that this one introduces delays ranging from 21ms to about 600ms depending

[5]deegree is an open-source project implementing several OGC standards. It was the reference implementation for the WMS specification. http://www.deegree.org.

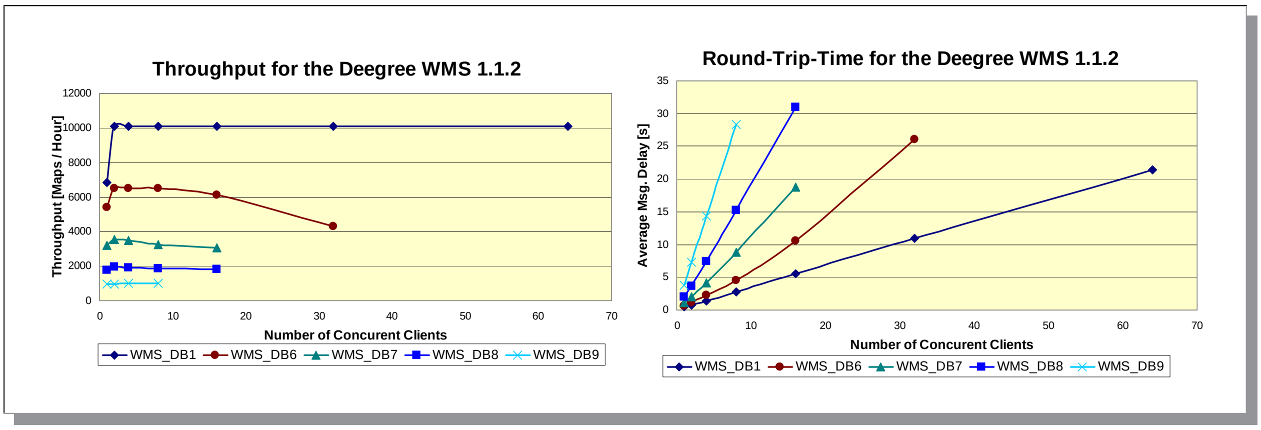

Figure 7.14: Performance measurements for WMS according to [83]

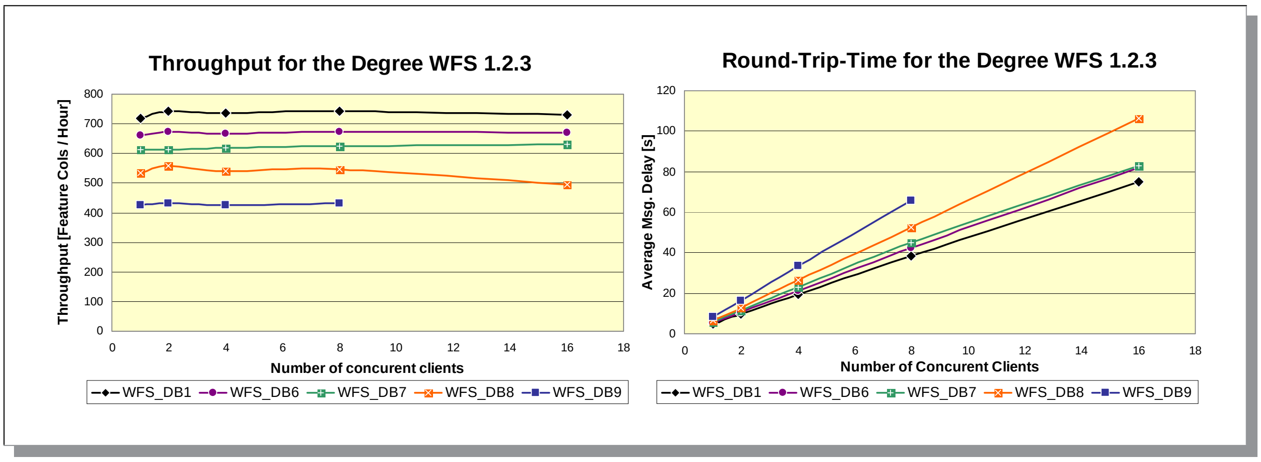

Figure 7.15: Performance measurements for WFS according to [83]

on the deployment configuration and message size.
As described in section 3.1, the delay introduced by the security system ads to the message delay of the protected service. Introducing a ***SOSA*** security system would have some influence on the overall application performance. For the faster WMS and WFS operations the influence is greater, while for the slower ones the overhead introduced by the security framework will be barely observable.

Chapter 8

Conclusions and Outlook

This chapter concludes with the results of this work, enumerates the contributions this dissertation brings to computer science and the world of geospatial information, and provides an outlook for future research.

8.1 Conclusions

Web services have been a major technology trend in the IT industry for almost a decade now, and this trend is continuing as SOA and Web services replace other methods and technologies used in the design, development, deployment and distribution. The motivation behind their wide-spread adoption is related to the advantages this technology brings: cost-reduction, increase reuse, simplified integration, more agile infrastructures.
Quite naturally everybody willing to relay on this technology expects a solid foundation for security that would allow for secure back-office transactions and support various business models over the Internet. Security has been repeatedly described as a key requirement in the success of any SOA project [97, 71, 107].
In this dissertation the topic of security in the context of Web services was addressed and an architecture for security systems was proposed. The main idea behind this architecture is that security functions such as authentication, authorization, audit, etc. are implemented externally to the protected application, and not embedded in the application or implemented at the message middleware layer. Furthermore, the different security functions are realized as independent services and designed according to the principle of separation of concerns. This leads to a collection of *security services* forming a security infrastructure: these services can be used to protect several applications within the network. Enterprise Application Integration (EAI) techniques are used for combining the security services together and binding them to the applications being protected through the definition of orchestrations or choreographies. Because of its service-oriented approach to security, the architecture is named the **S**ervice **O**riented **S**ecurity **A**rchitecture or ***SOSA*** .
In this dissertation this architecture has been described by means of the ISO RM-ODP, prototype implementations have been showed, and experiences using these implementations in different projects have been presented. These experiences show the feasibility of the proposed architecture. Furthermore, a detailed evaluation of ***SOSA*** was presented, both from an architectural point of view (using metrics suggested by [73]), as well as from a performance point of view. The evalu-

ation showed that while a ***SOSA*** based security system (depending on the implementation and deployment) can have significantly degrade the overall application performance, it brings a lot of advantages with regard to system design, implementation and maintenance. The following constitute strong points of the proposed architecture: scalability, understandability, verifiability, extensibility, reusability, portability.
With regard to the performance concerns, it was showed that while ***SOSA*** is not adequate for applications where high performance (low latency, high throughput) is expected, there are multiple classes of applications for which ***SOSA*** is adequate. One such example are OGC Web services.

8.2 Contributions to Information and Computer Science

This dissertation brings the following contributions to the field of Information and Computer Science:

- An architectural evaluation based on a consistent set of architectural properties of different approaches to implementing security for Web services;
- The Service Oriented Security Architecture or ***SOSA*** - an architecture for the design of Web services security systems, including a taxonomy of security services;
- A set of patterns addressing common security issues related to Web services;
- An evaluation of the proposed architecture using the aforementioned consistent set of architectural properties together with a performance evaluation based on the ***SOS!e*** prototype implementation.

8.3 Contribution to the World of Geospatial Information

Through the practical experiences described in chapter 6, this dissertation contributes to the world of geospatial information by showing how to deal with different security-related issues associated with the distribution of geospatial information. The topics discussed here are: click-through licensing, GeoDRM, non-repudiation and confidentiality, and integration with legacy infrastructures. The solutions described show how these issues are handled using open standards and open-source software.

8.4 Outlook for Future Research

The following topics have not been further considered in this dissertation and are possible directions for future research:

Dynamic Binding Extend the concept of security services to business registries (such as UDDI). This will certainly not fit all scenarios, but one can imagine for example an enterprise-wide UDDI registry containing information about all security services deployed in the enterprise. A proper categorization and good service meta-data would allow a client to discover the necessary services and integrate them without having prior knowledge of their existence and capabilities. Another example here are security services that have a clear task such as encryption or digital signing. One can envisage such services listed in a public UDDI registry (such as `http://uddi.org`). Of course, in order to be trusted by the clients (nobody wants to give their private key to an untrusted entity) they should have some sort of certification (mechanisms similar to the ones for code signing could also be applied to services).

WS-CDL Complex security systems, containing many services and having complex security policies, can be described by means of WS-CDL. This would provide a good way for modeling the system because various graphical tools are available. Furthermore, automated system-testing can be performed on the WS-CDL representation of the system. In addition to this, specialized tools could be developed that generate runnable representations of the service aggregations (either as message routing rules or as BPEL scripts).

SOS!e There are several shortcomings to the existing implementations of the ***SOS!e*** framework (see both section 5.6.2 and section 5.6.3 in the Technology Viewpoint). After building ***SOS!e 1.0*** and experiencing with it, ***SOS!e 2.0*** was developed. This new version incorporated the feedback received together with some new ideas. However, this new framework also has some shortcomings. These shortcomings should be addressed by future versions of the framework. Below, a list of issues to be addressed by future versions of the framework is provided:

- Support for Web services deployed on AXIS2. AXIS2 is a complete re-write and redesign of the Apache AXIS SOAP stack. It provides better performance (because of the StAX[1] parser) and lower memory footprint.
- Development of more security services performing common security tasks, such as: image watermarking, DRM packaging using different existing DRM stacks (ISO REL, ODRL, etc.), integration with popular services storing user attributes (such as eBay or PayPal).

[1]Streaming API for XML

Bibliography

[1] The merriam-webster english dictionary. online. http://www.m-w.com/.

[2] Mule 1.4.0 esb user guide. online. http://dist.codehaus.org/mule/docs/mule-1.4.0-website-docs.pdf.

[3] Webopedia. online. http://www.webopedia.com/.

[4] *ISO 10181-3:1996 Information technology – Open Systems Interconnection – Security frameworks for open systems: Access control framework*. International Standards Organization (ISO), September 1996.

[5] *ISO/IEC 10746-2: Information technology - Open Distributed Processing - Reference Model: Foundations*. International Standards Organization (ISO), 1996. http://isotc.iso.org/livelink/livelink/fetch/2000/2489/Ittf_Home/PubliclyAvailableStandards.htm.

[6] *ISO/IEC 10746-3: Information technology - Open Distributed Processing - Reference Model: Architecture*. International Standards Organization (ISO), 1996. http://isotc.iso.org/livelink/livelink/fetch/2000/2489/Ittf_Home/PubliclyAvailableStandards.htm.

[7] *ISO/IEC 10746-1: Information technology - Open Distributed Processing - Reference Model: Overview*. International Standards Organization (ISO), 1998. http://isotc.iso.org/livelink/livelink/fetch/2000/2489/Ittf_Home/PubliclyAvailableStandards.htm.

[8] *ISO/IEC 10746-4: Information technology - Open Distributed Processing - Reference Model: Architectural Semantics*. International Standards Organization (ISO), 1998. http://isotc.iso.org/livelink/livelink/fetch/2000/2489/Ittf_Home/PubliclyAvailableStandards.htm.

[9] *Information processing systems – Open Systems Interconnection – Basic Reference Model – Part 2: Security Architecture*. International Standards Organization (ISO), June 2000.

[10] *UDDI Version 2*. OASIS, July 2002. Status: OASIS Standard, http://www.oasis-open.org/committees/uddi-spec/doc/tcspecs.htm.

[11] *Web Service Choreography Interface (WSCI) 1.0*. World Wide Web Consortium (W3C), August 2002. Status: W3C Note, http://www.w3.org/TR/2002/NOTE-wsci-20020808/.

[12] *Assertions and Protocol for the OASIS Security Assertion Markup Language (SAML) V1.1*. OASIS, September 2003. Status: OASIS Standard, http://www.oasis-open.org/committees/download.php/3406/oasis-sstc-saml-core-1.1.pdf.

[13] *eXtensible Access Control Markup Language v1.0*. OASIS, February 2003. Status: OASIS Standard, http://www.oasis-open.org/committees/download.php/2406/oasis-xacml-1.0.pdf.

[14] *OGC Reference Model*. Open Geospatial Consortium (OGC), 2003. Version 0.1.3, http://www.opengeospatial.org/specs/?page=baseline.

[15] *UDDI Version 3.0.2*. OASIS, October 2004. Status: Committee Draft, http://uddi.org/pubs/uddi-v3.0.2-20041019.htm.

[16] *WS-Reliability 1.1*. OASIS, November 2004. Status: OASIS Standard, http://docs.oasis-open.org/wsrm/ws-reliability/v1.1/wsrm-ws_reliability-1.1-spec-os.pdf.

[17] *XML Schema Part 0: Primer Second Edition*. World Wide Web Consortium (W3C), 2004. Status: W3C Recommendation, http://www.w3.org/TR/2004/REC-xmlschema-0-20041028/.

[18] *eXtensible Access Control Markup Language v2.0*. OASIS, February 2005. Status: OASIS Standard, http://docs.oasis-open.org/xacml/2.0/access_control-xacml-2.0-core-spec-os.pdf.

[19] *Information technology - Vocabulary - Part 1: Fundamental terms*. International Standards Organization (ISO), 2005.

[20] *ISO/IEC 2382-01: Information Technology - Open Distributed Processing - Use of UML for ODP system specifications*. International Standards Organization (ISO), committee draft v02.00 edition, May 2005.

[21] *The Open Grid Services Architecture, Version 1.0*. Global Grid Forum, 2005. https://forge.gridforum.org/sf/go/doc13515?nav=1.

[22] *Security Assertions Markup Language (SAML) V2.0 - Core.* OASIS, March 2005. Status: OASIS Standard, http://docs.oasis-open.org/security/saml/v2.0/saml-core-2.0-os.pdf.

[23] *SOAP Message Transmission Optimization Mechanism.* World Wide Web Consortium (W3C), January 2005. Status: W3C Recommendation, http://www.w3.org/TR/2005/REC-soap12-mtom-20050125/.

[24] *Web Services Atomic Transaction (WS-AtimicTransaction).* IBM, BEA Systems, Microsoft, Arjuna, Hitachi, IONA, August 2005. http://www-128.ibm.com/developerworks/library/specification/ws-tx.

[25] *Web Services Business Activity (WS-BusinessActivity).* IBM, BEA Systems, Microsoft, Arjuna, Hitachi, IONA, August 2005. http://www-128.ibm.com/developerworks/library/specification/ws-tx.

[26] *Web Services Choreography Description Language Version 1.0.* World Wide Web Consortium (W3C), November 2005. Status: W3C Candidate Recommendation, http://www.w3.org/TR/2005/CR-ws-cdl-10-20051109/.

[27] *Web Services Coordination (WS-Coordination).* IBM, BEA Systems, Microsoft, Arjuna, Hitachi, IONA, August 2005. http://www-128.ibm.com/developerworks/library/specification/ws-tx.

[28] *Web Services Secure Conversation Language.* IBM, BEA Systems, Microsoft, Computer Associates, Actional, VeriSign, Layer 7 Technologies, Oblix, OpenNetwork Technologies, Ping Identity, Reactivity, RSA Security, February 2005. http://www-128.ibm.com/developerworks/library/specification/ws-secon/.

[29] *Web Services Security Policy Language (WS-SecurityPolicy), Version 1.1.* IBM, Microsoft, RSA Security, VeriSign, July 2005. ftp://www6.software.ibm.com/software/developer/library/ws-secpol.pdf.

[30] *XML-binary Optimized Packaging.* World Wide Web Consortium (W3C), January 2005. Status: W3C Recommendation, http://www.w3.org/TR/2005/REC-xop10-20050125/.

[31] *Basic Profile Version 1.1.* Web Services Interoperability Organization (WS-I), 2006. Status: Final Material, http://www.ws-i.org/Profiles/BasicProfile-1.1-2006-04-10.html.

[32] *Extensible Markup Language (XML) 1.1 (Second Edition).* World Wide Web Consortium (W3C), 2006. Status: W3C Recommendation, http://www.w3.org/TR/2006/REC-xml11-20060816.

[33] *Web Services Federation Language (WS-Federation).* BEA Systems, BMC Software, CA, Inc., IBM, Layer 7 Technologies, Microsoft, Novell, VeriSign, version 1.1 edition, December 2006. http://www-128.ibm.com/developerworks/library/specification/ws-fed/.

[34] *Web Services Policy 1.2 - Attachment (WS-PolicyAttachment).* World Wide Web Consortium (W3C), 2006. Status: Member Submission, http://www.w3.org/Submission/2006/SUBM-WS-PolicyAttachment-20060425/.

[35] *Web Services Security: SOAP Message Security 1.1.* OASIS, February 2006. Status: OASIS Standard, http://www.oasis-open.org/committees/download.php/16790/wss-v1.1-spec-os-SOAPMessageSecurity.pdf.

[36] *Web Services Security SOAP Messages with Attachments (SwA) Profile 1.1.* OASIS, February 2006. Status: OASIS Standard, http://www.oasis-open.org/committees/download.php/16672/wss-v1.1-spec-os-SwAProfile.pdf.

[37] *Architecture of Spatial Data Infrastructure Germany, Version 1.0.* GDI-DE, May 2007.

[38] *Digital Signature Service Core Protocols, Elements, and Bindings Version 1.0.* OASIS, 1.0 edition, February 2007. Status: OASIS Standard, http://docs.oasis-open.org/dss/v1.0/oasis-dss-core-spec-cs-v1.0-r1.htm.

[39] *Web Services Description Language (WSDL) Version 2.0 Part 1: Core Language.* World Wide Web Consortium (W3C), June 2007. Status: W3C Recommendation, http://www.w3.org/TR/2007/REC-wsdl20-20070626/.

[40] *WS-Trust 1.3.* OASIS, March 2007. OASIS Standard, http://docs.oasis-open.org/ws-sx/ws-trust/200512/ws-trust-1.3-os.html.

[41] Mohamad Afshar, Nickolaos Kavantzas, Ramana Turlapati, Roger Goudarzi, Barmak Meftah, and Prakash Yamuna. Best practices for securing your soa: A holistic approach. *Java Developer's Journal*, June 2006. http://java.sys-con.com/read/232071.htm.

[42] Apache axis user's guide. online. http://ws.apache.org/axis/java/user-guide.html.

[43] Amit Asaravala. Giving soap a rest. online, October 2002. http://www.devx.com/DevX/Article/8155.

[44] Norman B. Leventhal Map Center at the Boston Public Library. Planisphaerium ptolemaicum siue machina orbium mundi ex hypothesi ptolemaica in plano disposita. online. http://maps.bpl.org/id/M8727.

[45] Michael Blow (BEA), Yaron Goland (BEA), Matthias Kloppmann (IBM), Frank Leymann (IBM), Gerhard Pfau (IBM), Dieter Roller (IBM), and Michael Rowley (BEA). Bpelj: Bpel for java. *white paper*, March 2004. ftp://www6.software.ibm.com/software/developer/library/ws-bpelj.pdf.

[46] Norton Bedford. *A Statement of Basic Accounting Theory.* Chicago American Accounting Association, 3rd edition, 1970.

[47] M. Bishop. A model of security monitoring. In *Fifth Annual Computer Security Applications Conference*, December 1989.

[48] Andr B. Bondi. Characteristics of scalability and their impact on performance. In *2nd international workshop on Software and performance*, pages 195 – 203.

[49] David Booth, Hugo Haas, Fracis McCabe, Eric Newcomer, Michael Champion, Chris Ferris, and David Orchard, editors. *Web Services Architecture*. World Wide Web Consortium (W3C), 2004. Status: W3C Working Group Note, http://www.w3.org/TR/2004/NOTE-ws-arch-20040211/.

[50] Boutell.Com. Www faqs: What is the maximum length of a url? online, October 2006. http://www.boutell.com/newfaq/misc/urllength.html.

[51] *Web Services Business Process Execution Language Version 2.0*. OASIS, April 2007. Status: OASIS Standard, http://docs.oasis-open.org/wsbpel/2.0/OS/wsbpel-v2.0-OS.html.

[52] G. Brose. Securing Web Services with SOAP Security Proxies. *Proc. Int'l Conf. Web Services (ICWS'03)*, pages 231–234, 2003.

[53] Russell Butek. Which style of wsdl should i use? online, October 2004. http://www-128.ibm.com/developerworks/webservices/library/ws-whichwsdl/.

[54] Rick Caccia. An introduction to oracle web services manager. May 2005. http://www.oracle.com/technology/products/webservices_manager/pdf/oracle_wsm_402_wp.pdf.

[55] Michele Cantara. Market focus: Trends and forecast for it professional services for web services and soa, 2005. June 2005. Gartner Dataquest Paper.

[56] David A. Chappell. *Enterprise Service Bus*. O'Reilly, 2004.

[57] Cape Clear. Cape clear esb security. *white paper*, November 2005. http://developer.capeclear.com/SecurityPaper.

[58] Frank Cohen. Performance testing soap-based applications. *IBM developerWorks*, November 2001. http://www.ibm.com/developerworks/webservices/library/ws-testsoap/.

[59] Open Geosaptial Consortium. Ogc glossary of terms. online. http://www.opengeospatial.org/resource/glossary.

[60] IBM Corporation and Microsoft Corporation. Security in a web services world: A proposed architecture and roadmap. *Whitepaper*, April 2002. ftp://www6.software.ibm.com/software/developer/library/ws-secmap.pdf.

[61] Microsoft Corporation. Windows 2000 kerberos authentication. *Microsoft TechNet*. http://www.microsoft.com/technet/prodtechnol/windows2000serv/deploy/confeat/kerberos.mspx.

[62] Tony Cowan. Web services security with websphere application server v6 – part 1. *IBM WebSphere Developer Technical Journal*, March 2006. http://www-128.ibm.com/developerworks/websphere/techjournal/0603_cowan/0603_cowan.html.

[63] Jeff de la Beaujardiere, editor. *OpenGIS Web Map Service (WMS) Implementation Specification, Version 1.3.0*. Open Geospatial Consortium (OGC), 2006. http://www.opengeospatial.org/standards/wms.

[64] Andreas Donaubauer. *Interoperable Nutzung verteilter Geodatenbanken mittels standardisierter Geo Web Services*. PhD thesis, Technical University of Munich, 2004. http://mediatum2.ub.tum.de/node?id=601056.

[65] Jan Drewnak, Christian Elfers (Editor), Roland M. Wagner (Editor), and Cristian Opincaru. Geodrm engineering viewpoint and supporting architecture. Interoperability program report, Open Geospatial Consortium (OGC), April 2007. OGC #06-184r2.

[66] Philippe Duschene and Jerome Sonnet. *WMS Change Request: Support for WSDL and SOAP*. Open Geospatial Consortium (OGC), 2005. Status: Discussion Paper, OGC #04-050r1.

[67] ebay web services rest api. online. http://developer.ebay.com/developercenter/rest/.

[68] Cristian Opincaru (Editor). *Trusted GeoServices IPR*. Open Geospatial Consortium (OGC), April 2007. OGC #06-107.

[69] Wolfgang Eibach and Dietmar Kuebler. Metering and accounting for web services. *IBM developerWorks*, July 2001. http://www-128.ibm.com/developerworks/webservices/library/ws-maws/.

[70] Esri arcweb services website. online. http://www.esri.com/software/arcwebservices/index.html.

[71] Mark O'neill et al. *Web Services Security*. McGraw Hill, 2003.

[72] Von Welch et al. Globus toolkit version 4 grid security infrastructure: A standards perspective. September 2005. http://www.globus.org/toolkit/docs/4.0/security/GT4-GSI-Overview.pdf.

[73] Roy Thomas Fielding. *Architectural Styles and the Design of Network-based Software Architectures*. PhD thesis, University of California, Irvine, 2000. http://roy.gbiv.com/pubs/dissertation/fielding_dissertation_2up.pdf.

[74] Tim Fisher. Sichere web services unter dem microsoft .net framework. Student Assignment IS 34/2004, University of the German Armed Forces, Munich, December 2004.

[75] Tim Fisher. Ein sicherheits-framework fuer opengis web services unter nutzung des bpel-standard. Diploma Thesis ID 08/2005, University of the German Armed Forces, Munich, September 2005.

[76] Dr. Martin Fornefeld and Peter Oefinger. Billing models for geo web services a contribution to create a spatial data infrastructure in switzerland. Technical report, MICUS Management Consulting GmbH, January 2005.

[77] Apache Software Foundation. Apache axis2 user's guide. online. http://ws.apache.org/axis2/1_1_1/userguide.html.

[78] Erich Gamma, Richard Helm, Ralph Johnson, and John Vlissides. *Design Patterns: Elements of Reusable Object-Oriented Software*. Addison-Wesley, 1995.

[79] Rdiger Gartman. Ows-4 ipr: Ows common change request: Add ws-security to ows soap binding. Technical report, Open Geospatial Consortium (OGC), April 2007. OGC #06-177.

[80] Anne Geraci, Freny Katki, Louise McMonegal, Bennett Meyer, John Lane, Paul Wilson, Jane Radatz, Mary Yee, Hugh Porteous, and Fredrick Springsteel, editors. *IEEE Standard Computer Dictionary: Compilation of IEEE Standard Computer Glossaries*. Institute of Electrical and Electronics Engineers Inc., The, 1991.

[81] Sebastian Gerken. Kerberos authentifikation fuer web services mit .net. Student Assignment IS 68/2006, University of the German Armed Forces, Munich, December 2006.

[82] Gabriela Gheorghe. Implementation and test of a service oriented security (sos) system. Diploma thesis, Polytechnic University of Bucharest, Romania, October 2007.

[83] Lydia Gietler. Leistungsbewertung von ogc web services. 7. Seminar GIS and Internet, Neubiberg, Germany, September 2004.

[84] *OpenGIS Geography Markup Language (GML) Implementation Specification, Version 2.1.2*. Open Geospatial Consortium (OGC), September 2002. http://portal.opengeospatial.org/files/?artifact_id=11339.

[85] *OpenGIS Geography Markup Language (GML) Implementation Specification, Verson 3.1.0*. Open Geospatial Consortium (OGC), February 2004. http://portal.opengeospatial.org/files/?artifact_id=4700.

[86] Google earth com api documentation. online. http://earth.google.com/comapi/index.html.

[87] Google maps api. online. http://www.google.com/apis/maps/.

[88] Martin Gudgin, Marc Hadley, Noah Mendelsohn, Jean-Jacques Moreau, and Henrik Frystyk Nielsen, editors. *SOAP Version 1.2 Part 1: Messaging Framework*. World Wide Web Consortium (W3C), June 2003. Status: W3C Recommendation, http://www.w3.org/TR/2003/REC-soap12-part1-20030624/.

[89] Martin Gudgin, Marc Hadley, Noah Mendelsohn, Jean-Jacques Moreau, and Henrik Frystyk Nielsen, editors. *SOAP Version 1.2 Part 2: Adjuncts*. World Wide Web Consortium (W3C), June 2003. Status: W3C Recommendation, http://www.w3.org/TR/2003/REC-soap12-part2-20030624/.

[90] Andreas Haenel. Nutzung des biolancc system als authentifizierungsmodul fuer ogc web services. Diploma Thesis ID 06/2005, University of the German Armed Forces, Munich, September 2005.

[91] Charles Heazel and Rick Morrison. Geo-digital rights management working group charter. online, visited 06.2007, May 2004. http://www.opengeospatial.org/projects/groups/geormwg.

[92] Heather Hinton, Maryann Hondo, and Dr. Beth Hutchison. Security patterns within a service-oriented architecture. November 2005. http://searchwebservices.techtarget.com/searchWebServices/downloads/SecuritySOA_(2).pdf.

[93] Frederick Hirsh, John Kemp, and Jani Ilkka. *Mobile Web Services - Architecture and Implementation*. John Wiley and Sons, 2006.

[94] Jason Hogg, Don Smith, Fred Chong, Lonnie Wall Dwayne Taylor, and Paul Slater. *Web Service Security: Scenarios, Patterns, and Implementation Guidance for Web Services Enhancements (WSE) 3.0*. Microsoft Press, December 2005. http://msdn.microsoft.com/practices.

[95] Gregor Hohpe and Bobby Woolf. *Enterprise Integration Patterns: Designing, Building, and Deploying Messaging Solutions*. Addison-Wesley, 2003.

[96] IBM. Websphere datapower xml security gateway xs40 - features and benefits. *online*. http://www-306.ibm.com/software/integration/datapower/xs40/features/index.html.

[97] Gartner Inc. Gartner Survey of 1,300 CIOs Shows IT Budgets to Increase by 2.5 Percent in 2005. *Press Release*, 2005. http://www.gartner.com/press_releases/asset_117739_11.html.

[98] PayPal Inc. Paypal soap api developer reference. online, April 2007. Document Number: 100002.$enUS - 20070411$, https://www.paypal.com/en_US/pdf/PP_APIReference.pdf.

[99] Jmeter user manual. online. http://jakarta.apache.org/jmeter/usermanual/index.html, (visited August 2007).

[100] Martin Keen, Amit Acharya, Susan Biscop, Alan Hopkins, Sven Milinski, Chris Nott, Rick Robinson, Jonathan Adams, and Paul Verschueren. *Patterns: Implementing an SOA Using an Enterprise Services Bus*. IBM, July 2004. http://www.redbooks.ibm.com/.

[101] Gregor Kiczales, John Lamping, Anurag Menhdhekar, Chris Maeda, Cristina Lopes, Jean-Marc Loingtier, and John Irwin. Aspect-oriented programming. In Mehmet Akşit and Satoshi Matsuoka, editors, *Proceedings European Conference on Object-Oriented Programming*, volume 1241, pages 220–242. Springer-Verlag, Berlin, Heidelberg, and New York, 1997.

[102] M. Kloppmann, D. Koenig, F. Leymann, G. Pfau, A. Rickayzen, C. von Riegen, P. Schmidt, and I. Trickovic. WS-BPEL Extension for People–BPEL4People. *Joint white paper, IBM and SAP, July*, 2005.

[103] Rob H. Koenen, Jack Lacy, Michael Mackay, and Steve Mitchell. The long march to interoperable digital rights management. volume 92, pages 883 – 897, June 2004. http://ieeexplore.ieee.org/iel5/5/28864/01299164.pdf.

[104] Ron Lake. Introduction to gml geography markup language. *W3C online*. http://www.w3.org/Mobile/posdep/GMLIntroduction.html.

[105] Joshua Lieberman, Roderick Morrison, Cristian Opincaru, Ugo Taddei, and Roland M. Wagner. *OWS-3 GeoDRM Thread Activity and Interoperability Program Report: Access Control and Terms of Use (ToU) Click-through IPR Management*. Open Geospatial Consortium (OGC), February 2006.

[106] Skylab Mobilesystems Ltd. Ogc wms server list. October 2006. http://www.skylab-mobilesystems.com/en/wms_serverlist.html.

[107] Andreas Matheus. *Declaration and Enforcement of Access Restrictions for Distributed Geospatial Information Objects*. PhD thesis, Technical University of Munich, 2005. http://mediatum2.ub.tum.de/node?id=601767.

[108] Daniel A. Menasce and Virgilio A.F. Almeida. *Capacity Planning for Web Services: Metrics, Models, and Methods*. Prentice Hall, 2002.

[109] Microsoft virtual earth developer website. online. http://dev.live.com/virtualearth.

[110] Sun Microsystems. Java authentication and authorization service (jaas) overview. online. http://java.sun.com/products/jaas/overview.html.

[111] Nilo Mitra, editor. *SOAP Version 1.2 Part 0: Primer*. World Wide Web Consortium (W3C), June 2003. Status: W3C Recommendation, http://www.w3.org/TR/2003/REC-soap12-part0-20030624/.

[112] Doug Nebert, Carl Reed, and Roland M. Wagner. Proposal for a compatible sdi standards suite, sdi 1.0. In *GSDI-9*, November 2006. www.gsdi9.cl/english/papers/TS19.1paper.pdf.

[113] Douglas D. Nebert, editor. *The SDI Cookbook, Version 2.0*. online, January 2004. http://www.gsdi.org/docs2004/Cookbook/cookbookV2.0.pdf.

[114] *OpenGIS Abstract Specification, Topic 12 - The OpenGIS Service Architecture*. Open Geospatial Consortium (OGC), version 4.3 edition, 2002. http://portal.opengeospatial.org/files/?artifact_id=1221.

[115] *OpenGIS Web Service Common Implementation Specification, Version 1.0*. Open Geospatial Consortium (OGC), 2005. http://www.opengeospatial.org/standards/common.

[116] Jon Oltsik. Web services meet the network. *IBM whitepaper*, December 2005. http://publibfp.boulder.ibm.com/epubs/pdf/22476020.pdf.

[117] Cristian Opincaru. A security concept for the targeteam system. Diploma thesis, Polytechnic University of Bucharest, Romania, July 2003.

[118] Cristian Opincaru. Study about securing opengis web services. Technical report, Runder Tisch GIS e.V., January 2004.

[119] Cristian Opincaru and Gabriela Gheorghe. Service oriented security architecture. *EMISA 2007: Enterprise Modelling and Information Systems Architectures, Lecture Notes in Informatics*, October 2007.

[120] Cristian Opincaru and Andreas Matheus. Disclaimer-enablement fuer den web map service des ogc. Project documentation, University of the German Armed Forces, September 2006.

[121] Peyman Oreizy, Nenad Medvidovic, and Richard N. Taylor. Architecture-based runtime software evolution. In *ICSE '98: Proceedings of the 20th international conference on Software engineering*, pages 177–186, Washington, DC, USA, 1998. IEEE Computer Society.

[122] Request for quotation (rfq) and call for participation in the ows-3 web services initiative (ows-3), February 2005.

[123] Request for quotation (rfq) and call for participation in the ows-4 web services initiative (ows-4), April 2006.

[124] Michael P. Papazoglou and Dubray Jean-jacques. A survey of web services technologies. Technical Report DIT-04-058, University of Trento, July 2004. http://eprints.biblio.unitn.it/archive/00000586/01/mike.pdf.

[125] Chris Peltz. Web services orchestration and choreography. *Computer*, 36(10):46– 52, October 2003.

[126] Srinath Perera and Ajith Ranabahu. Axis2 - the future of web services. *JAX Magazine*. http://www.jaxmag.com/itr/online_artikel/psecom,id,747,nodeid,147.html.

[127] Mathias Pitt. Entwicklung einer sicheren 3-tier architektur fr ein biometrisches zugangskontrollsystem einschlielich der netzwerkschnittstelle zur anbindung von fingerscannern. Diploma thesis, University of the German Armed Forces, Munich, 2005.

[128] Ole Pophal. Ein sicherheitsframework fuer ogc webservices auf basis des soap. Diploma Thesis IS 07/2005, University of the German Armed Forces, Munich, September 2005.

[129] Kerry Raymond. Reference model of open distributed processing (rm-odp): Introduction. 1995. http://www.dstc.edu.au/Research/Projects/ODP/papers/icodp95.ps.

[130] *RFC2616: Hypertext Transfer Protocol – HTTP/1.1.* Internet Engineering Task Force (IETF), June 1999. Category: Standards Track, http://tools.ietf.org/html/rfc2616.

[131] *RFC2828: Internet Security Glossary.* Internet Engineering Task Force (IETF), May 2000. Category: Informational, http://tools.ietf.org/html/rfc2828.

[132] *RFC2903: Generic AAA Architecture.* Internet Engineering Task Force (IETF), August 2000. Category: Experimental, http://www.rfc-editor.org/.

[133] *RFC3198: Terminology for Policy-Based Management.* Internet Engineering Task Force (IETF), November 2001. Category: Informational, http://www.rfc-editor.org/.

[134] Rick Robinson. Understand enterprise service bus scenarios and solutions in service-oriented architecture, part 1, 2 and 3. *IBM developerWorks*, June 2004. http://www-128.ibm.com/developerworks/webservices/library/ws-esbscen/.

[135] Michael Roth. A java ee framework for managing biometric data based on biolancc. Diploma thesis, University of Zurich, Department of Informatics (IFI), December 2006. https://www.ifi.unizh.ch/fileadmin/site/teaching/Diplomarbeiten/Abgeschlossene_Diplomarbeiten/Jahrgang_2007/Roth_Michael.pdf.

[136] Deborah Russell and G.T. Gangemi Sr. *Computer Security Basics.* O'Reilly, 1992.

[137] Pierangela Samarati and Sabrina de Capitani di Vimercati. Access control: Policies, models, and mechanisms. *Lecture Notes in Computer Science*, 2171/2001:137, 2001.

[138] M.-T. Schmidt, B. Hutchison, P. Lambros, and R. Phippen. The enterprise services bus: Making service-oriented architecture real. *IBM Systems Journal*, 44(4):781–797, 2005. http://www.research.ibm.com/journal/sj/444/schmidt.html.

[139] A.P. Gonalves Serra, S.A. Vicente, D. Karam Jr, and M. Martucci Jr. Architecting frameworks for specific applications with rm-odp. In *1st International Workshop on ODP in the Enterprise Computing (WODPEC 2004)*, 2004.

[140] *Reference Model for Service Oriented Architecture.* OASIS, August 2006. Status: OASIS Standard, http://www.oasis-open.org/committees/download.php/19679/soa-rm-cs.pdf.

[141] Jerome Sonnet and Charles Savage. *OWS 1.2 SOAP Experiment Report.* Open Geospatial Consortium (OGC), 2003. Status: Discussion Paper, OGC #03-014.

[142] Burkhard Stiller, Thomas Bocek, Ming Peter, Frank Eyermann, Jrgen Sauerland, Jan Angrabeit, Mark Voelkl, and Jan-Christian Tylka. Management of biometric data in a distributed internet environment. *Integrated Management 2007 (IM2007)*, May 2007.

[143] Rajesh Sumra and Arulazi D. Quality of service for web servicesdemystification, limitations, and best practices. *developer.com*, March 2003. http://www.developer.com/services/article.php/2027911.

[144] Ofelia Tindea. Encryption and digital signatures for ogc web services. Internal report / project documentation, University of the German Armed Forces, Munich, July 2006.

[145] David Trowbridge, Dave Mancini, Dave Quick, Gregor Hohpe, James Newkirk, and David Lavigne. *Enterprise Solution Patterns Using Microsoft .NET.* Microsoft Press, June 2003. http://msdn.microsoft.com/practices.

[146] K. Ueno and M. Tatsubori. Early Capacity Testing of an Enterprise Service Bus. *Proceedings of the IEEE International Conference on Web Services (ICWS'06)-Volume 00*, pages 709–716, 2006.

[147] J. Vollbrecht, P. Calhoun, S. Farrell, L. Gommans, G. Gross, B. de Bruijn, C. de Laat, M. Holdrege, and D. Spence. *RFC2904: AAA Authorization Framework.* Internet Engineering Task Force (IETF), August 2000. Category: Informational, http://www.rfc-editor.org/.

[148] J. Vollbrecht, P. Calhoun, S. Farrell, L. Gommans, G. Gross, B. de Bruijn, C. de Laat, M. Holdrege, and D. Spence. *RFC2905: AAA Authorization Application Examples.* Internet Engineering Task Force (IETF), 2000. Category: Informational, http://www.rfc-editor.org/.

[149] Graham Vowles, John R. Herring, Cecil Goodwin, Mark V. Scardina, Andreas Matheus, Cristian Opincaru, and Wayne Debeugny. *Topic 18 - Geospatial Digital Rights Management Reference Model (GeoDRM RM).* Open Geospatial Consortium (OGC), 2006. Version 1.0.0, http://portal.opengeospatial.org/files/?artifact_id=17802.

[150] *SOAP 1.2 Attachment Feature.* World Wide Web Consortium (W3C), June 2004. Status: W3C Working Group Note, http://www.w3.org/TR/2004/NOTE-soap12-af-20040608/.

[151] *Web Services Addressing 1.0 - Core.* World Wide Web Consortium (W3C), 2006. Status: W3C Recommendation, http://www.w3.org/TR/2006/REC-ws-addr-core-20060509/.

[152] Roland Wagner. *A model for the digital representation and transaction of complex pricing and ordering for high-value spatial products and services.* PhD thesis, Technical University Berlin, 2003.

[153] *Web Coverage Service (WCS), Version 1.0.0 (Corrigendum).* Open Geospatial Consortium (OGC), 2005. http://www.opengeospatial.org/standards/wcs.

[154] *Web Feature Service Specification 1.1.* Open Geospatial Consortium (OGC), 2005. http://portal.opengeospatial.org/files/?artifact_id=8339.

[155] Narada Wickramage and Sanjiva Weerawarana. A benchmark for web service frameworks. In *SCC '05: Proceedings of the 2005 IEEE International Conference on Services Computing*, pages 233–242, Washington, DC, USA, 2005. IEEE Computer Society.

[156] P. Wohed, W.M.P. van der Aalst, M. Dumas, and A.H.M. ter Hofstede. Analysis of Web Services Composition Languages: The Case of BPEL4WS. *22nd International Conference on Conceptual Modeling (ER 2003)*, 2813:200–215, 2003.

[157] Bill Woodward and Arliss Whiteside, editors. *Feature Portrayal Service*. Open Geospatial Consortium (OGC), September 2005. OGC Discussion Paper, `http://portal.opengeospatial.org/files/?artifact_id=13186`.

[158] *Web Services Description Language (WSDL) 1.1*. World Wide Web Consortium (W3C), 2001. Status: W3C Note, `http://www.w3.org/TR/2001/NOTE-wsdl-20010315`.

[159] *XML-Signature Syntax and Processing*. World Wide Web Consortium (W3C), 2002. Status: W3C Recommendation, `http://www.w3.org/TR/2002/REC-xmldsig-core-20020212/`.

[160] *XML Encryption Syntax and Processing*. World Wide Web Consortium (W3C), 2002. Status: W3C Recommendation, `http://www.w3.org/TR/2002/REC-xmlenc-core-20021210/`.

[161] Sebastian Zimmer. Authorization, accounting and charging (with paypal). Diploma thesis, University of the German Armed Forces, Munich, October 2007.

List of Figures

List of Tables

Listings

Appendices

Appendix A

SOAP Message Examples

A.1 Land Register Information System

A.1.1 WFS GetFeature Request

The following listing shows a GetFeature SOAP request requesting GML features of type Kataster and Flurstueck. The SOAP header of the request contains a WS-Security header which includes a username token. This is used in the authentication process.

```
<SOAP-ENV:Envelope xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/">
 <SOAP-ENV:Header>
  <wsse:Security xmlns:wsse="...">
   <wsse:UsernameToken xmlns:wsu="..." wsu:Id="UsernameToken-12864175">
    <wsse:Username>john_test</wsse:Username>
    <wsse:Password Type="...#PasswordDigest">2W3Yp3WX21CGM4nX0V+11U9vb3o=</wsse:Password>
    <wsse:Nonce>u9ue3VrobUibsYCPI5Sv4Q==</wsse:Nonce>
    <wsu:Created>2007-01-15T13:55:50.328Z</wsu:Created>
   </wsse:UsernameToken>
  </wsse:Security>
 </SOAP-ENV:Header>

 <SOAP-ENV:Body>
  <wfs:GetFeature xmlns:wfs="http://www.opengis.net/wfs"
   outputFormat="GML2" service="WFS" version="1.0.0">
   <wfs:Query typeName="Kataster">
     ... Filter Information Here ...
   </wfs:Query>
   <wfs:Query typeName="Flurstueck">
     ... Filter Information Here ...
   </wfs:Query>
  </wfs:GetFeature>
 </SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

Listing A.1: WFS request containing authentication information

A.1.2 WFS GetFeature Response

Listing A.2 shows a result to the previously listed request, where all GML features of type Kataster are encrypted. For simplicity reasons only one feature from each feature class is showed, and most of the feature attributes are omitted.

Please pay attention to the header of the SOAP message: because asymmetric encryption is slow, an AES123-CBC key is generated, this one is asymmetrically encrypted, and the GML features are encrypted using AES key. This encryption scheme saves time for both the encryption and the decryption process.

```
<soapenv:Envelope xmlns:soapenv="..." xmlns:xenc="..." xmlns:xsd="..." xmlns:xsi="...">
  <soapenv:Header>
   <wsse:Security xmlns:wsse="...">
    <xenc:EncryptedKey Id="EncKeyId-10736847">
     <xenc:EncryptionMethod Algorithm="http://www.w3.org/2001/04/xmlenc#rsa-1_5"/>
      <ds:KeyInfo xmlns:ds="http://www.w3.org/2000/09/xmldsig#">
       <wsse:SecurityTokenReference>
        <ds:X509Data>
         <ds:X509IssuerSerial>
          <ds:X509IssuerName>CN=Tom,OU=INF3,O=UniBw,L=Munich,ST=BY,C=DE</ds:X509IssuerName>
          <ds:X509SerialNumber>12</ds:X509SerialNumber>
         </ds:X509IssuerSerial>
        </ds:X509Data>
       </wsse:SecurityTokenReference>
      </ds:KeyInfo>
      <xenc:CipherData>
       <xenc:CipherValue>MZRB... at==</xenc:CipherValue>
      </xenc:CipherData>
      <xenc:ReferenceList>
       <xenc:DataReference URI="#EncDataId-11646147"/>
      </xenc:ReferenceList>
     </xenc:EncryptedKey>
    </wsse:Security>
 </soapenv:Header>

 <soapenv:Body>
  <featureCollection xmlns="..." xmlns:gmgml="..." xmlns:gml="...">
   <gml:boundedBy>
    <gml:Box srsName="EPSG:31467">
     <gml:coordinates>3476227.969,5367755.176 3477566.626,5368168.248</gml:coordinates>
    </gml:Box>
   </gml:boundedBy>
   <gml:featureMember>
    <Flurstueck fid="Flurstueck.8239">
     <STRID> 2 </STRID>
     <Geometry1_SK> 1BLHZ^Z/ </Geometry1_SK>
     <Geometry1>
      <gml:LineString srsName="EPSG:31467">
       <gml:coordinates> 3477565.436,5367829.68 3477562.343,5367824.654
         3477562.343,5367789.333 3477564.035,5367786.055 3477566.626,5367757.027
         3477555.733,5367755.176 3477555.441,5367756.59 </gml:coordinates>
      </gml:LineString>
     </Geometry1>
    </Flurstueck>
   </gml:featureMember>

   <gml:featureMember>
    <Kataster Id="id-11646147" fid="Kataster.43968">
     <xenc:EncryptedData Id="EncDataId-11646147"
     Type="http://www.w3.org/2001/04/xmlenc#Content">
     <xenc:EncryptionMethod Algorithm="http://www.w3.org/2001/04/xmlenc#aes128-cbc"/>
      <xenc:CipherData>
       <xenc:CipherValue>boSL...GYY=</xenc:CipherValue>
      </xenc:CipherData>
     </xenc:EncryptedData>
    </Kataster>
   </gml:featureMember>
 </soapenv:Body>
</soapenv:Envelope>
```

Listing A.2: WFS response containing encrypted GML features

Listing A.3 shows the result to the previously listed request, where a digital signature is created for all GML features of type Kataster. Similarly, for simplicity reasons only one feature from each feature class is showed,and most of the feature attributes are omitted.

```
<soapenv:Envelope xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/">
 <soapenv:Header>
  <wsse:Security xmlns:wsse="...">
   <ds:Signature Id="Signature-10688173" xmlns:ds="http://www.w3.org/2000/09/xmldsig#">
    <ds:SignedInfo>
     <ds:CanonicalizationMethod Algorithm="http://www.w3.org/2001/10/xml-exc-c14n#"/>
     <ds:SignatureMethod Algorithm="http://www.w3.org/2000/09/xmldsig#rsa-sha1"/>
     <ds:Reference URI="#id-6440356">
      <ds:Transforms>
       <ds:Transform Algorithm="http://www.w3.org/2001/10/xml-exc-c14n#"/>
      </ds:Transforms>
     <ds:DigestMethod Algorithm="http://www.w3.org/2000/09/xmldsig#sha1"/>
     <ds:DigestValue>STYKsvSV1JM/qavrZconm+Qa0yA=</ds:DigestValue>
    </ds:Reference>
   </ds:SignedInfo>
   <ds:SignatureValue>Re2A...ak=</ds:SignatureValue>
   <ds:KeyInfo Id="KeyId-19253663">
    <wsse:SecurityTokenReference wsu:Id="STRId-30926415" xmlns:wsu="...">
     <ds:X509Data>
      <ds:X509IssuerSerial>
       <ds:X509IssuerName>CN=LMIS,OU=INF3,O=UniBw,L=Munich,ST=Bayern,C=DE</ds:X509IssuerName>
       <ds:X509SerialNumber>100011</ds:X509SerialNumber>
      </ds:X509IssuerSerial>
     </ds:X509Data>
    </wsse:SecurityTokenReference>
   </ds:KeyInfo>
  </ds:Signature>
 </soapenv:Header>

 <soapenv:Body>
  <gmgml:featureCollection xmlns:gmgml="http://www.intergraph.com/geomedia/gml">
   <gml: xmlns:gml="http://www.opengis.net/gml">
    <gml:Box srsName="EPSG:31467">
     <gml:coordinates>3476227.969,5367755.176 3477566.626,5368168.248</gml:coordinates>
    </gml:Box>
   </gml:boundedBy>

   <gml:featureMember xmlns:gml="http://www.opengis.net/gml">
    <gmgml:Flurstueck fid="Flurstueck.8239">
     <gmgml:STRID>2</gmgml:STRID>
     <gmgml:Geometry1_SK>1BLHZ^Z/</gmgml:Geometry1_SK>
     <gmgml:Geometry1>
      <gml:LineString srsName="EPSG:31467">
       <gml:coordinates>3477565.436,5367829.68 3477562.343,5367824.654 3477562.343,
        5367789.333 3477564.035,5367786.055 3477566.626,5367757.027 3477555.733,
        5367755.176 3477555.441,5367756.59 </gml:coordinates>
      </gml:LineString>
     </gmgml:Geometry1>
    </gmgml:Flurstueck>
   </gml:featureMember>

   <gml:featureMember xmlns:gml="http://www.opengis.net/gml">
    <gmgml:Kataster Id="id-6440356" fid="Kataster.43968">
     <gmgml:HALINID>43968</gmgml:HALINID>
     <gmgml:MALFD>43634</gmgml:MALFD>
     <gmgml:Geometry1_SK>1BLHZ^vKF</gmgml:Geometry1_SK>
     <gmgml:Geometry1>
      <gml:LineString srsName="EPSG:31467">
       <gml:coordinates>3476234.43,5368168.248 3476227.969,5368152.664</gml:coordinates>
```

```
          </gml:LineString>
        </gmgml:Geometry1>
      </gmgml:Kataster>
    </gml:featureMember>
  </gmgml:featureCollection>
 </soapenv:Body>
</soapenv:Envelope>
```

Listing A.3: WFS response containing digitally signed GML features

Appendix B

UML Notation

Some of the diagrams in this document are presented using the Unified Modeling Language (UML) 2.0. The rest of this appendix presents the notations.

B.1 UML Class Diagrams

The UML class diagram notations used in this document are exemplified in figures B.1, B.2 and B.3.

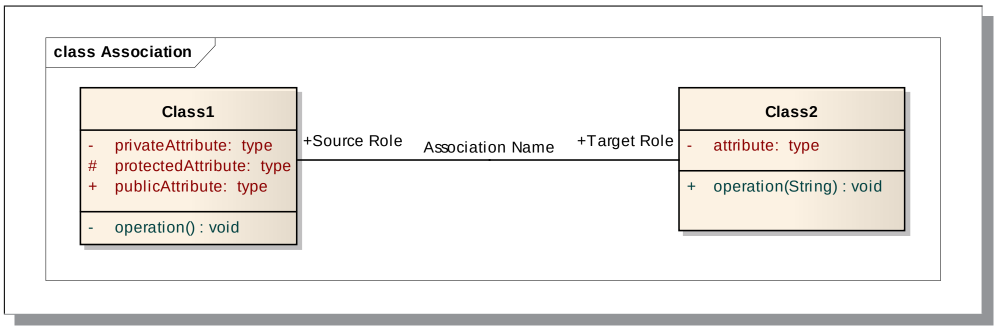

Figure B.1: Association between classes

In the diagrams, the following stereotypes are used:

- <<abstract>> A stereotyped class that cannot be instantiated, but can be subclassed. The class can include both operations and attributes. Some of the operations defined may be abstract.
- <<interface>> A definition of a set of operations that is supported by objects having this interface. An Interface class cannot contain any attributes
- <<type>> A stereotyped class used for specification of a domain of instances (objects), together with the operations applicable to the objects. A Type class may have attributes and associations.

In this document, the following standard data types are used:

- Boolean - A value specifying TRUE or FALSE
- Date - A timestamp, containing both date and time

- Double - A double precision floating point number
- Integer - An integer number
- String - A sequence of characters
- URI - An identifier of a resource that provides more information

In addition to those, the following non-standard types are used:

- ArbitraryContent - An arbitrary content (in a message)
- Configuration - Information describing configuration parameters for a service
- Message - A message
- State - Information describing the state of an object / service

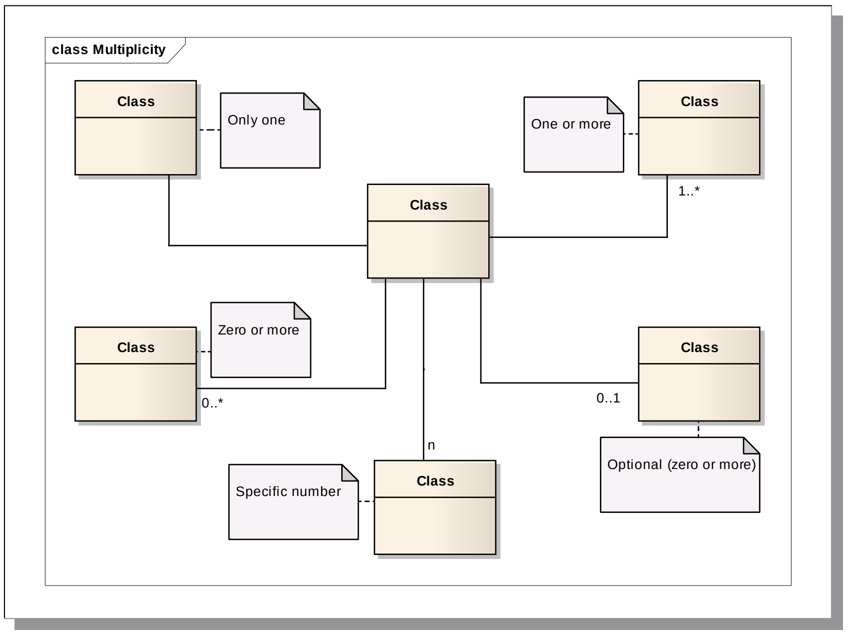

Figure B.2: Association cardinality

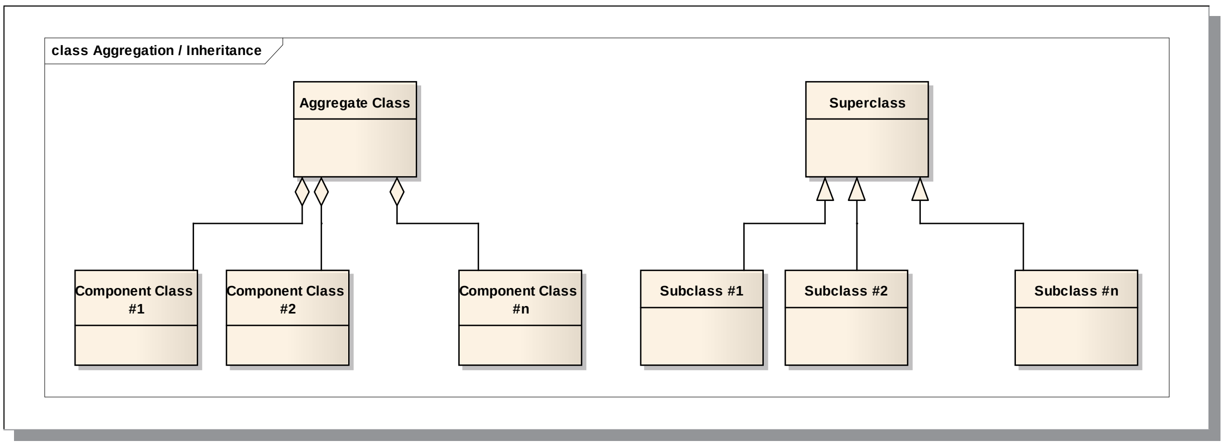

Figure B.3: Aggregation between classes and class inheritance

B.2 UML Sequence Diagrams

The basic elements of an UML sequence diagram, as used in this dissertation, are presented in figure B.4.

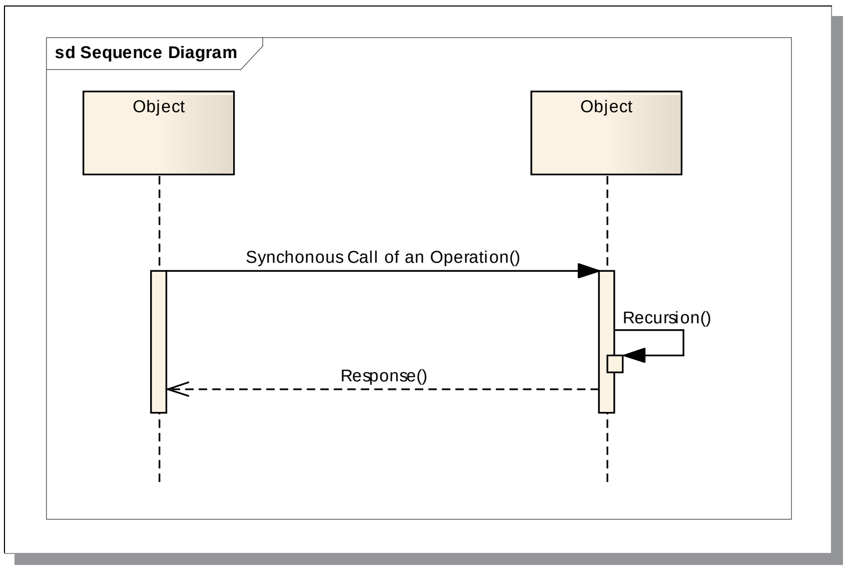

Figure B.4: UML sequence diagram

B.3 UML Activity Diagram

The basic elements of an UML activity diagram, as used in this dissertation, are presented in figure B.5.

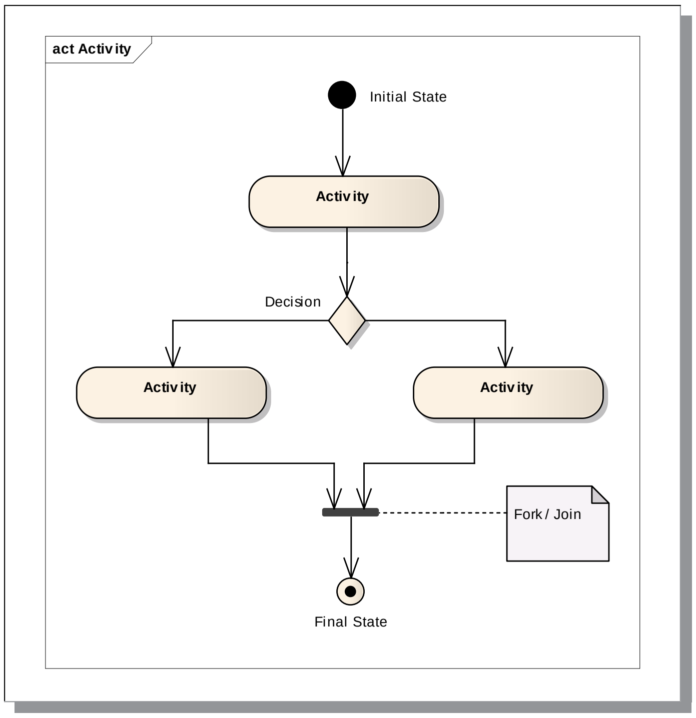

Figure B.5: UML activity diagram

Appendix C

Graphical Notations

Throughout this dissertation various diagrams are used. The graphical elements used in these diagrams are consistent throughout the dissertation, forming a kind of graphical language. The purpose of this is to provide clarity and make the reading easier. The elements of the different types of diagrams are illustrated in this appendix.

C.1 Service Diagrams

The main elements used in service diagrams are displayed in figure C.1. Figure C.2 shows the representation of communication between components.

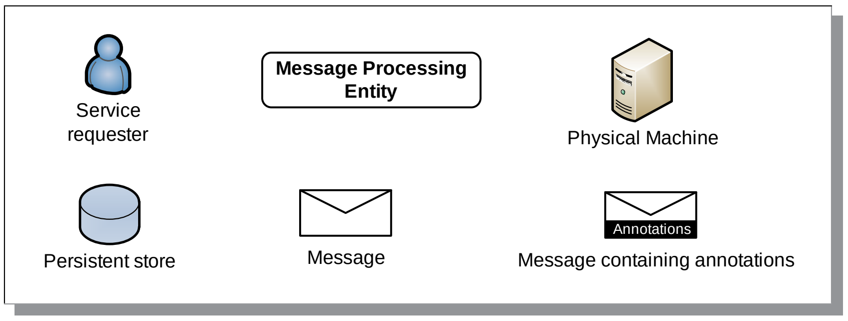

Figure C.1: Elements used in service diagrams

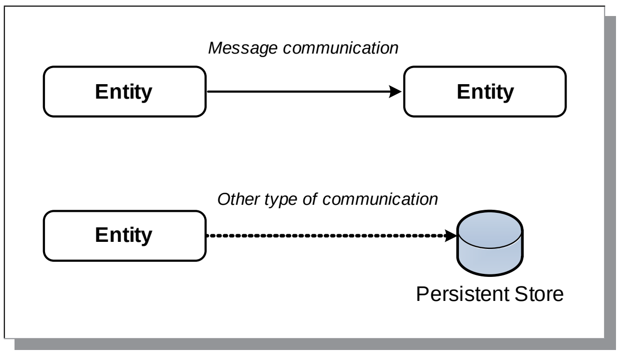

Figure C.2: Communication between entities in service diagrams

C.2 Message Patterns

The graphical notation used to represent message patterns is represented in figure C.3. The graphical elements are consistent with the graphical notations used in the literature (for example [95, 56]). The patterns are explained in section 2.3.5.

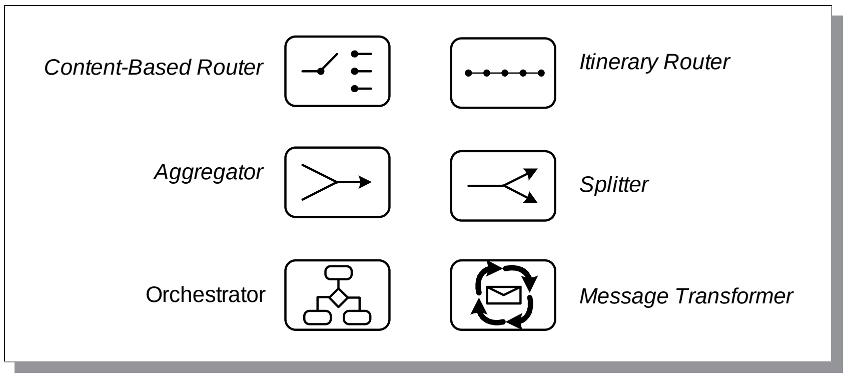

Figure C.3: Message routing patterns

C.3 Information Model Diagrams

Figure C.4 shows an example of an information model diagram. The most important diagram elements are explained (see red comments in boxes).

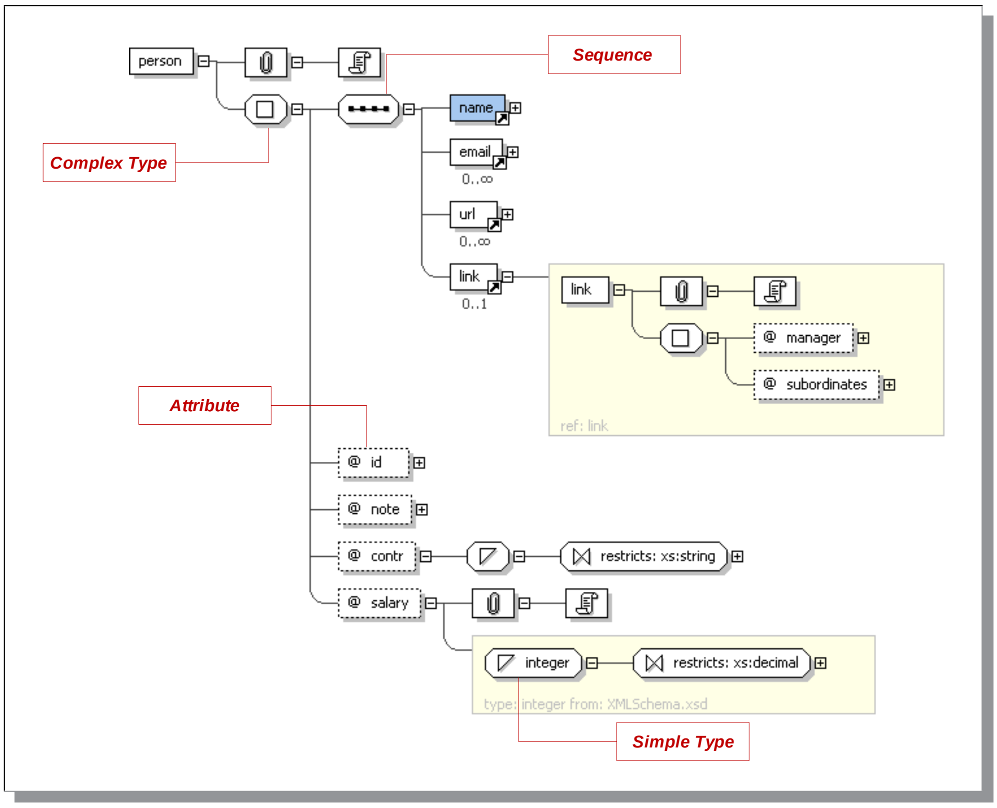

Figure C.4: Graphical notation for information model diagrams

Appendix D

Testing Environment

The following is the description of the hardware and software used for the performance evaluation of the ***SOS!e*** framework (see section 7.2).

D.1 Servers - Dell PowerEdge 750

- Single Intel Pentium 4, 2.8GHz Processor
- 2GB DDR-400MHz SDRAM
- 2 x 80GB SATA @7200RPM Disk Drives configured for RAID 1 (Mirroring)
- 2 x Gigabit NICs (only one of them was configured)

D.2 Network Switch - Dell PowerConnect 2424

- 24 x 10/100Base-T Ports
- 2 x 1000Base-T ports
- Switching Capacity: 8.8 Gbps
- Forwarding Rate: 14.5 Mpps

D.3 Software

- SuSe Linux 10.2 (Kernel 2.6.18.2-34)
- Apache Tomcat 5.5.23
- Apache AXIS 1.4
- Mule 1.4.0
- JMeter 2.3RC3

Appendix E

Abbreviations

AAA	Authentication, Authorization and Audit
A4C	Authentication, Authorization, Audit, Accounting and Charging
B2B	Business to Business
B2C	Business to Client
BioLANCC	Biometric Local Area Network Control Center
DCP	Distributed Computing Platform
EAI	Enterprise Application Integration
ESB	Enterprise Services Bus
GML	Geometry Markup Language (see [84, 85])
ISO	The International Organization for Standardization
JVM	Java Virtual Machine
KVP	Key-Value Pair
MOM	Message Oriented Middleware
RM-ODP	Reference Model for Open Distributed Processing (ISO standard)
OASIS	Organization for the Advancement of Structured Information Standards
OGC	Open Geospatial Consortium
OWS	OGC Web Service
OWS-3, OWS-4	OGC Web Services Test Bed 3,4
PAP	Policy Administration Point (see XACML [18])
PDP	Policy Decision Point (see XACML [18])
PEP	Policy Enforcement Point (see XACML [18])
PIP	Policy Information Point (see XACML [18])
PRP	Policy Retrieval Point (see AAAArch [147])
QoS	Quality of Service
RTT	Round-Trip-Time (delay)
SAML	Security Assertions Markup Language (see [22])
SOA	Service Oriented Architecture
SOSA	Service Oriented Security Architecture
SOS!e	Service Oriented Security - an Implementation Experiment (prototype implementation for ***SOSA***)
UML	Unified Modeling Language
UniBW	University of the German Armed Forces (Universität der Bundeswehr)
W3C	World Wide Web Consortium
XACML	eXtensible Access Control Markup Language (see [13, 18])
XML	eXtensible Markup Language

Appendix F

Curriculum Vitae

Cristian-Aurel OPINCARU

Education

Dr. Rer. Nat. (2008)
University of the German Armed Forces, Munich, Germany
Computer Science Department, Institute for Information Systems
Advisor: Prof. Dr. Gunnar Teege
Thesis: Service Oriented Security Architecture applied to Spatial Data Infrastructures

Computer Science Engineer (2003)
Polytechnic University of Bucharest, Romania
Major: Distributed Systems
Thesis: A Security System for the Targeteam System

Analyst (1998)
St. Sava High School, Bucharest, Romania

Professional Experience

10/2003 - 11/2007	Research Associate at the Institute for Information Systems, Computer Science Department, University of the German Armed Forces, Munich, `http://www.unibw.de`
10/2004 - 10/2005	Developer at Intergraph GmbH Munich, Germany (part-time), `http://www.intergraph.de`
10/2001 - 02/2003	Teaching Assistant at the Polytechnic University of Bucharest Bucharest, Romania, `http://www.acs.pub.ro`
10/2000 - 12/2001	Java Programmer at WebCab Components Ltd. Bucharest, Romania, `http://www.webcabcomponents.com`
08/1999 - 09/2000	Programmer at Icemenerg S.A. Bucharest, Romania, `http://www.icemenerg.ro`

Publications

Books

- Valeriu Iorga, Cristian Opincaru, Corina Stratan and Paul Chirita. Data Structures and Algorithms - Aplications in C++ using STL, Polirom Publishing House, 2005, Romanian language, 352 pages. ISBN: 973-681-872-1
- Valeriu Iorga, Paul Chirita, Cristian Opincaru and Corina Stratan - C/C++ Programming Problems, Niculescu Publishing House, 2003, Romanian language, 256 pages. ISBN: 973-568-800-X
- Livia Toca, Andreea Demco, Cristian Opincaru and Adrian Sindile. Informatics text book for the 10th grade, Niculescu Publishing House, 2000, Romanian language, 156 pages, ISBN: 973-568-422-5

Industry Standards / Papers

- Cristian Opincaru (Editor), Andreas Matheus, Christian Elfers, Jan Martin Senne and Roland Wagner. Trusted Geo Services IPR. February 2006
- Jan Drewnak, Christian Elfers, Roland M. Wagner and Cristian Opincaru. GeoDRM Engineering Viewpoint and supporting Architecture IPR. February 2006

- Joshua Lieberman, Roderick Morrison, Cristian Opincaru, Ugo Taddei and Roland M. Wagner. OWS-3 GeoDRM Thread Activity and Interoperability Program Report: Access Control & Terms of Use (ToU) Click-through IPR Management, February 2006
- Graham Vowles, John R. Herring, Cecil Goodwin, Mark V. Scardina, Andreas Matheus, Cristian Opincaru and Wayne Debeugny. The OpenGIS®Abstract Specification Topic 18: Geospatial Digital Rights Management Reference Model (GeoDRM RM), January 2006
- Cristian Opincaru. A general concept of associating credentials with OWS messages. Position paper within the OGC GeoDRM WG. November 2005
- Andreas Donaubauer, Cristian Opincaru, Stephan Plabst, Julia Stahl. Verfeinerung der Leitungsauskunft - Abschlussbericht. Runder Tisch GIS. November 2005
- Cristian Opincaru. Preliminary Study: Securing OpenGIS Web Services. Runder Tisch GIS. January 2004

Articles

- Cristian Opincaru, Gabriela Gheorghe. Service Oriented Security Architecture. EMISA 2007: Enterprise Modelling and Information Systems Architectures, Lecture Notes in Informatics. October 2007
- Cristian Opincaru et. al. Trendanalyse zur INTERGEO vom 10. 12. Oktober 2006 in Mnchen. zfv 6/2006, pp. 370 - 376
- Gunnar Teege, Cristian Opincaru, Andreas Donaubauer. The OGC Test Platform of Runder Tisch GIS e.V. Information about architecture, use and operation. Runder Tisch GIS. October 2006
- Cristian Opincaru et. al. Trendanalyse zur INTERGEO 2005. PFG 1/2006, pp. 67 - 74

Industry Fairs

- Project: *Security for OGC Web Services.* SYSTEMS 2006, Munich, Germany
- Projects: *Click-Through Licensing for GeoWeb Services, Biometric Authentication for GeoWeb Services.* SYSTEMS 2005, Munich, Germany
- Project: *Click-Through Licensing for GeoWeb Services.* INTERGEO 2005, Düsseldorf, Germany

Organized Workshops

- Sicherheit für Geo Web Services (4h). Speakers: Andreas Matheus, Cristian Opincaru and Gunnar Teege. Fortbildungsseminar 2007, Munich, Germany, March 2007
- Security for Geo Web Services: From Problem Statement to Implementation (4h). Speakers: Cristian Opincaru, Andreas Matheus. GITA Conference 2007, San Antonio, TX, USA, March 2007
- GeoDRM Workshop (8h) Speakers: Andreas Matheus and Cristian Opincaru. Organized for the Bavarian Mapping Agency, Munich, 2005
- Security for Geo Web Services (3h). Speaker: Cristian Opincaru. Organized for the Municipality of Strasbourg, France, February 2006

Invited talks

- *Trusted Geo Services IPR.* 62th OGC TC Meeting, Ottawa, Canada, April, 2006
- *Authentication in OWS4.* 61th OGC TC Meeting, San Diego, CA, USA, December, 2006
- *Trusted Geo Services IPR / OGC# 06-107.* 61th OGC TC Meeting, San Diego, CA, USA, December, 2006
- *GeoDRM: Keeping Free and Remaining Open*, with Joshua Lieberman. Free and Open Source Software for Geoinformatics Conference (FOSS4G2006), Lausanne, Switzerland, November, 2006
- *Security for Geo Web Services: Problems & Solutions.* GeoWeb Conference (GeoWeb2006), Vancouver, Canada, July, 2006.
- *GeoDRM: Enabling Technologies.* 59th OGC TC Meeting, Edinburgh, United Kingdom, June, 2006
- *[Interoperable] Digital Rights Management for Geospatial Web Services.* OASIS Symposium 2006, San Francisco, CA, USA, May, 2006.
- *Sicherheit bei Service-basierten Geodateninfrastrukturen*, with Andreas Matheus. Cegi OGC Interoperabilittstag, Bonn, Germany, November, 2005
- *Requirements & Use-Cases in OWS3.GeoDRM*, with Joshua Lieberman. 55th OGC TC Meeting, Bonn, Germany, November, 2005
- *Implementation of the University of German Armed Forces in OWS3.* 55th OGC TC Meeting, Bonn, Germany, November, 2005
- *Security for OGC Web Services.* GDI NRW AG Security Meeting, Münster, Germany, May, 2005
- *A SOAP-Based Security Framework for Securing OGC Web Services.* 53th OGC TC Meeting, Frascati, Italy, April, 2005

Organizing Committee / Review Committee

- 41th Hawaii International Conference on System Sciences (HICSS 2008) - *Review Committee*
- 10th IEEE Integrated Network Management Conference (IM 2007) - *Organizing Committee*
- 22nd Annual Computer Security Applications Conference (ACSAC 2006) - *Review Committee*

Professional Associations / Organizations

- Association for Computing Machinery (ACM)
- Geospatial Information & Technology Association (GITA)
- Open Geosaptial Consortium (OGC) - GeoDRM WG, Security WG, XACML RWG