Web services for application development in next generation telecommunications networks: An architecture for the common functions

Fatna Belqasmi

A Thesis

in

The Department

of

Electrical and Computer Engineering

Presented in Partial Fulfillment of the Requirements
for the Degree of Master of Applied Science at
Concordia University
Montréal, Québec, Canada

August 2004

© Fatna Belqasmi, 2004

Library and Archives Canada

Published Heritage Branch

395 Wellington Street
Ottawa ON K1A 0N4
Canada

Bibliothèque et Archives Canada

Direction du Patrimoine de l'édition

395, rue Wellington
Ottawa ON K1A 0N4
Canada

Your file Votre référence
ISBN: 0-612-94693-2
Our file Notre référence
ISBN: 0-612-94693-2

The author has granted a non-exclusive license allowing the Library and Archives Canada to reproduce, loan, distribute or sell copies of this thesis in microform, paper or electronic formats.

The author retains ownership of the copyright in this thesis. Neither the thesis nor substantial extracts from it may be printed or otherwise reproduced without the author's permission.

L'auteur a accordé une licence non exclusive permettant à la Bibliothèque et Archives Canada de reproduire, prêter, distribuer ou vendre des copies de cette thèse sous la forme de microfiche/film, de reproduction sur papier ou sur format électronique.

L'auteur conserve la propriété du droit d'auteur qui protège cette thèse. Ni la thèse ni des extraits substantiels de celle-ci ne doivent être imprimés ou aturement reproduits sans son autorisation.

In compliance with the Canadian Privacy Act some supporting forms may have been removed from this thesis.

While these forms may be included in the document page count, their removal does not represent any loss of content from the thesis.

Conformément à la loi canadienne sur la protection de la vie privée, quelques formulaires secondaires ont été enlevés de cette thèse.

Bien que ces formulaires aient inclus dans la pagination, il n'y aura aucun contenu manquant.

Canada

ABSTRACT

Web services for application development in next generation telecommunications networks: An architecture for the common functions

Fatna Belqasmi

Next generation telecommunications networks aim to provide new value added services to customers. To facilitate the creation of such services, network operators have to open up their networks to applications developers. The service architectures that can be used to develop new services are either signaling protocol neutral or signaling protocol specific. The associated frameworks to each of these architectures share the drawback of requiring knowledge which non-experts may not have. A good way to open up telecommunications networks is to use web services, seen that web services paradigm ease new services development and a wide range of developers are getting acquainted with the technology.

This thesis proposes a novel web services based - architecture that provides functions required for developing, deploying and using web services in telecommunications networks (e.g. security and charging). These functions are called common functions. The proposed architecture is based on requirements that we have identified using as basis the Open Mobile Alliance (OMA) ones, which we have extended. It presents each common function as a web service. The architecture components inter-communicate using functions calls. The execution of the common functions is optimized using the execution scope concept that we have defined. It gives the possibility to execute a given common function just when needed.

A proof of concept prototype has also been built and tested using a call control scenario. The taken performance measurements have shown that the overhead induced by using the new proposed architecture is reasonable.

ACKNOWLEDGEMENTS

I would like to express my gratitude to my supervisors Dr. Rachida Dssouli and Dr. Roch Glitho for their support, their guidance and their motivation. I thank Dr. Dssouli for her kindness, here encouragement and her good advices. I thank Dr. Glitho for his continuous assistance and for always being their when i need him. I learned a lot from working with him.

I'm grateful to Ericsson and SNERS (The Natural Sciences and Engineering Research Council of Canada) for their financial support. I thank Ericsson for giving me the opportunity to work in its Research and Development department. I thank Concordia University for giving me the chance to follow one of its interesting programs.

I would like to thank all my colleagues at the TSE lab at Ericsson Canada for their help and their suggestions. It has been a good pleasure to work with all of you. I also thank my friend Saloua Ouassil for her support.

I would also like to express my gratitude to my parents, my brothers and my sisters for encouraging me and for always having confidence in me.

Table of Contents

LIST OF FIGURES

LIST OF TABLES

LIST OF ACRONYMS AND ABBREVIATIONS

OMA: Open Mobile Alliance

QoS: Quality of Service

PSTN: Public Switched Telephone Network

PSDN: Public Switched Digital Network

VoIP: Voice over IP

IP: Internet Protocol

SIP: Session Initiation Protocol

ITU-T: International Telecommunication Union - Telecommunication

IETF: Internet Engineering Task Force

LAN: Local Area Networks

RSVP: Resource reSerVation Protocol

RTP: Real-time Transport Protocol

SDP: Session Description Protocol

HTTP : Hyper Text Transfer Protocol

IN : Intelligent Network

PBX: Private Branch eXchange

CGI: Common gateway Interface

API: Application Programming Interface

JAIN: Java APIs for Integrated Networks

JCC/JCAT: Java Call Control/Java Coordination and Translation

OSA: Open Service Architecture

NGN: Next Generation Network

WSDL: Web Services Description Language

SOAP: Simple Object Access Protocol

XML: Extensible Markup Language

HTML: HyperText Markup Language

RPC: Remote Procedure Call

UDDI: Universal Description, Discovery and Integration

CPXe: Common Picture eXchange environment

HP: Hewlett-Packard

GIS: Geographic information systems

SSL: Secure Session Layer

TLS: Transport Layer Security

WS-Security: Web Service Security

SAML: Security Assertion Markup Language

WS-DBC: Services Domain Boundary Controller

P3P: Platform Privacy Preferences

SLA: Service Level Agreement

WS-I: Web Services Interoperability forum

UML: Unified Modeling Language

IDL: Interface Description Language

CORBA: Common Object Request Broker Architecture

SSP: Service switching point

HLR: Home Location Registry

SCF: Service Capability Features

WSLA: Web Services Level Agreement

SSO: Single Sign On

OASIS: : Organization for the Advancement of Structured Information Standards

URL: Uniform Resource Locator

W3C: World Wide Web Consortium

WSE: Web Services Enhancements

WSLA: Web Service Level Agreement

ID-FF: Identity Federation Framework

ID-WSF: Identity Web Services Framework

ID-SIS: Identity Services Interfaces Specifications

Chapter 1:

Introduction

This chapter gives an introduction to the research area. It starts by introducing the next generation telecommunications networks. It presents the relating protocols and service architectures and introduces the use of web services in next generation networks. Then, it presents the goals of the thesis and ends with the presentation of the thesis organization.

I.1 Application development in next generation telecommunications networks

This section is about next generation telecommunications networks. It starts by a definition of the next generation networks. Then, it presents the next generation signaling protocols. After that, it presents the service architectures for next generation networks and introduces the use of web services in these networks.

I.1.1 Definitions

Next generation networks refer to networks with internet telephony capabilities, third generation networks and beyond [1]. They are especially packet switching based, QoS enabled and integrate voice and data applications. They seamlessly blends the public switched telephone network (PSTN) and the public switched data network (PSDN) to create a single multi-service network.

The internet telephony (VoIP) uses the Internet Protocol (IP) to transmit voice as packets over an IP network [2]. There are mainly two sets of standards for internet telephony: H.323 and Session Initiation Protocol (SIP).

I.1.2 Signaling protocols

Next generation signaling protocols include mainly H.323 from the ITU-T [3] and Session Initiation Protocol (SIP) from IETF [4]. This section gives an overview of each of them.

I.1.2.1 H.323

H.323 is an ITU-T standard [3]. It was initially targeted to multimedia conferencing over Local Area Networks (LANs) that do not provide guaranteed quality of service (QoS). It provides a foundation for audio, video and data communications over Ip-Networks.

H.323 is an umbrella recommendation since it includes various other ITU-T standards. It includes signaling and call control standards (e.g., H.245, Q.931, and H.225), video processing standards (e.g. H.261), data conferencing standards (e.g. T.120), media transportation standards (e.g. H.235) and other supplementary standards (e.g., H.450).

H.323 architecture is composed of the terminal, the gatekeeper, the gateway and the multipoint control unit. The terminal is an end-point that provides real-time two-way communications with another H.323 end-point, gatekeeper or multipoint control unit. The gatekeeper is the entity that provides address translation and controls how a terminal accesses the network. The gateway is used to provide real-time communications between H.323 terminals on the packet-based networks and terminals in the Public Switched Telephony Network (PSTN). The multipoint control unit provides multipoint conferencing capabilities.

I.1.2.2 Session Initiation Protocol (SIP)

The session initiation protocol (SIP) is an IETF signaling protocol for the establishment, modification and tear down of multimedia sessions [4]. These sessions include multimedia conferences, internet phone calls and multimedia distribution. SIP can also be used to invite participants or add media to existing sessions. It can be used in conjunction with other IETF protocols such as RSVP [5] for QoS management, RTP for media transportation [6] and others (e.g. SDP for multimedia sessions description [7]).

SIP uses a textual encoding for its messages; it is based on HTTP and is a request/response protocol. Its functional entities are user agents, proxy servers, redirect servers and registrars [8]. User agents are the end-points that initiate requests and are usually their destination. Proxy servers are application level routers. Redirect servers redirect clients to an alternate server, and registrars keep track of users within their assigned network domain.

I.1.3 Service architectures

There are three types of service architectures for developing applications for next generation networks. We have the today's signaling protocol neutral and signaling protocol specific architectures and the emerging web services based architecture. These architectures are presented here after.

I.1.3.1 Signaling protocol specific architectures

Signaling protocol specific service architectures are used with specific signaling protocols. There are two categories of these architectures: H.323 specific architecture and SIP specific architecture.

The H.323 specific architecture is based on pre-IN (Intelligent Network) and PBX thinking. It is based on a supplementary service approach and it includes no service creation framework. It is concerned with the standardization of services instead of services capabilities. The mainly supplementary services standardized so far are included in the H.450 recommendations (e.g. call transfer, call diversion, and call forwarding).

The H.323 specific architecture is highly unsuitable for next generation networks, since it provides only a limited range of services and it does not allow third parties. It is more suitable for PBX environments with low expectations of services.

For the SIP specific architecture, two service creation frameworks exist: SIP Common gateway Interface (CGI) and SIP servlet API. SIP CGI [9] is similar to HTTP CGI which makes it appealing to web programmers. It is language independent and it gives full access to all fields of SIP request. It also provides access to environment variables. Therefore, it allows the development of a wide range of services. However, SIP CGI is not exactly the same concept as HTTP CGI and the use of CGI is less and less used in the web world.

SIP Servlet [10] relies on HTTP Servlet that is widely used in the internet world. It primarily targets experienced and trusted developers. Just like SIP CGI, it provides the possibility to create a wide range of services. It can be used to create services that combine HTTP and SIP. Nevertheless, it is JAVA dependent and it is not exactly the same thing as HTTP Servlet. Its APIs are more complicated and their usage requires more knowledge.

I.1.3.2 Signaling protocol neutral Architectures

They are service architectures that can be applied to networks using any signaling protocol, including H.323 and SIP. They are applicable to both circuit switched telephony and next generation networks.

Three main signaling protocol neutral frameworks exist: JAIN's JCC/JCAT, Parlay and Call Processing Language (CPL). Java Call Control (JCC)/ Java Coordination and Translation (JCAT) [11] is a JAIN community product. It's applicable to

SS7/ISUP/TCAP, SIP and H.323 and it provides access to call control capabilities. Nowadays, it has lost momentum to Parlay/OSA.

Parlay/OSA [12] open up telecommunications networks to application developers. It presents two types of APIs: services APIs and framework APIs. The services APIs expose telecommunications network capabilities (e.g. call control, presence) for application development. Framework APIs make the use of the services APIs secure and resilient by providing necessary functions such as security management, event notification and integrity management.

Parlay/OSA gives possibility for creating a wide range of services including those combining different network capabilities. However, it presents a low level of abstraction and it is not easy to grasp by people with no circuit switching telephony background.

Call Processing Language [13] specifies an architecture that aims at service creation by end-users and targets primary un-trusted parties. However, very few end-users are interested in creating services and the range of services that can be created is limited.

I.1.3.3 Emerging web services based architectures

Web services are becoming the preferred solution for program to program interactions. They provide standard means to allow interoperability between different software applications implemented in diverse programming languages and running on various platforms [14]. Web services technology provides a high level of abstraction. This allows applications developers to develop and integrate needed functionalities to their applications easily and rapidly.

Web services provide a good way to open up telecommunications networks to application developers, by supporting new services development and a widely acceptance of the technology among developers.

When using web services in the telecommunications domain, two issues are to be solved. The first one is to expose telecommunication capabilities (e.g. call control, presence, messaging) as web services. We have to define web services for making telecommunications capabilities available to applications in the same or foreign domain. The second issue is to identify the common functions to web services when used in telecommunications settings (e.g. security, charging). This will ensure interoperability among systems and also help developers to build innovative and robust applications in a minimum time by focusing only on the application development.

I.2 Goals assigned to the thesis

This thesis focuses on the second issue relating to the usage of web services in the telecommunications domain. It is concerned with the common issues encountered by developers when using web services in telecommunications networks.

Many functions are common to web services when used in telecommunications settings (e.g. security, charging). The goals of this thesis are:

- Identifying the set of the common functions
- Identifying the requirements to be fulfilled by the solution providing these functions
- Proposing an architecture for providing the common functions in telecommunications networks
- Implementing a proof of concept prototype
- Taking some performance measurements.

The architecture we are proposing is based on web services and we call the resulting solution a framework. This framework can be defined as a consistent environment for building and running web services applications. It provides a set of technologies and functions that can be used to publish, discover and use web services in a secure, controlled and auditable manner. It provides the common functions needed to facilitate the development and usage of web services.

I.3 Organization of the thesis

The rest of this thesis is organized as follows: Chapter 2 introduces the requirements for common functions. It starts with a global overview of web services principals, architecture, technologies and some application areas. It also presents the identified requirements.

Chapter 3 gives an overview of the state of the art. It presents the Parlay framework solution and the existing web services standards and solutions. It also provides an analysis of these solutions in light of the identified requirements.

Chapter 4 is dedicated to the proposed solution. It starts by stating some assumptions on which the solution architecture relies. Then, it describes the novel architecture we are proposing and shows how client applications can be executed using this architecture.

Chapter 5 is about the proposed architecture optimization. It specifies how to optimize the framework execution and the execution of each of its functional entities.

Chapter 6 presents the implemented prototype and some performance measurements. It includes also an analysis of these measurements.

Chapter 7 concludes the thesis by recapitulating the major contributions of the thesis and discussing some items for future work.

Chapter 2:

Requirements for common functions

This chapter is composed of two sections. The first section gives a general overview of web services. The second section presents the identified requirements for common functions.

II.1 Introduction to Web Services

This section gives a global introduction to web services. It starts with a brief definition of web services. Then, it presents fundamental principals of web services and their architecture. After that, it outlines the different technologies needed for web services usage.

II.1.1 Definition

"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" [14].

The web service interface description provides necessary information to interact with the service. It includes service location, message formats, operations description, transport protocols and bindings. The bindings describe how to map the service interface and its associated operations to a particular concrete message format and transmission protocol. Web services provide standard means of interoperability between various applications, running on different platforms. Web service interfaces hide the implementation details of the service, allowing it to be easily used independently of the platform on which it is implemented and independently of the programming language used.

II.1.2 Fundamental principals

Web services have three fundamental principles [15] that make their usage very beneficial. These principals are as follow:

1. Coarse grained approach: The web service technology provides a high level of abstraction, which allows developers to integrate needed functionalities to their applications easily and rapidly.
2. Loose coupling: Applications developed using web services are loosely coupled, which makes them independent. For instance, if an application "a" is talking to an application "b" and "b" is modified, the application "a" should not necessarily be re-written. This loose-coupling between applications provides more flexibility, scalability, and extensibility.
3. Synchronous and asynchronous mode of communication: Web service applications support both synchronous and asynchronous mode of communication.

II.1.3 Architecture

Web services architecture is based on the interaction between three principal roles: service provider, service requestor and service registry (Figure:II.1). The service provider creates a web service, defines its description and publishes this description to the web service registry. The service requestor finds out the web service, retrieves its description from the service registry, and then, using this description, it binds the request to the service implementation and starts interacting with it.

The service description may also be given directly to the service developer. It can be provided on a CD with a product, downloaded from the service provider web site or provided via other means.

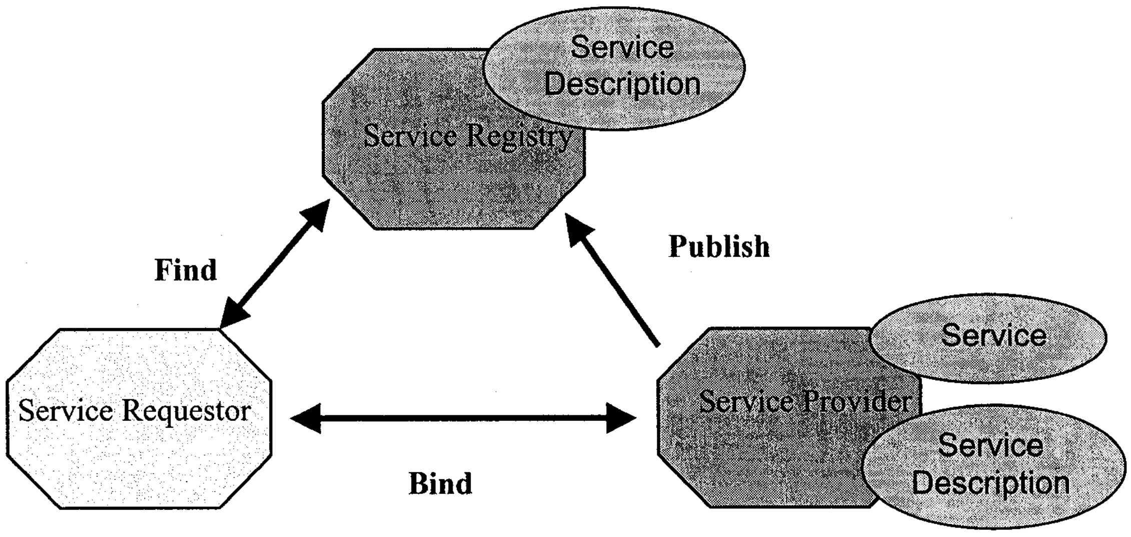


Figure II.1: Web services roles

II.1.4 Web services technologies

The main technologies involved in the web services implementation and usage are shown in the figure bellow [14]:

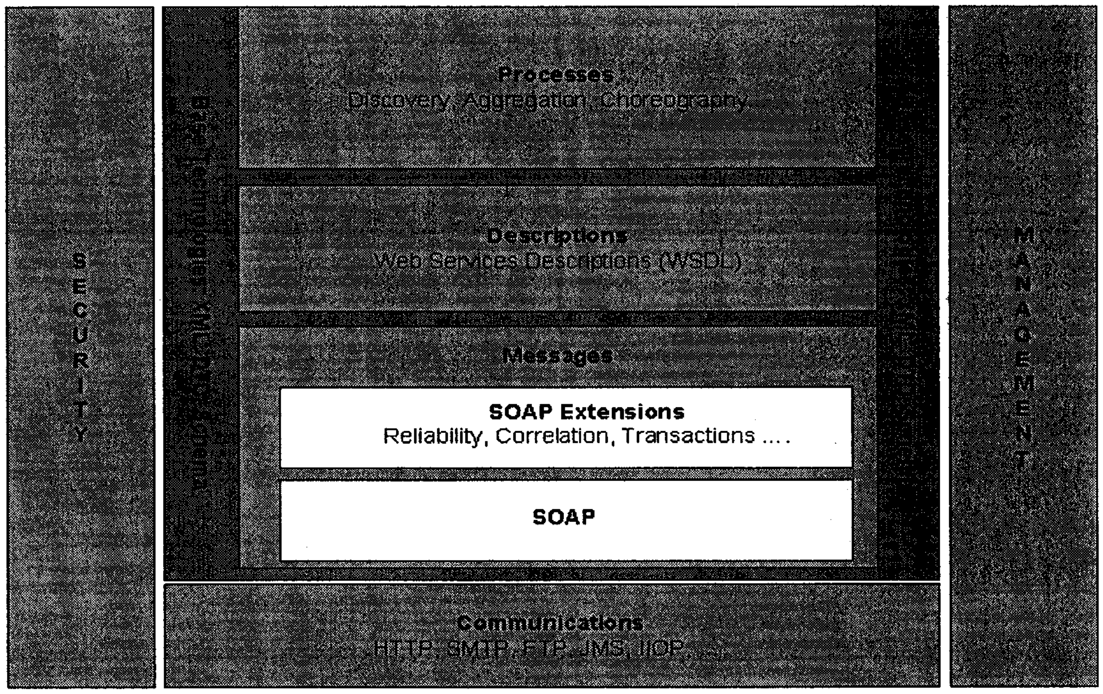

Processes
Discovery, Aggregation, Choreography
Descriptions
Web Services Descriptions (WSDL)
Messages
SOAP Extensions
Reliability, Correlation, Transactions ....
SOAP
Communications
HTTP, SMTP, FTP, JMS, IIOP
SECURITY
MANAGEMENT
Base Technologies: XML, DTD, Schema

Fig II.2: Web services technologies

- **XML [16]**: Extensible Markup Language is a simple and very flexible text format. It enables the presentation of data in a structured and unambiguous way. It is platform independent and it plays an important role in the exchange of a wide variety of data on the Web and elsewhere.

  XML makes use of tags just like HTML. However, in HTML, both tag semantics (<p> means paragraph) and tag set are fixed, contrary to XML. XML is extensible and it uses the tags only to delimit data peaces. The tags interpretation is leaved to the processing application.

- **SOAP [17]:** Simple Object Access Protocol is a lightweight protocol for exchanging typed and structured information in a decentralized and distributed environment. It is based on XML and it consists of three parts: an envelope, a set of encoding rules and a RPC representation. SOAP messages are fundamentally one-way transmissions from a sender to a receiver, but they are often combined to implement patterns such as request/response.

The SOAP message is an XML document and it consists of a mandatory SOAP envelope, an optional SOAP header, and a mandatory SOAP body. The SOAP envelope is the top element of the XML document representing the message, and the SOAP header includes some attributes to indicate who the header recipient is, how the message should be processed, and whether the header is optional or mandatory for the recipient to process. The SOAP body element provides a container for mandatory information intended for the ultimate recipient of the message.

The SOAP encoding rules define a serialization mechanism that can be used to exchange instances of application-defined datatypes. The SOAP encoding style is based on a simple type system that is a generalization of the common features found in type systems in programming languages, databases and semi-structured data. A type either is a simple type or is a compound type constructed as a composite of several parts, each with a type

The SOAP RPC representation defines a convention that can be used to represent remote procedure calls and responses. It relies on the protocol binding to provide a mechanism for carrying the information. SOAP can potentially be used in combination with a variety of other protocols; however, the widespread bindings defined describe how to use SOAP in combination with HTTP.

Example: SOAP Message Embedded in HTTP Request

This example presents a SAOP message used to connect to the server www.stockquoteserver.com and get the last trade price:

```
POST /StockQuote HTTP/1.1
Host: www.stockquoteserver.com
Content-Type: text/xml; charset="utf-8"
Content-Length: nnnn
SOAPAction: "Some-URI"
<SOAP-ENV:Envelope
   xmlns:SOAP- ENV=http://schemas.xmlsoap.org/soap/envelope/
    SOAP- ENV:encodingStyle="http://schemas.xmlsoap.org/soap/encoding/">
```

```
        <SOAP-ENV:Body>
            <m:GetLastTradePrice xmlns:m="Some-URI">
                <symbol>DIS</symbol>
            </m:GetLastTradePrice>
        </SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

- **WSDL [18]:** Web Services Description Language (WSDL) provides a model and an XML format for describing web services. A WSDL document is a set of definitions. The top level element of a WSDL document is a definitions element, that can encapsulate other definitions elements. A definitions element includes:
    - ***Types:*** provides datatype definitions used to describe the exchanged messages.
    - ***Messages***: describes the abstract format of a particular message that a Web service sends or receives, using the defined types. The format of a message is typically described in terms of XML element.
    - ***Operation:*** It's an abstract definition of an action supported by the service.
    - ***Interface:*** describes a set of messages that a service sends and/or receives. These messages are grouped into subsets named operations. An operation is a set of input and output messages, an interface is a set of operations.

- ***Bindings:*** describes a concrete binding of an interface element and associated operations to a particular concrete message format and transmission protocol.
- ***Endpoint:*** defines the particulars of a specific end-point at which a given service is available.
- ***Service:*** describes one and only one interface that a service provides, and the endpoints it is provided over.

WSDL document example:

```
<?xml version="1.0" encoding="UTF-8"?>
<definitions name="TicketAgent"
    targetNamespace="http://airline.wsdl/ticketagent/"
     xmlns="http://schemas.xmlsoap.org/wsdl/"
     xmlns:tns="http://airline.wsdl/ticketagent/"
     xmlns:xs="http://www.w3.org/2001/XMLSchema" xmlns:xsd1= "http://airline/">
     <import location="TicketAgent.xsd" namespace="http://airline/"/>
     <message name="listFlightsRequest">
          <part name="depart" type="xs:dateTime"/>
          <part name="origin" type="xs:string"/>
          <part name="destination" type="xs:string"/>
      </message>
      <message name="listFlightsResponse">
          <part name="result" type="xsd1:ArrayOfString"/>
      </message>
    <message name="reserveFlightRequest">
        <part name="depart" type="xs:dateTime"/>
        <part name="origin" type="xs:string"/>
        <part name="destination" type="xs:string"/>
        <part name="flight" type="xs:string"/>
    </message>
    <message name="reserveFlightResponse">
        <part name="result" type="xs:string"/>
    </message>
    <interface name="TicketAgent">
        <operation name="listFlights" parameterOrder="depart origin destination">
                <input message="tns:listFlightsRequest" name="listFlightsRequest"/>
                <output message="tns:listFlightsResponse" name="listFlightsResponse"/>
        </operation>
        <operation name="reserveFlight" parameterOrder="depart origin destination
        flight">
                <input message="tns:reserveFlightRequest"
                  name="reserveFlightRequest"/>
                  <output message="tns:reserveFlightResponse"
                  name="reserveFlightResponse"/>
        </operation>
   </interface>
</definitions>
```

- **UDDI [19]:** The Universal Description, Discovery and Integration specification defines the way to publish and discover web services on the net. It provides an open and independent platform enabling enterprises to easily and dynamically find commercial partners. The UDDI information model is composed of instances of the following entity types:
    - businessEntity: Contains descriptive information about the business or other organizations that typically provide web services, and information about the web service itself. It includes information such as names and descriptions in multiple languages, contact information and classification information, like activity sector and geographic zone.
    - businessService: Describes a collection of related web services offered by an organization described by a businessEntity. It represents a logical grouping of web services under a common rubric.
    - bindingTemplate: Describes the technical information necessary to use a particular web service. This concerns the information needed by applications to bind and interact with the web service being described.
    - tModel: Describes a "technical model" representing a reusable concept, such as a web service type, a protocol used by web services, or a category system.
    - publisherAssertion: Describes, in the view of one businessEntity, the relationship that the businessEntity has with another businessEntity. Indeed, many of the large businesses are represented by more than one businessEntity, since they may have many subsidiaries, and each

subsidiary has a different activity. However, these different businessEntity belongs to the same business, then it may be preferable to have a relationship between them

II.2 Application Areas

Web services can be applied to any area that requires program-to-program interactions over a network. This section gives some examples of such web services application areas.

II.2.1 Telecommunication

Web services can be used for value added service engineering in next generation telecommunications networks. They can be used to expose network capabilities (e.g. call control, presence, messaging) as web services, in order to make them available to applications in either the same or foreign domains.

Mainly, two sets of specifications are available in this domain. We have Parlay-X and Open Mobile Alliance (OMA) specifications. Parlay-X web services [20] are building blocks of telecommunications capabilities that applications developers can quickly comprehend and use to generate new and innovative applications. Parlay-X specifications are in their first version and only a few services are specified so far (e.g. call control, messaging, and terminal location). They aim to cover all telecommunications capabilities.

OMA specifications focus more on mobile services. They aim at providing solutions to common problems faced by the application designers when using web services in OMA environments (e.g. practical deployment patterns, security, charging, and tests requirements).

II.2.2 Digital imagery

In the digital imagery industry, the Common Picture eXchange environment (CPXe) [21] uses the web services paradigm to automate the manipulation, printing and sharing of digital images over a network. It was created by a large number of companies active in the digital imaging industry (e.g. Kodak, HP, Konica, Olympus and others).

The CPXe allows providers to register their services in a centralised directory, and gives them the ability to categorize these services. Its business model consists of three components: the requestor, the provider and the broker. The requestor is the application that discovers and uses digital imagery services. The provider is the entity that provides such services, and the broker is the entity that provides requestors with necessary information about published services.

There are two types of brokers: UDDI registry and the service locator. The service locator interacts with the UDDI and/or catalogues to find service(s) meeting specific criteria. The catalogue is a standardized way for service providers to provide more details about their services (e.g. pricing information, information about each retail store including street address and hours of operation). It is kept in the service provider domain.

II.2.3 Geographic information systems

Geographic information systems (GIS) are “computer-based information systems that enable capture, modeling, manipulation, retrieval, analysis, and presentation of geographically referenced data” [22]. They use multiple computer-science topics (e.g. databases, graphics, and computational geometry) to process queries about spatial data such as which location satisfies given requirements and what is a particular location.

The goal of GIS is to dynamically assemble applications from multiple GIS web services, for use in a variety of client applications. It provides means for combining results of complementary services to create customised applications. Used in the geographical information systems, the service model provides users with just data and functions needed in that particular application domain

II.3 Requirements

This section is about the web services framework to propose for providing common functions. It presents the set of requirements that we have identified as requirements to be fulfilled by this framework.

The requirements we have identified are based on the ones identified by the Open Mobile Alliance (OMA) [23]. We have extended these OMA requirements in order to make the use of the framework more flexible.

Beside the requirements concerning the common functions to be provided, we have identified other additional requirements that we have judged necessary for the consistency and the ease of usage of the proposed framework, and to support interoperability between applications using this framework.

II.3.1 Common functions requirements

This section presents the common functions to be provided by the framework and their relating technologies. The Common functions requirements are especially based on the OMA requirements.

II.3.1.1 OMA requirements

The OMA requirements on common functions include:

- **Security:** It is intended to meet the traditional security goals of reducing vulnerabilities of assets and resources. The security requirements include:
    - ***Authentication***: used to verify who the interacting party is. It can also be used to identify the message sender, the message recipient and the content signer. It can be ensured using SSL/TLS authentication with client & server certificate or using SOAP message security.
    - ***Data Integrity***: refers to the ability to detect whether the content has been changed since creation. It may be provided at different protocol layer:
        a. At transport layer using SSL/TLS.

    b. At Message layer using web service security (WS-Security [24]) to assure SOAP message integrity.
    c. At application layer using XML-Digital signature [25].

- ***Confidentiality:*** Prevents unauthorized parties from viewing information. It's typically provided using SSL/TLS, SOAP Messaging confidentiality and Application Level Confidentiality.
- ***Key Management***: key management functions are to provide secure key generation, storage, renewal, revocation, exchange and use. The appropriate standards to key management are Online Certificate Protocol [26] & XML Key Management Specification [27].
- ***Authorization***: used to determine whether an authenticated party is allowed to access a resource or perform some action. Normative requirements for authorization are: Security Assertion Markup Language (SAML [28]), WS-Security and SOAP message security.
- ***Non Repudiation***: is used to reduce the risk of repudiation to an acceptable level Using XML Digital Signature.
- ***Availability***: is a denial of service threat mitigation. It is used to assure the service to be available and not denied by attacks against the server. SSL/TLS & WS-Security are the mechanisms specified normatively for availability.

- **Network identity management**

Network identity refers to a set of information upon which one may receive personalized services [29]. This information consists of the overall attributes and references contained in the various accounts of individuals with different service providers. It includes personal information such as name, phone number, social security number and address, commercial information such as public key and certificate for signing information and other. Network identity can also be associated to devices and processes.

The widespread practice for network identity management is that each individual maintains its different accounts. This places a serious burden on the individual because one must manage the accuracy of multiple accounts information and remember multiple username/password pairs. Network identity management must make the identity control easier, while ensuring privacy and information security.

The framework must provide identity management and authentication to make the interaction between individuals and businesses easier. It must also be compatible with W3 web services standards like Liberty alliance project [38].

- **Privacy management**

Privacy has three aspects: personal, territorial and informational. The main issue in the OMA web services security context is the informational privacy. It is concerned with the protection of user privacy according to the existing privacy regulations. It is about end user's rights to determine how, when and to what extent personal information are communicated to other parties. It states and conveys privacy policies and enforces these policies.

The proposed framework should not add any additional privacy risks and should be associated to mechanisms that let the user specify the use of any information exchanged by mobile web services.

Informational privacy can be assured using the Platform Privacy Preferences (P3P [30]). Mechanisms designed to ensure confidentiality may also be used to reduce risks of inappropriate information disclosure.

- **Service management**

Service management requirements include service registration, service publication, service discovery, service Level Agreement Management, service delivery and service Integrity Management.

Service Level Agreement is a contract between the service provider and the service consumer that defines the rules and the conditions under which the service is supplied and the constraints on the quality level to be satisfied. Service Level Agreement management includes:

- Service usage negotiation: the service and the application negotiate the service capabilities to be supplied and the conditions under which the service is supplied.

- Service Level Agreement (SLA) creation: create the SLA according to the outcome of the previous stage. The SLA created is associated with the particular application and the particular service.
- Committing to the service level agreement: The consuming application and the service provider commit to the SLA, for example, by digitally singing and exchanging the SLA document created.
- Service level agreement provisioning: Agreed SLA is provisioned somewhere in the system for evaluation during service execution.

The service delivery is concerned by the SLA enforcement during the service usage. The service integrity management can be done using heartbeat management, load management and alarm management.

Service registration, publication and discovery may be ensured using a UDDI implementation. However, the use of UDDI for service management is not necessary, but when there is a need for distributed metadata publication & discovery, UDDI specifications are to be used. In this case, UDDI 2.03 Data Structure Schema must be used for web service description & UDDI 2.04 API is to be used for publishing and discovery.

- **Charging**

The framework must allow appropriate charging models such as [31]:

- User charging for service usage
- Service and content provider charging for service usage
- Charging of all actors in the value chain of a web service enabler usage (referral fee).

Furthermore, the framework must support various charging mechanisms, such as pay-per-usage or instance, pay-per-volume or data rate, and pay-per-subscription. It must also support various billing models such as percentage of transaction and pre-paid and post-paid payment types.

II.3.2 Other requirements

The additional requirements we have identified are composed of some OMA requirements and some other requirements that we have specified.

II.3.2.1 OMA requirements

OMA requirements relates especially to the common technologies to be supported by the framework in order to insure applications interoperability. They include the use of:

- XML-based messages for communication between various entities
- SOAP as a basic message format for communication between the service provider and the service requester.
- WSDL as a service interface description language

- HTTP as a recommended transfer mechanism for SOAP messages.

To assure interoperability between these different technologies, Web Services Interoperability Forum (WS-I) [43] recommends the use of:

- XML 1.0
- SOAP 1.0 & WSDL 1.0 as modified by the WS-I Basic Profile 1.0
- HTTP 1.1 (HTTP 1.0 may be used under certain circumstances).

II.3.2.2 Our requirements

Our additional requirements aim to make the use of the framework more flexible. They comprise three types of requirements which are:

- **As little impact as possible** on existing web services application development environment. This means that the application developers should not have to learn a new development environment in order to develop telecommunications applications.

  For instance, there are two architectural approaches for implementing security: Security libraries and security proxies

  - The framework have to enable interaction with services developed within each approach
  - In each case, the relevant security implementation must be sufficient to apply security.

- **Framework functions should be optional**: The network operator (i.e. the Web services provider) can either choose to handle a given common function by himself or delegate it to the framework

- **Plug and Play**: it should be possible to plug in existing web services products
  - Example:
    - UDDI may be easily plugged in to the framework without changing existing applications.
    - Security implementations may be automatically added to the framework.

II.4 Summary

In this chapter, we have seen that the common functions to be provided by the framework include security, network identity management, privacy management, service management and charging. We have discussed the meaning of each function and the required technologies for its implementation. The following chapter will give an overview of existing standards and solutions that provide some or all of these functions.

Chapter 3:

Parlay and existing web services standards: State of the art

This chapter presents an overview of the state of the art. It starts by an overview of the Parlay framework solution. After that, it presents some existing standards and solutions for using web services. It ends with discussion of the limitations of existing solutions.

III.1 Overview of Parlay framework

The Parlay/OSA specifications [12] define an architecture and a set of APIs that enable service and application developers to access the telecom network capabilities through an open standardized interface (i.e. the Parlay/OSA APIs). They are provided in Unified Modeling Language (UML) representation. Several mappings and mapping rules are defined for these specifications in several contexts including CORBA IDL, WSDL/SOAP and Java.

III.1.1 Parlay architecture

The Parlay architecture (Figure: III.1) consists of four major components: The application, the framework, the service capability features and the resources. The Parlay application is the client application that access the network capabilities (resources) using Parlay APIs. It is developed and deployed in a network and technologies independent manner.

The Parlay framework is the core of the Parlay architecture. It provides necessary functions to secure and control access to the service capability features. It also protects the network from applications misuse and provides functions for incremental introduction of new service capabilities.

Service capability features are entities that implement the Parlay APIs. They are composed of a service interface and a service object. A service interface provides access to the network capabilities exported by the Network Operator, whilst the service object provides a service interface implementation. Resources are the core network elements accessed through the Parlay service objects (e.g. SSP, HLR ...).

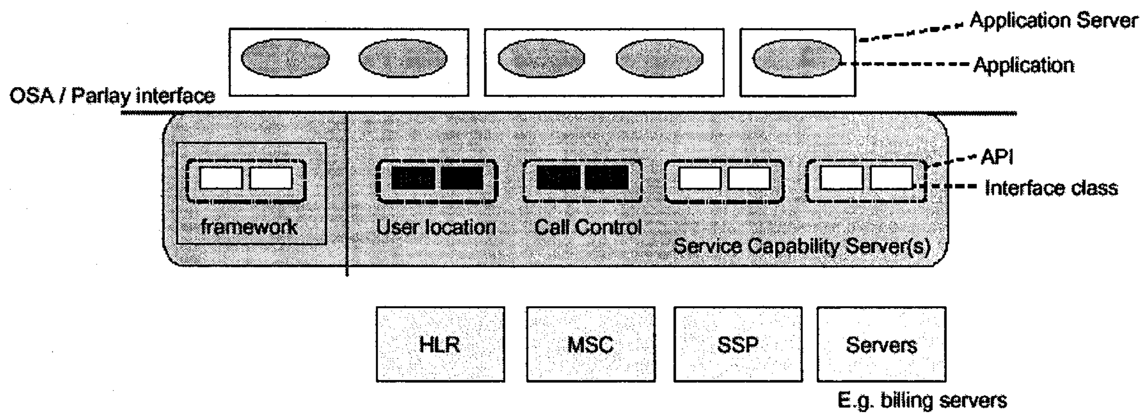


Fig III.1: Parlay/OSA Architectural Model

III.1.2 Parlay framework interfaces

Parlay/OSA framework provides mechanisms that enable applications to make use of network capabilities in a secure and controlled manner. It offers mechanisms for security and service management. Its interfaces are split into three distinct sets (Figure: III.2):

Framework to client application interfaces, framework to service capability features interfaces and framework to enterprise operator interfaces.

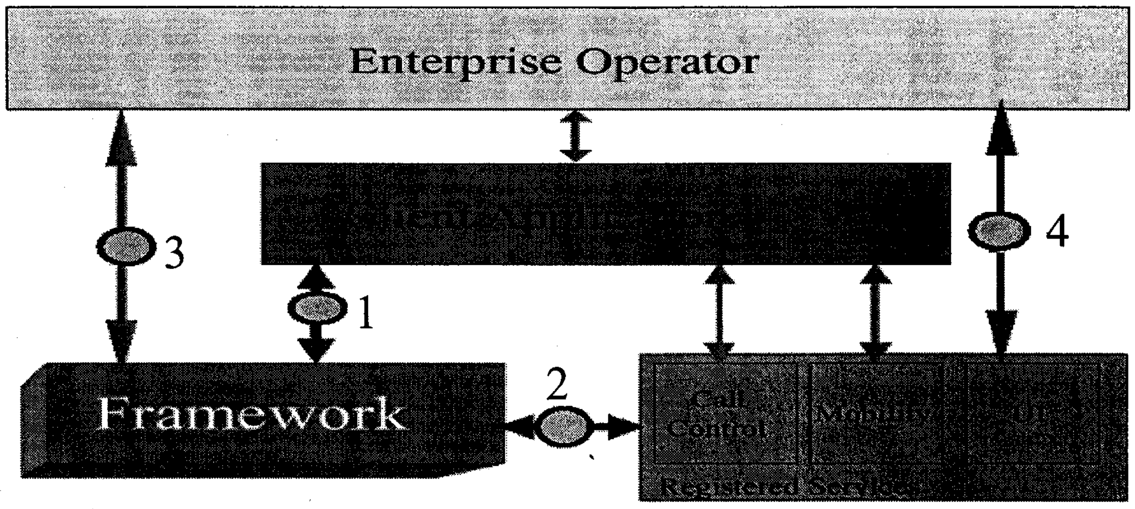


Fig III.2: Parlay framework interfaces

III.1.2.1 Framework-to-application interfaces

They provide applications with basic mechanisms that enable them to make use of the service capability features in the network. They include mechanisms for:

- **Authentication:** The client application must be authenticated before it's allowed to use Parlay/OSA interfaces. The Parlay authentication model is a peer-to-peer model. However, the authentication doesn't have to be mutual.

  The Parlay/APIs support multiple authentication techniques. The authentication mechanism may be supported by cryptographic process to provide confidentiality and by digital signature to ensure integrity. Other authentication mechanisms can be used like the underlying distributed technology mechanism. Application and framework may also recognise each other as a trusted party.

- **Authorization**: It determines what an authenticated application is allowed to do and which service capability features are allowed to be accessd.
- **Access control to service capability features**: It enables the framework to control application access to the service capability features (SCF).
- **Discovery of framework and network service capability features:** An authenticated application can obtain available framework interfaces. Furthermore, it can use the discovery interface to obtain information on authorized network service capability features.
- **Service agreement management:** After service discovery, the application identifies the SCF to use and obtains the service agreement (specifying the way to use the specified SCF) from the framework. The agreement may consist of an off-line and an on-line part. The on-line part is signed by the application and the framework.
- **Event notification**: Enables application notification, so that, subsequent framework events can be sent to the application. It can be used to notify the application about a new SCF, about SCFs that are no more available and about other events.
- **Integrity management:** Enables application integrity management using load management, heartbeat management and fault management.

III.1.2.2 Framework-to-services interfaces

Basic mechanisms between the framework and the service capability server [figure III.1] are:

- **Service registration:** Enables the registration and the publication of a new network service capability feature in the framework.
- **Service discovery:** Provides the service types supported by the framework (e.g. P_GENERIC_CALL_CONTROL, P_MULTI_PARTY_CALL_CONTROL, P_CONFERENCE_CALL_CONTROL, P_USER_INTERACTION ...), and the description of each service type. The description includes the properties associated with the service type, the super-types of the service type and whether the service type is available or not.
- **Event notification**: Used to notify the service of generic events that have occurred.
- **Integrity management**: Enables service integrity management via load management, heartbeat and fault management.

III.1.2.3 Framework-to-enterprise operator interfaces

In some cases, the client applications must explicitly subscribe to the service before they can use it. The subscription may be done by the enterprise operator on behalf of the client application. The enterprise operator represents an organisation or a company which will be hosting client applications, whilst, the subscription presents a contractual agreement between the enterprise operator and the framework operator.

To enable enterprise operators to subscribe to specific services before their client applications can use them, the Parlay framework provides a service subscription interface. The service subscription is performed on-line by the enterprise operator in the frame of an existing off-line negotiated contract. The contracted services may be provided to the enterprise operator directly by a service provider or indirectly through a retailer, such as a framework. The interaction between the Enterprise Operator, the client application and the framework is shown in the following figure.

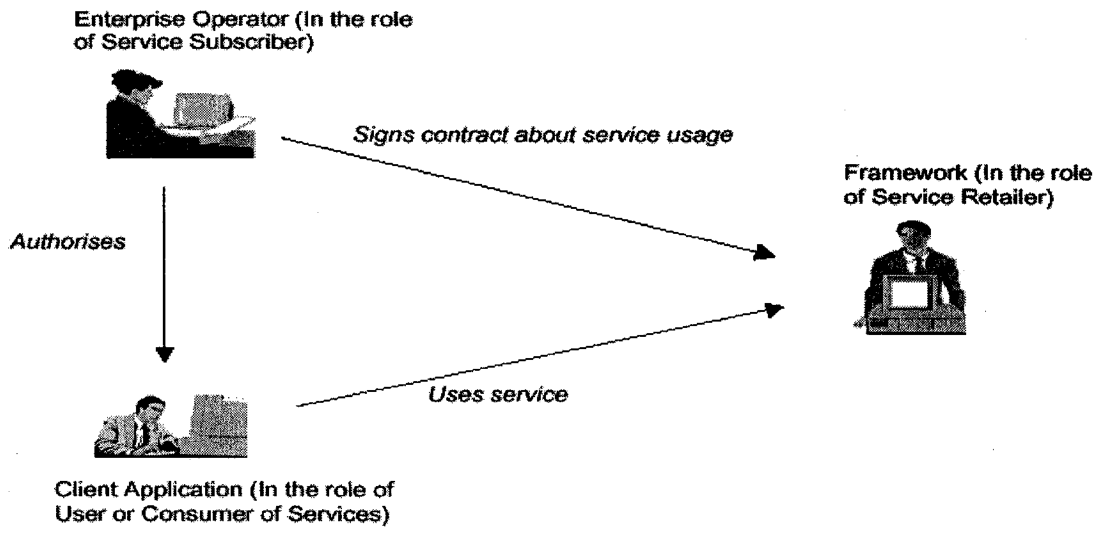

Enterprise Operator (In the role of Service Subscriber)
Signs contract about service usage
Framework (In the role of Service Retailer)
Authorises
Uses service
Client Application (In the role of User or Consumer of Services)

Figure III.3: Parlay subscription business model

III.1.3 Analysis

The Parlay framework provides solutions to some of the identified requirements. It provides support for the common supporting technologies and provides some solution for security and service management. However, some problems still need to be solved such as network identity management, privacy management and charging. The following table gives a summary of the analysis of the Parlay framework according to the requirements identified in the previous chapter.

Requirements			**provided**
Common functions	Security		Partial (Authorization, Authentication)
	Network Id		NO
	Privacy management		NO
	Service management	Service registration, publication & discovery	YES (but not using UDDI)
		SLA & Service Delivery	YES
		Service integrity	YES
	Charging		NO
Common supporting technologies (XML, SOAP,WSDL,HTTP)			YES
As little impact as possible on existing application development environments			NO
Framework functions optional			NO
Plug & Play			NO

Table III.1: Parlay framework analysis

III.2 Existing web services standards and solutions

This section gives an overview of existing standards and solutions for using web services. It starts with security standards including WS-Security and SAML. Then, it presents a Web Services Level Agreement (WSLA) solution. After that, it presents the Liberty Alliance specifications for network identity management.

III.2.1 Security

This section presents emerging web services security standards and provides an insight into the most important security models for SOAP.

III.2.1.1 Emerging standards

The emerging web services security standards are Web Services Security (WS-Security) and Security Assertion Markup Language (SAML). These two standards are presented here after.

III.2.1.1.1 Web Services Security (WS-Security)

WS-Security specification [24] proposes an abstract message security model for SOAP messages. It provides means to protect a message by encrypting and/or digitally signing a body, a header, an attachment or any part or combination of them. It defines a way of representing security credentials, such as username, password, public key certificate, security tickets, security tokens and digital signature using XML-based syntax. It also defines how to insert these security credentials in the SOAP message.

Technically, SOAP security is provided by the definition of a security header block. The latter provides a mechanism for attaching security-related information targeted at a specific receiver. The syntax of this header block is as follow:

```
<s:Envelope>
        <s:Header>
                ...
                < s:role="..." s:mustUnderstand="...">
                                ...
                </ wsse:Security>
```

```
            ...
        </s:Header>
        ...
</s:Envelope>
```

The s:role attribute identifies the recipient (named SOAP role) of the security block. It may be the ultimate message recipient or an intermediary. A SOAP message may contain multiple security headers, each targeted to a specific SOAP role. An intermediary may either add a new sub-element to the security block addressed to it or add one or more security header blocks to the SOAP message.

Expected elements to be used within the security block are:

- **Username Token element**: used to provide a user name and optional password information. The creation time may also be specified. Username token element syntax is as follow:

  ```
  <wsse:UsernameToken wsu:Id="...">
          <wsse:Username> ... </wsse:Username>
          <wsse:Password Type="..."> ... </wsse:Password>
          <wse:Created> ... </wsse:Created>
          ...
  </wsse:UsernameToken>
  ```

  This syntax is extensible and other attributes and elements can be added to this element.

- **Binary Security Tokens**: used to include binary-encoded security tokens (like X.509 certificates and Kerberos tickets) and non-XML format tokens. The syntax used is as follows:

```
<wsse:BinarySecurityToken wsu:Id=...
        EncodingType=...
        ValueType=.../>
```

EncodingType attribute specifies how the security token is encoded (e.g. Base64Binary) while the ValueType attribute indicates what the security token is (e.g. a Kerberos ticket).

- **Security Token Reference element**: used to reference the security token if it is not incorporated into the sent message.

```
<wsse:SecurityTokenReference wsu:Id="...">
        ...
</wsse:SecurityTokenReference>
```

WS-Security specification defines three specific reference mechanisms: direct references, key identifiers and key names. Direct reference is the most specific mechanism since it allows referencing security tokens by identifying the URI location where they can be found. To directly reference security tokens using URIs, the <wsse:Reference> element is added to the previous syntax. The direct references syntax corresponds to:

```
<wsse:SecurityTokenReference wsu:Id="...">
        <wsse:Reference URI="..." ValueType="..."/>
</wsse:SecurityTokenReference>
```

The key identifier allows tokens to be referenced using an opaque value (unique identifier) that represents the token. Key names mechanism allows tokens to be referenced using a string that matches an identity assertion within the security token.

- **ds-Key Info element**: carries the key information and is allowed for different key types and for future extensibility. However, if the key type contains binary data, it is recommended to use <wsse:BinarySecurityToken>.
- **ds-Signature element**: It conforms to XML-Signature specification and allows multiple signatures to be attached to a message. It includes the reference to the element to be signed, the digest method reference and the signature value.
- **Encryption element**: It leverages XML-Encryption standard. It includes the reference list of the encrypted elements, the encrypted key (encryption key encrypted using recipient's key) and the encrypted data.

When a sender or an active intermediary encrypts one or more portions of a SOAP message, they must present a <xenc:ReferenceList> sub-element to the corresponding security header block(s), to specify references to the portion(s) encrypted. If necessary, they must create a new <wsse:Security> header block, specify the intended recipient and insert the sub-element. Encrypted elements are replaced by corresponding <xenc:EncryptedData> according to XML encryption.

WS-Security specification allows different <xenc:EncryptedData> elements referenced by the same <xenc:ReferenceList> to be encrypted by different keys. The encryption key can be specified using <ds:KeyInfo> element within the corresponding <xenc:EncryptedData> element.

When the encryption is performed using a symmetric key (e.g a randomly generated symmetric key), the encryption key must be encrypted by the recipient public key and prepended to the security header within a <xenc:EncryptedKey> element.

- **Security Timestamps**: To prevent replay attacks, a <wsu:Timestamp> element is used to reference message timestamps. It indicates creation and expiration time of the message. Its schema is as follows:

```
<wsu:Timestamp wsu:Id="...">
        <wsu:Created ValueType="...">...</wsu:Created>
        <wsu:Expires ValueType="...">...</wsu:Expires>
</wsu:Timestamp>
```

  The Time type defined in XML-Schema is to be used. However, if other types are used, they must be specified using the ValueType attribute. The receipt time reference may also be included using the <wsu:Received> element.

WS-Security is intended to provide end-to-end security. It describes SOAP enhancements to provide message authentication, integrity and confidentiality. Message authentication and integrity are provided by XML-Signature in conjunction with security tokens. Message confidentiality leverages XML Encryption in conjunction with security tokens to keep portions of SOAP message confidential. Integrity mechanisms are designed to support multiple signatures (potentially by multiple SOAP roles) and additional signature format. Encryption mechanisms are designed to support additional encryption processes and operations by multiple SOAP roles.

Security Assertion Markup Language (SAML) assertions may be conveyed within the <wsse:Security> header block. The SAML token profile [34] specifies how SAML assertions can be used in the context of WS-Security specification. They are attached to SOAP messages using WS-Security by placing assertion elements or references to assertions inside the <wsse:Security> header. They may be referenced from different elements within a security header, or from the security header element itself. The preferred method to reference SAML assertions is by key identifier reference.

WS-Security includes other profiles for the use of other tokens within the WS-Security specification, such as Kerberos tickets.

III.2.1.1.2 Security Assertion Markup Language (SAML)

The Security Assertion Markup Language specification [28] defines an XML-based syntax for exchanging security information. This security information is expressed in the form of assertions and is issued by SAML authorities, who can use various information sources to create responses. They can use either the assertions received as input in request or information retrieved from external policy stores. The figure III.6 presents the SAML conceptual model [32].

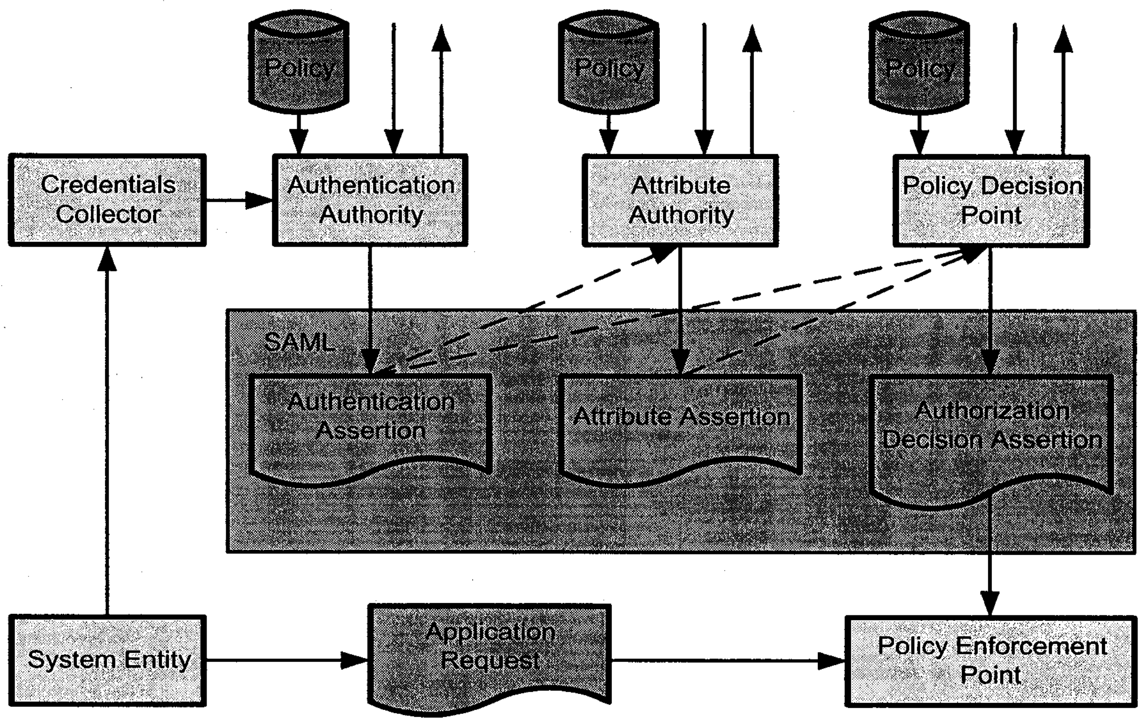


Figure III.4: SAML Conceptual Model

A SAML assertion is an XML construct that groups necessary information to describe one or more data elements made by an issuer. SAML specification defines three types of assertions:

- **Authentication assertion**: asserts that the specified subject was authenticated by particular means (specific authentication method and specific authentication authority) at a particular time.
- **Authorization Decision assertion**: states whether the specified subject has granted access to the specified resource or not. The set of authorization actions to be performed on the specified resource (Read, Write, Execute, Delete and/or Control) is also specified.

- **Attribute assertion:** states that the specified subject is associated with the specified attributes and the specified values.

These assertions have a nested structure. A single structure may convey several different statements about authentication, authorization decision and attributes. The schema of SAML assertion is as follow:

```
<element name="Assertion" type="saml:AssertionType"/>
<complexType name="AssertionType">
        <sequence>
                <element ref="saml:Conditions" minOccurs="0"/>
                <element ref="saml:Advice" minOccurs="0"/>
                <choice maxOccurs="unbounded">
                        <element ref="saml:Statement"/>
                        <element ref="saml:SubjectStatement"/>
                        <element ref="saml:AuthenticationStatement"/>
                        <element ref="saml:AuthorizationDecisionStatement"/>
                        <element ref="saml:AttributeStatement"/>
                </choice>
                <element ref="ds:Signature" minOccurs="0"/>
        </sequence>
        <attribute name="MajorVersion" type="integer" use="required"/>
        <attribute name="MinorVersion" type="integer" use="required"/>
        <attribute name="AssertionID" type="saml:IDType" use="required"/>
        <attribute name="Issuer" type="string" use="required"/>
        <attribute name="IssueInstant" type="dateTime" use="required"/>
</complexType>
```

The “AssertionType” element has different required attributes. “MajorVersion” and “MinorVersion” attributes are used to specify the highest and lowest assertion version supported. The values specified by SAML Core specification [32] for these attributes are respectively 1 and 0. The “AssertionID” attribute is the assertion unique identifier, “Issuer” presents the assertion issuer unambiguous name (may be a URI reference) and the “IssuerInstant” defines the time instant of an assertion issue.

The “AssertionType” element may contain conditions of assessing the assertion validity, additional information to assist processing (Advice), the XML-Signature to authenticates the assertion, authentication statement, authorization decision statement and attributes statement.

The “Conditions” element may provide information about the earliest time instant at which the assertion is valid, the time instant at which the assertion has expired and the specific audience to which the assertion is addressed.

“SubjectStatement” element describes the principal that is the subject(1) of the statement, by specifying its name identifier (name and security domain) and information that allow its authentication (e.g. the protocol to be used, cryptographic key held by the subject and additional information to be used by the authentication protocol).

----

(1) A subject is an entity (either human or computer) that has an identity in some security domain. A typical example of a subject is a person, identified by his or her email address in a particular Internet DNS domain.

SAML assertions can be exchanged using diver protocols, however, currently, SAML defines only SAML SOAP binding over HTTP [33]. This protocol binding specifies how SAML request and response exchanges are mapped into SOAP message exchanges and how these are mapped into HTTP message exchanges.

To enable communication between requestors and the authorities, SAML defines a request-response protocol [32]. This protocol consists of the XML-based request response message format. It may be used by clients to request assertions from SAML authorities via <request> element and get a <response> element from them. It can be bound to different underlying communication and transport protocols, but currently, only SOAP over HTTP bindings are specified.

SAML assertions, requests and responses may be signed using XML-Signature, to ensure authentication and message integrity. If a public-private key pair is used for the signature, non-repudiation of its origin is also provided.

One major goal of SAML is Single-Sign-On (SSO). It is the ability to authenticate only to one site (called source site) and access a secured resource on another site (named destination site) without directly authenticating to the latter. Various SAML profiles are designed to support different SSO scenarios and secure SOAP payload [33].

A SAML profile is a set of rules describing how SAML assertions are embedded in or combined with other objects (such as HTML data forms, files, or protocol data units), how they are sent to the destination site and how they are processed at the destination. The OASIS SAML Committee has specified two web browser-based profiles to support SSO, which are browser/POST profile and browser/artifact profile.

In case of the browser/POST profile, the SAML assertion is uploaded to the browser using an HTML data form and submitted to the destination site within the same form. The details of this interaction are shown here after (figure III.5):

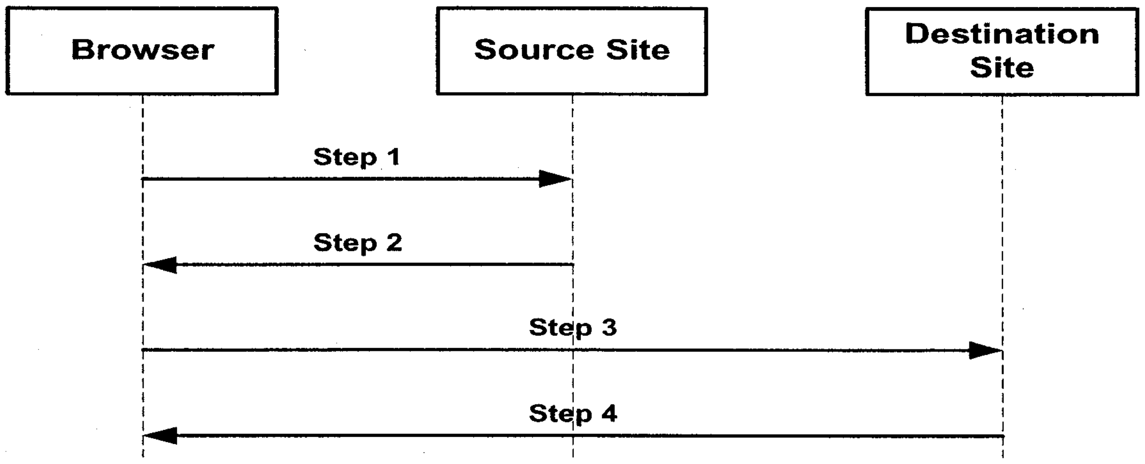


Figure III.5: SAML browser/POST profile sequence diagram

- In **step 1**, the user's browser accesses the source site with information about the target to reach at the destination site.
- In **step 2**, the source site generates a SAML response containing an SSO assertion within an HTTML form data. This form contains the URL of the targeted resource. The SSO assertion is used to refer to an assertion that:

    - Has a <saml:Conditions> element with NotBefore and NotOnOrAfter attributes present.
    - Contains one or more authentication statements.
- In **step 3**, the browser submits the generated form to the destination site.
- In **step 4,** the destination site responds to the browser request by granting him or denying him access to the desired resource.

In case of the browser/artifact profile, the user's authentication assertion is not included in the request. A small SAML artifact is attached to the URL query string, and then redirected by the source site to the destination site. This artifact is referencing unambiguously the user's authentication assertion. After receiving the request, the destination site acquires the authentication assertion via SAML protocol message exchanges with the source site, and sends an HTML response to the browser.

In the two cases, some security measures are taken in order to secure communication and avoid malicious attacks.

III.2.1.2 Security models

WS-Security and SAML specifies how to secure message exchanges, however, they do not define nor imply any specific implementation architecture. Therefore, it's necessary to have a look at the possible security architecture solutions.

There are two architectural approaches for implementing security aspects: libraries and proxies. This section presents the two approaches and gives an example for each of them.

III.2.1.2.1 Libraries approach

In case of the libraries approach, the application developer is supplied with appropriate security libraries that he has to use to program applications that perform security control. This approach has the drawback of instructing the application developer to correctly enforce security.

Example: Web Services Enhancements (WSE)

WSE is a class library proposed by Microsoft [35] for applying advanced web services protocols to SOAP message, including WS-Security, WS-Routing and WS-Attachment. Its usage entails reading headers from inbound SOAP messages and writing headers to outbound SOAP messages. It may also involve SOAP message body transforming like decrypting inbound message body or encrypting the outbound message body.

WSE functionalities are encapsulated by two sets of filters: inbound filters and outbound filters for respectively inbound and outbound messages. Outbound filters generate necessary SOAP headers for the message being created whereas inbound filters process incoming SOAP message headers.

WSE provides a toolset for implementing security within a SOAP message. It gives the possibility to authenticate SOAP message, verify its integrity and enables encryption using the mechanisms defined by WS-Security specification. It includes the use of the username token, the digital signature and the message encryption. It allows sending X.509 certificates as WS-Security Tokens. It can be used to build a wide range of applications, but it has the disadvantage of delegating much of the work required to integrate protocols with the application to the developer. To avoid this problem, a security proxy implementation approach may be used.

III.2.1.2.2 Proxy approach

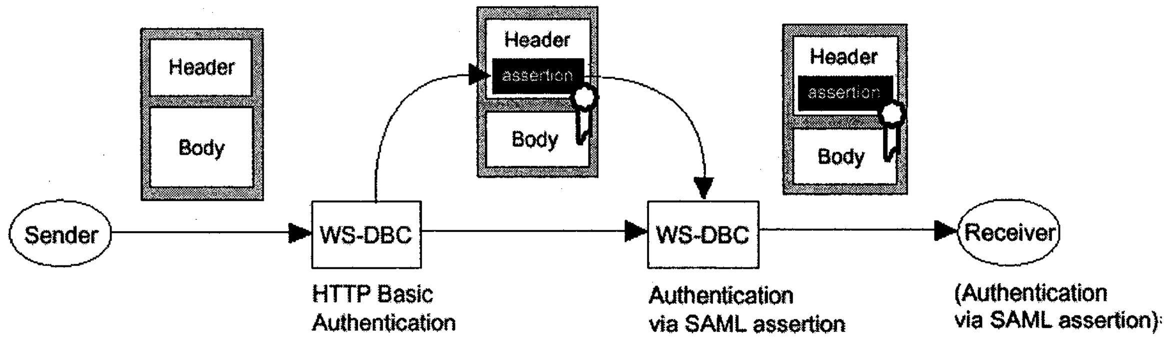
SOAP security proxy is an application level security gateway. It resolves the libraries approach problems since it can be easily deployed and transparently integrated into existing infrastructures.

Operating at the application level, the proxy-based solutions can perform advanced security checks such as XML-schema validation and content filtering. Furthermore, it reduces administration overhead by centralizing the management tasks.

Example: Web Services Domain Boundary Controller (WS-DBC)

WS-DBC is an application level security software proposed by Xtradyne [36]. It protects web services and SOAP messages. It adheres to the WS-Security [24] standards and relies on the SAML specification to ensure message integrity and confidentiality. For message authentication, WS-DBC uses SAML assertions located in the message header, SSL, X.509 certificates or simply HTTP basic authentication. It can also grant anonymous access or authenticates the message sender according to the IP source address.

In the case of an outgoing message, the WS-DBC inserts the security credentials into the message header, before forwarding it to the recipient or to another WS-DBC intermediary. The following example illustrates this process.



Header
Body
Header
assertion
Body
Header
assertion
Body
Sender
WS-DBC
WS-DBC
Receiver
HTTP Basic Authentication
Authentication via SAML assertion
(Authentication via SAML assertion)

Figure III.6: Authentication in WS-DBC

To provide more protection, WS-DBC can additionally sign SOAP messages by complying with the XML Digital Signature specification. It also enables authorization management, auditing and graphical administration (for security policies, audit event notification, public key management, SAML profile, access control …).

WS-DBC has the advantage of easily integrating with existing network infrastructure and applications, since it requires no modifications. It provides a complete A4 solution (Authentication, Authorization, Audit, Administration) without impairing existing applications performances.

III.2.2 Web Services Level Agreement

WSLA Framework of IBM is an example of web Services SLA management solution. It consists of a flexible and extensible language for describing the different SLA artifacts and a runtime architecture comprising of several SLA monitoring services [37]. To provide a higher level of objectivity, the WSLA Framework provides the possibility to outsource the SLA monitoring to third parties.

The WSLA language is based on an XML-Schema and describes the SLA structure in three sections: parties, service description and obligations. The parties section identifies all the involved parties, including the signatory parties (service customer and service provider) and supporting parties (third parties entrusted with SLA monitoring). The service description section specifies:

- The SLA parameters (throughput, service availability, response time ...)
- Services to which these parameters relate
- How these parameters are measured or computed (which functions and which metrics to use)
- How are the metrics of a managed resource accessed (typically by giving the Uniform Resource Identifier of a hosted computer program, a protocol message or the command for invoking scripts or compiled programs).

The service level obligations section defines the service level objectives to be guaranteed and the actions to be performed especially, when the SLA is violated.

The SLA management life cycle provided by the WSLA framework consists of [figure III.7]:

- SLA negotiation and establishment, which generates a signed SLA document

- Deployment of the entire or part of the SLA document to the involved parties
- Measurement and reporting of the SLA
- Execution of the corrective management actions when needed
- SLA termination.

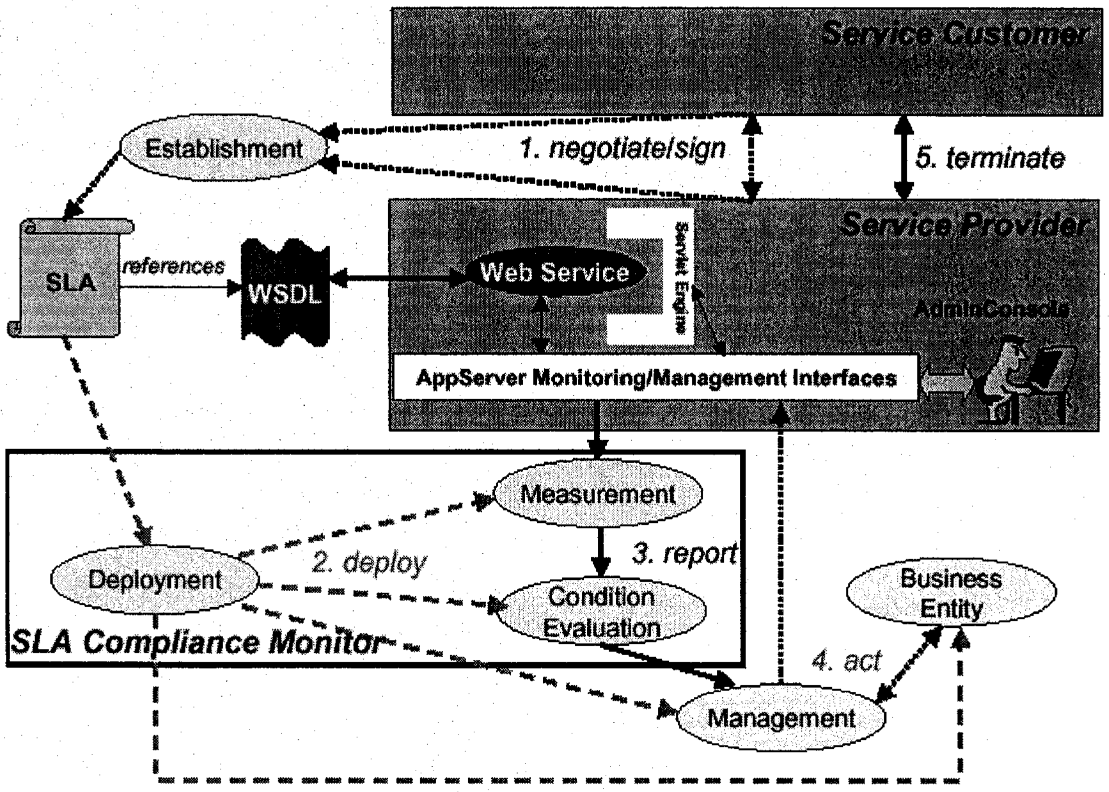


Figure III.7: WSLA Services and their interactions

III.2.3 Network identity management

The Liberty Alliance is the first open standards organization for federated network identity management and identity-based services [29]. Its specifications [38] intend to facilitate the interaction between businesses and their customer and partners, while respecting the privacy and the security of shared identity information.

Federated identity management makes it possible for an authenticated party to be recognized across multiple domains [38]. It allows users to authenticate only once to one company or web site, and access personalized content and services in an other location without having to re-authenticate to this location or sign-on with a separate username and password. The first web site that authenticates the user is called identity provider and the target location is called service provider.

The Liberty Alliance architecture is as follow:

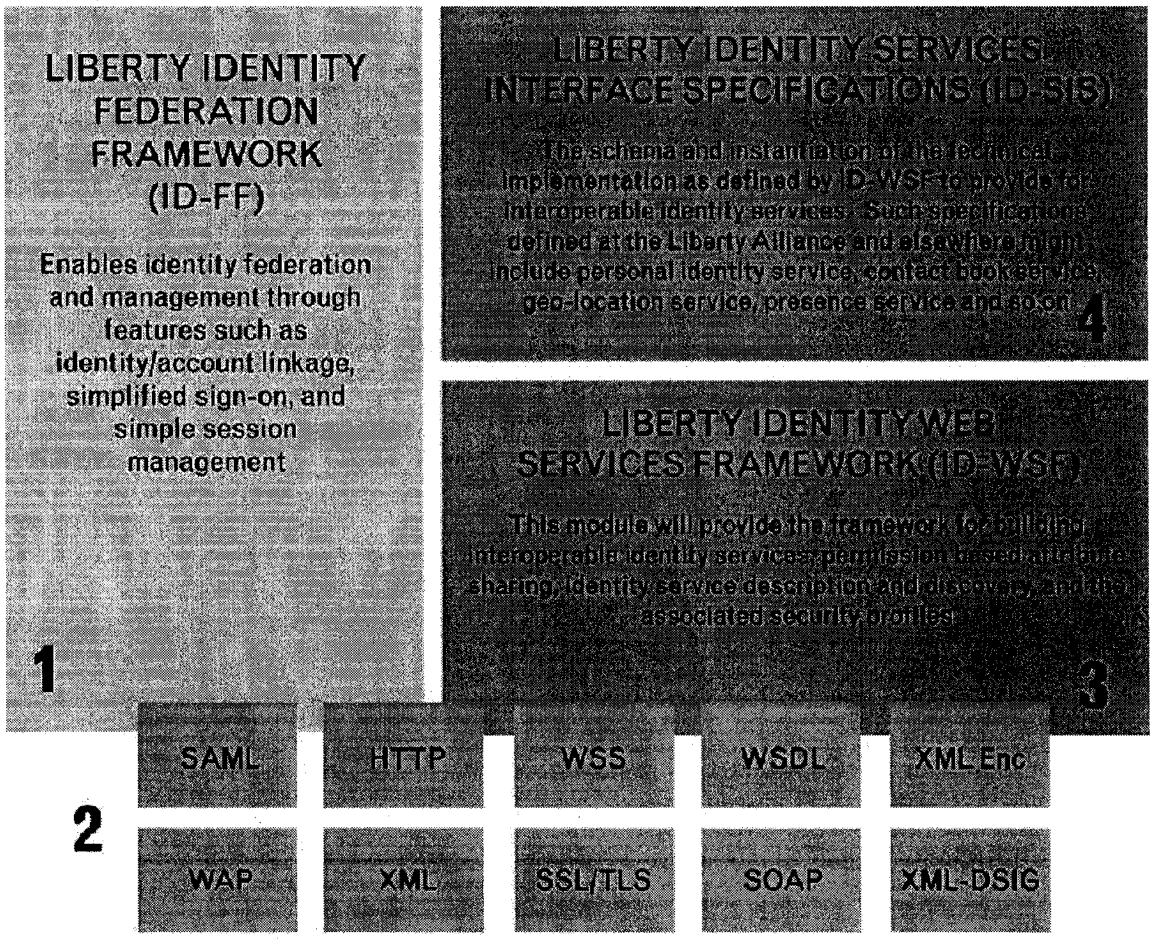

LIBERTY IDENTITY FEDERATION FRAMEWORK (ID-FF)
Enables identity federation and management through features such as identity/account linkage, simplified sign-on, and simple session management
1
LIBERTY IDENTITY SERVICES INTERFACE SPECIFICATIONS (ID-SIS)
The schema and instantiation of the technical implementation as defined by ID-WSF to provide for interoperable identity services. Such specifications defined at the Liberty Alliance and elsewhere might include personal identity service, contact book service, geo-location service, presence service and so on
4
LIBERTY IDENTITY WEB SERVICES FRAMEWORK (ID-WSF)
This module will provide the framework for building interoperable identity services, permission based attribute sharing, identity service description and discovery, and the associated security profiles
3
2
SAML
HTTP
WSS
WSDL
XML Enc
WAP
XML
SSL/TLS
SOAP
XML-DSIG

Figure III.8: High-Level Overview of the Liberty Alliance Architecture

Within this architecture [figure III.8], the federated identity management may be provided using account linkage, simplified sign-on or fundamental session management. This allows users with multiple accounts at different Liberty enabled sites to link these accounts for future authentication and sign-on. The simplified sign-on allows users to sign-on once at a Liberty-enabled site, and to be seamlessly singed-on when navigating to another Liberty-enabled site. This is supported both within a circle of trust[2] and across circles of trust. The Liberty Federation Framework (ID-FF) enables also the real-time exchange of Meta Data, such as X.509 certificates and service endpoints, between Liberty-compliant entities.

The Fundamental session management features enable global sign-out from all Liberty-enabled sites that are linked together in a given session, and also enables the associated entities to communicate the type of required authentication when signing-on.

The Liberty Identity Web Services Framework (ID-WSF) defines a framework for creating, discovering and consuming identity services[3]. It defines the fundamental features for the use of identity services, like identity services SOAP binding (how ID.* messages are mapped into the <body> element of SOAP messages) and security profiles.

----

(2) The group of service providers that share linked identities and have business agreements in place is known as a circle of trust

(3) " An identity service is a particular type of web services that acts upon some resource to either retrieve information about an identity, update information about an identity, or perform some action for the benefit of some identity" [44].

The Liberty Identity Services Interfaces Specifications (ID-SIS) [38] are a set of specifications for identity services that can be built. These services might include registration, contact book, calendar and geo-location services. These specifications are used to ensure the interoperability among services. For instance, the personal profile identity service defines a basic schema for user profile. It specifies the most commonly used attributes and their possible values, including name, legal identity contact information such as home and work addresses, phone number, email and other online information. It also includes mechanisms for employment and public key information inclusion, and service extension to include other arbitrary data [39].

Some protocols have been defined for communication between the three actors in the Liberty architecture (principal, identity provider, service provider) including protocols for identity federation, single-sign-on, federation termination, single logout and others [40]. Bindings and profiles of these protocols and Liberty messages to HTTP-based communication frameworks are also specified [41]. A framework for describing and discovering identity management is also specified [42].

III.2.4 Analysis

There are some standards and solutions (or implementations) in the web services domain that relate to the identified requirements. There are solutions and/or standards for security, for SLA management [37], for network identity management [29] and others. However, these solutions and standards tackle just specific requirements and do not provide a comprehensive and integrated solution. The following table gives a summary of the requirements that are satisfied by each solution and each standard.

Standards & Solutions	**Satisfied Requirements**
Web Services Security (WS-Security)	Security (standard)
Security Assertion Markup Language (SAML)	Security (standard)
Web Services Enhancements (WSE)	Security (solution)
Web Services Domain Boundary Controller (WS-DBC)	Security (solution)
WSLA Framework	SLA management (solution)
Liberty Alliance Project	Network identity management (standard)

Table III.2: Web services standards and solutions analysis

III.3 Summary

The Parlay/OSA framework provides common functions for accessing telecommunications networks capabilities. However, it comes as a separate functions library and its use is not transparent to the application developers. Parlay X [20] is a more recent initiative that focuses on Web services. However, the specifications do not include anything on common functions (e.g. security, SLA).

On the other hand, there are standards for some of the common functions of web services. Examples are the Liberty Alliance Project specification for federated identity management, WS-Security and SAML for security management and P3P [30] for privacy management. There are also solutions that meet some of the identified requirements (e.g.

WSLA framework of IBM for web services SLA management, WS-DBC of Xtradyne and web services enhancements of Microsoft for security issues). However, these standards and solutions tackle just specific requirements and do not provide a comprehensive and integrated solution.

The solution (named framework) we are proposing is based on web services paradigm. It uses existing web services standards and technologies to provide common functions to the applications developers(*). The architecture of this solution is presented in the next chapter.

(*) A paper concerning this solution was published on June 2004 [45].

Chapter 4:

Proposed architecture

This chapter is about the framework we are proposing for the common functions. It presents the architecture of the framework, its functional entities and its execution. It starts by stating some assumptions made for the proposed architecture. Then, it presents a general new web services architecture. After that, the general framework usage is presented, and its different functional entities are described. The chapter ends by a description of the proposed framework and its functional entities interactions.

IV.1 General assumptions

Depending on the business model, the network operator may choose to implement the common functions within each web service, use its own framework or use a framework provided by a third party. This leads to the following assumptions for the architecture:

- The network operator may own the framework or have a business agreement with the framework operator.
- The network operator may delegate no function, some or all of the required functions to the framework. For instance, the network operator may prefer to handle the security issues itself, and delegates all the other common functions execution to the framework.

- The network operator may have its own policy server that maintains information about the functions to be delegated to the framework. It may also use the framework policy server for that.

IV.2 New web services architecture

To give a solution to the common functions, we have defined a new web services architecture. This architecture allows the use of the common functions framework by multiple web services simultaneously and easily. This section starts by explaining the need for a new architecture. Then, it presents the new architecture and discusses some issues related to its usage for providing common functions.

IV.2.1 Motivations

The architecture we are proposing for providing common functions is based on the web services paradigm. In fact, this architecture is intended for a web services environment, then, the use of a different technology in this case will generate some heterogeneity problems. The second reason of this choice relates to the fundamental principals of web services. They provide a high level of abstraction, they are loosely coupled and they are programming language independent. This will make the framework easy to use and makes its components reusable.

In the current web services architecture, web services execution is done by directly interacting with services implementations. Executing a web service requires the client application to bind to the service implementation and communicate directly with it. To use the framework for providing common functions, web services architecture must allow the service execution to go through an intermediary system. This is not enabled by the current web services architecture.

IV.2.2 New architecture

Instead of the traditional web services architecture (figure IV.1a), we have defined a new framework based architecture (figure IV.1b). The framework in figure IV.1b includes and extends the service registry functionalities. Furthemore, it implements new common and additional functal requirements identified eurlierin the second chapter.

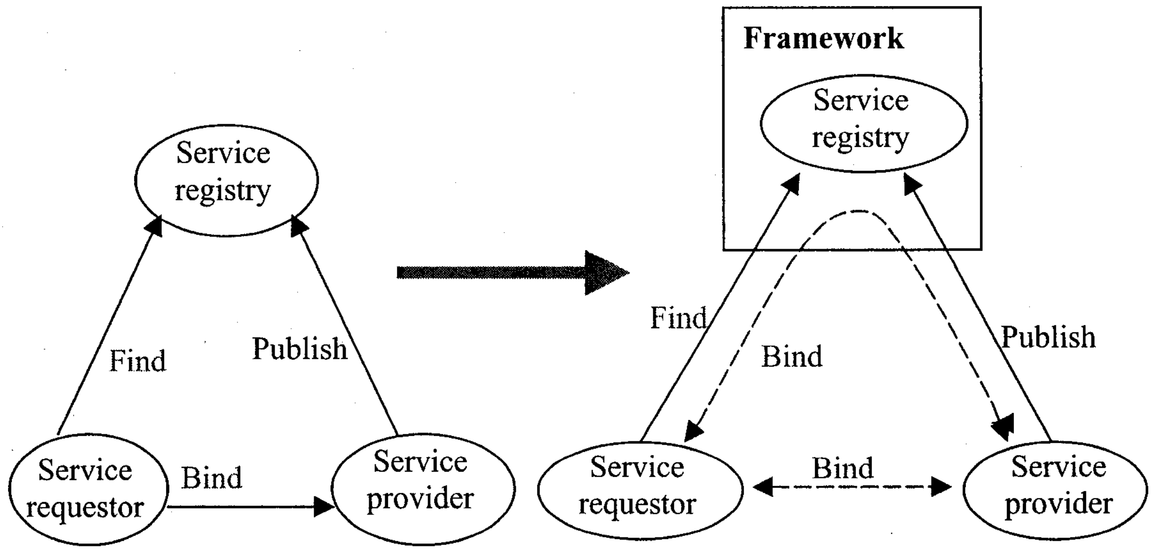


Figure IV.1a: Traditional web services architecture

Figure IV.1b: New web services architecture

IV.2.3 Delegation issues

In the context of this thesis, the web service provider is the network operator. Then, in the new architecture, three execution scenarios are possible, depending on the common functions delegated to the framework. The service provider can delegate no function, some or all of the common functions to the framework. These scenarios are presented here-after and the relating use cases and sequence diagrams are presented using the UML (Unified Modeling Language) notation.

IV.2.3.1 No function is delegated to the framework

In this case, it is better to have a direct communication between the service requestor and the service provider, since no framework function is needed. However, if the framework is used, it plays a simple gateway role. It receives a message from its source and sends it to the destination without any treatments or transformations.

a. Use case

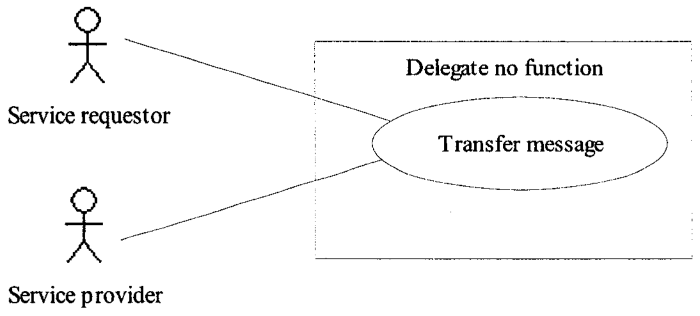


Figure IV.2: No functions is delegated to the framework use case

b. Sequence diagram

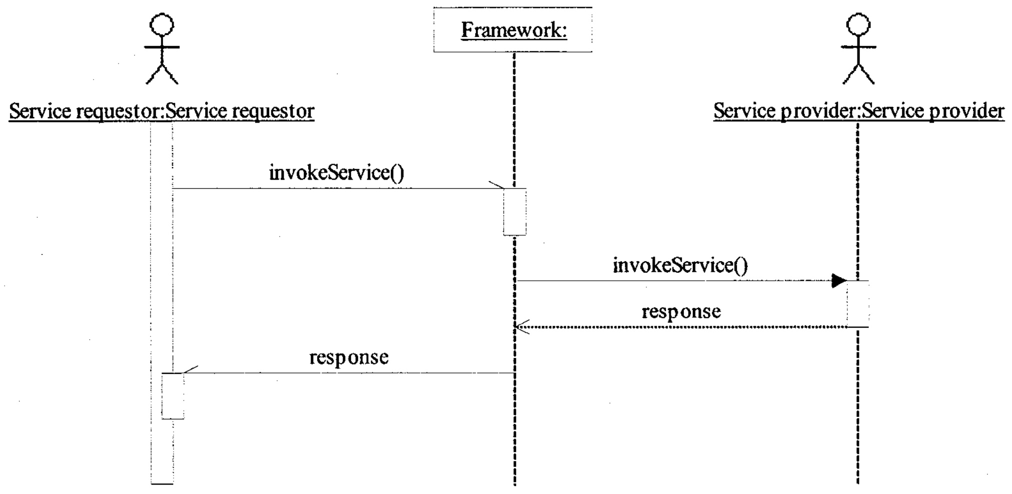

Framework:
Service requestor:Service requestor
Service provider:Service provider
invokeService()
invokeService()
response
response

Figure IV.3: No functions is delegated to the framework sequence diagram

IV.2.3.2 All functions are delegated to the framework

In this case, the framework has to execute all required functions, including security, service management, network identity management, privacy management and charging. The following use case illustrates the overall functions to be performed by the framework.

a. Use case

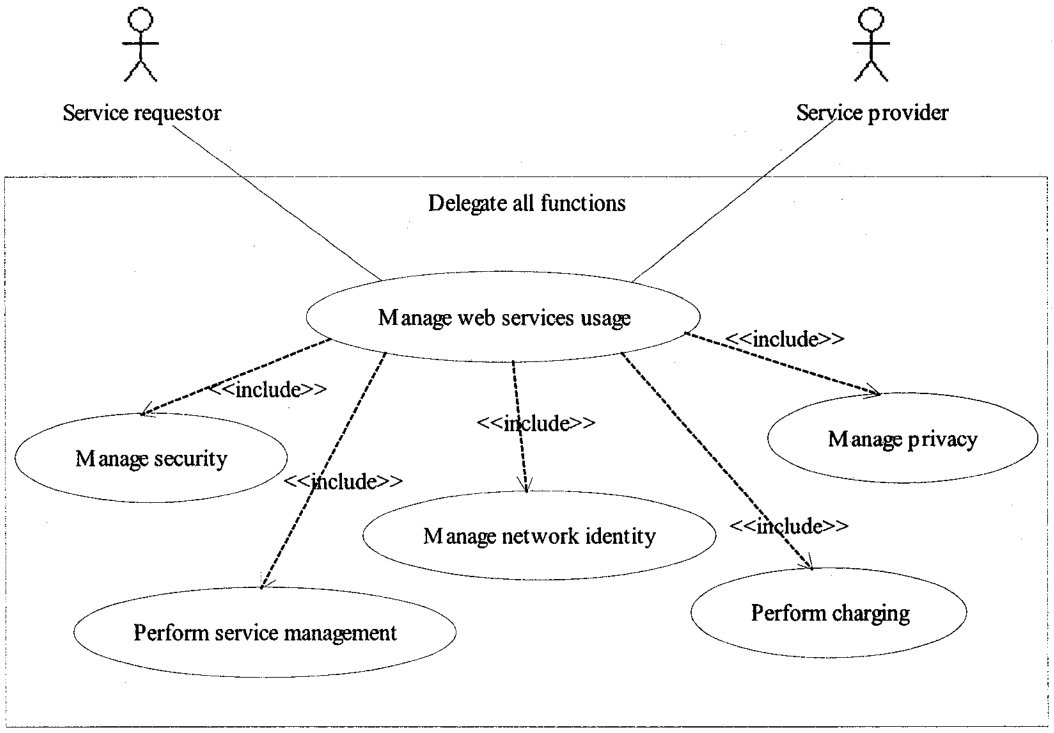


Figure IV.4: All functions are delegated to the framework use case

b. Sequence Diagram

Scenario:

1- The service requestor invokes a service.

2- The framework processes the request by executing all the adequate functions.

3- The framework transfers the processing outcome message to the service provider.

4- The service provider sends the response to the framework.

5- The framework processes the response the same way as the request

6- The framework sends the processed response to the service requestor.

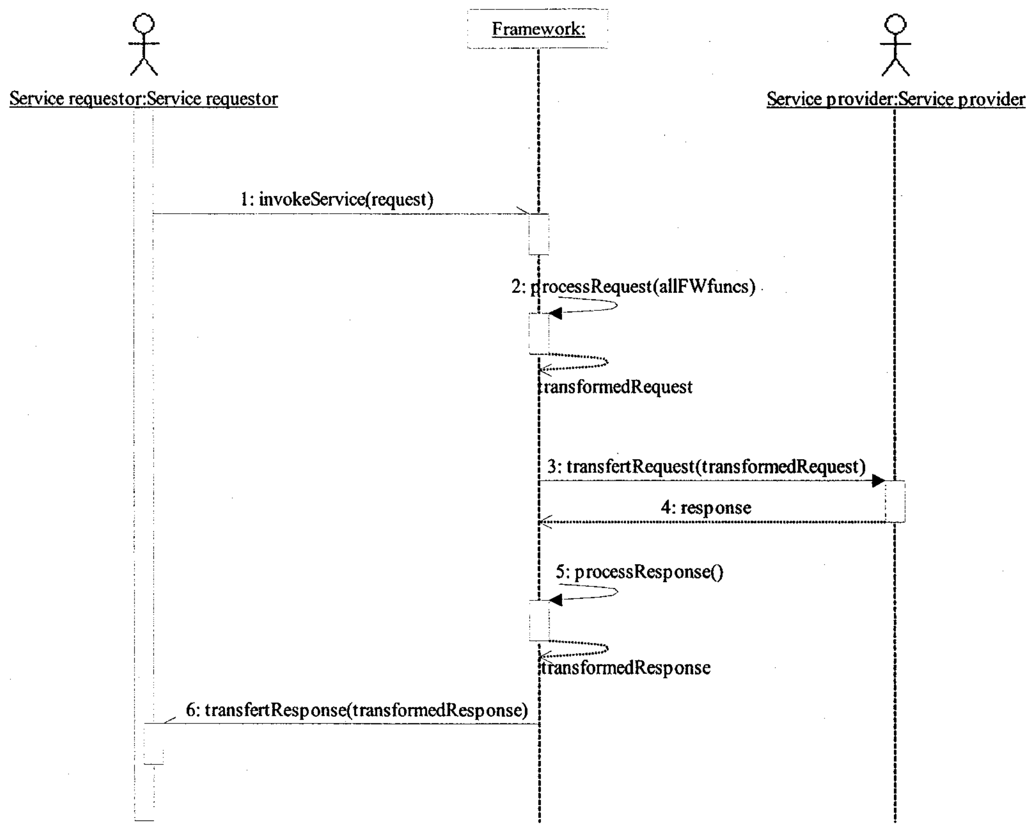

Framework:
Service requestor:Service requestor
Service provider:Service provider
1: invokeService(request)
2: processRequest(allFWfuncs)
transformedRequest
3: transfertRequest(transformedRequest)
4: response
5: processResponse()
transformedResponse
6: transfertResponse(transformedResponse)

Figure IV.5: All functions are delegated to the framework sequence diagram

IV.2.3.3 Some functions are delegated to the framework

In this case, a policy server is used to determine which functions are delegated to the framework.

a. Use case

The request processing includes a preliminary step before executing the delegated common functions. When receiving a service invocation request, the framework starts by interrogating the policy server to get the list of the common functions that are delegated to the framework.

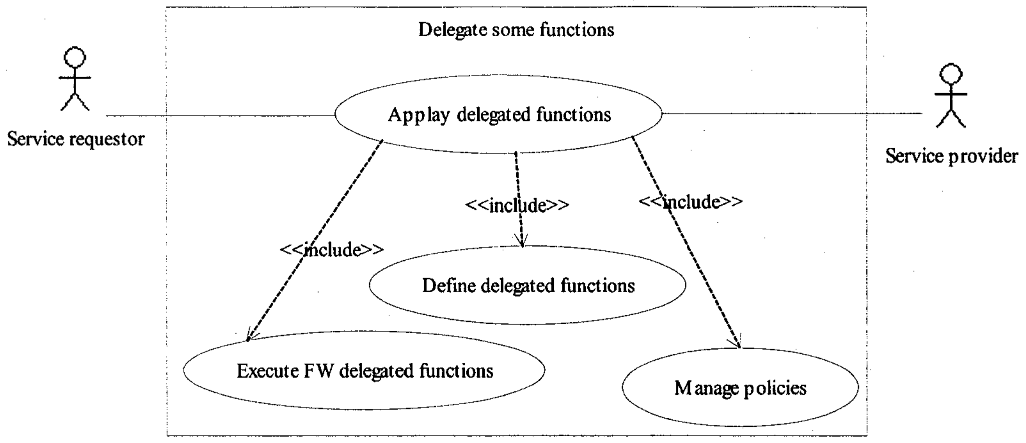


Figure IV.6: Some functions are delegated to the framework use case

b. Sequence Diagram

The main scenario for this use case is as follows (figure IV.7):

1- The service requestor invokes a service.

2- The framework identifies the set of delegated functions by consulting the policy server.

3- The framework processes the request by executing the delegated functions.

4- The framework transfers the processed request to the service provider and receives a response from it.

5- The framework processes the response and sends the processing results to the service requestor.

This scenario includes the two previous ones. In the first case, the set of delegated functions is empty; then, the actions "processRequest" and "processResponse" can be removed. In the second case, the “subSetDelegatedFunctions” includes all common functions.

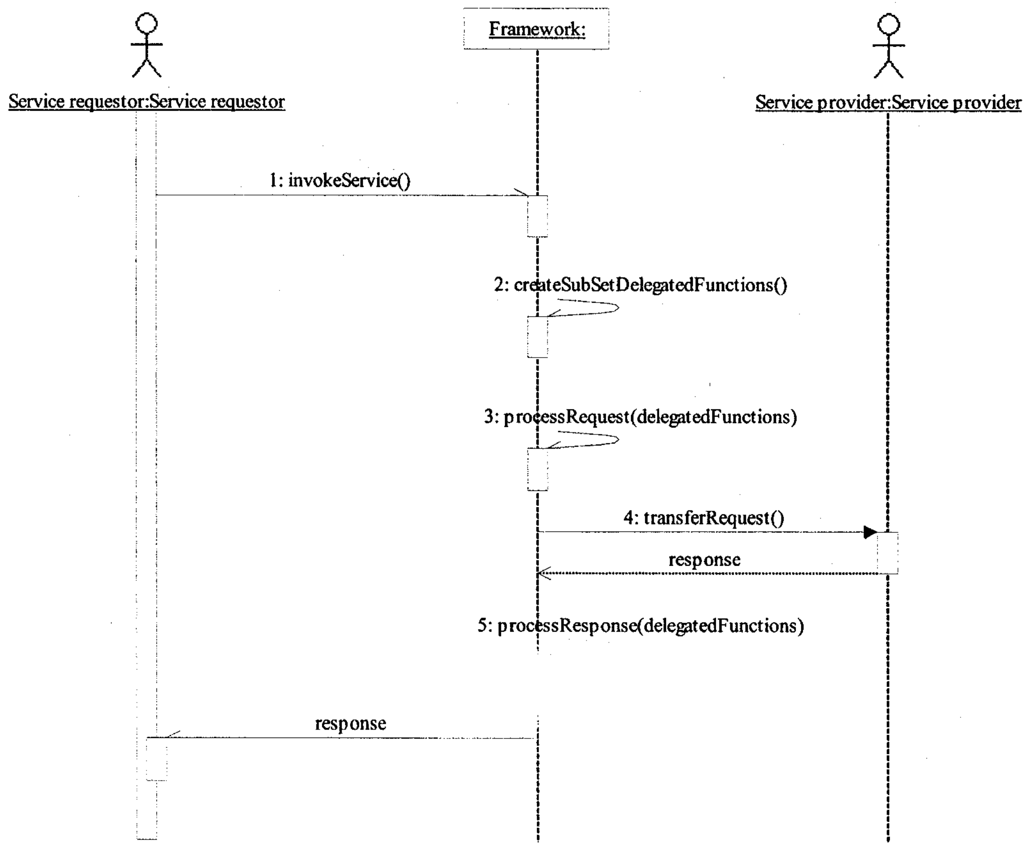

Framework:
Service requestor:Service requestor
Service provider:Service provider
1: invokeService()
2: createSubSetDelegatedFunctions()
3: processRequest(delegatedFunctions)
4: transferRequest()
response
5: processResponse(delegatedFunctions)
response

Figure IV.7: Some functions are delegated to the framework sequence diagram

IV.3 Framework functional entities

In the framework architecture we propose that each common function is provided by a different functional entity. The functional entities included in the framework are: Security server, network identity management server, SLA management server, charging server, policy server, registry server, privacy management server and the framework entry point that we call framework interface. This section gives the description of each of these entities. It presents each functional entity as an independent use case. The use cases and the sequence diagrams are presented using the UML notation.

IV.3.1 Security server

The framework may have to perform some or all of the security aspects. The detail of the security functions to be executed (i.e. authentication, authorization ...) and the related means (i.e. encryption key, signing certificate ...) are retrieved from the policy server.

Once a request is received, the framework starts by identifying the set of required security functions by consulting the policy server. Then, it executes them according to the associated information.

IV.3.1.1 General security use case

The security functions include authentication, authorization, data integrity, confidentiality, non-repudiation, availability and key management.

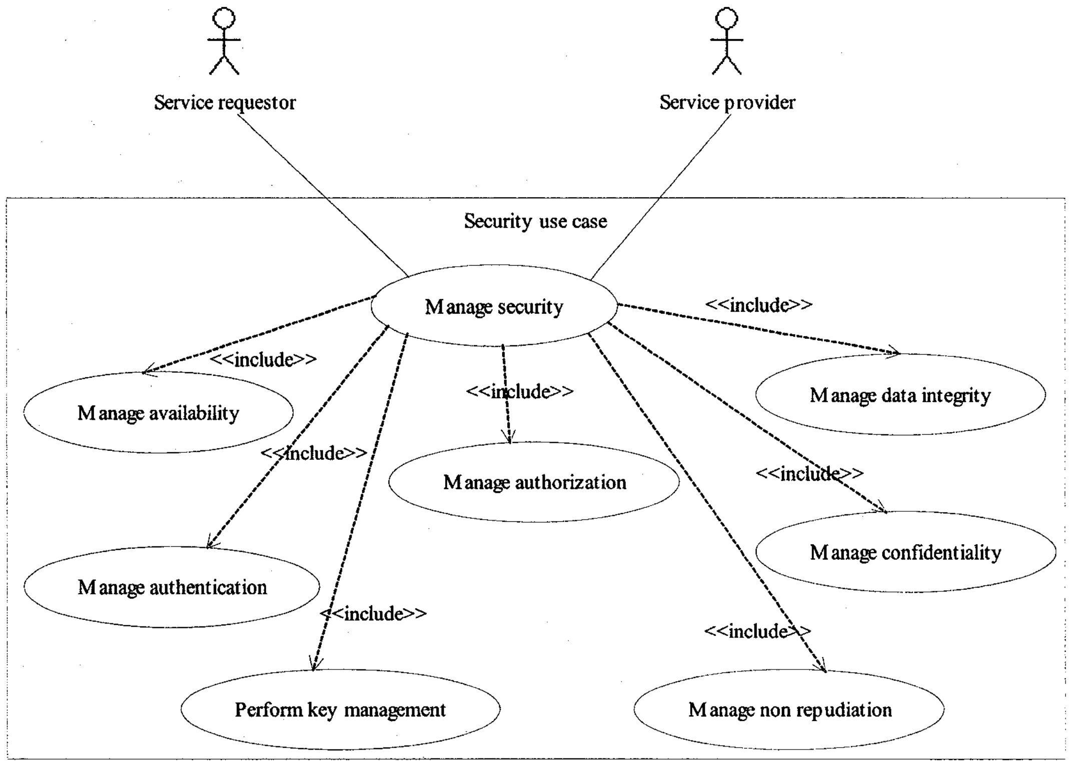


Figure IV.8: General security use case

IV.3.1.2 "Manage security" use case

The "manage security" use case is the core of the security process. It determines the security functions to be executed, performs adequate security treatments, and inserts necessary security assertion to the received message.

a. Use case

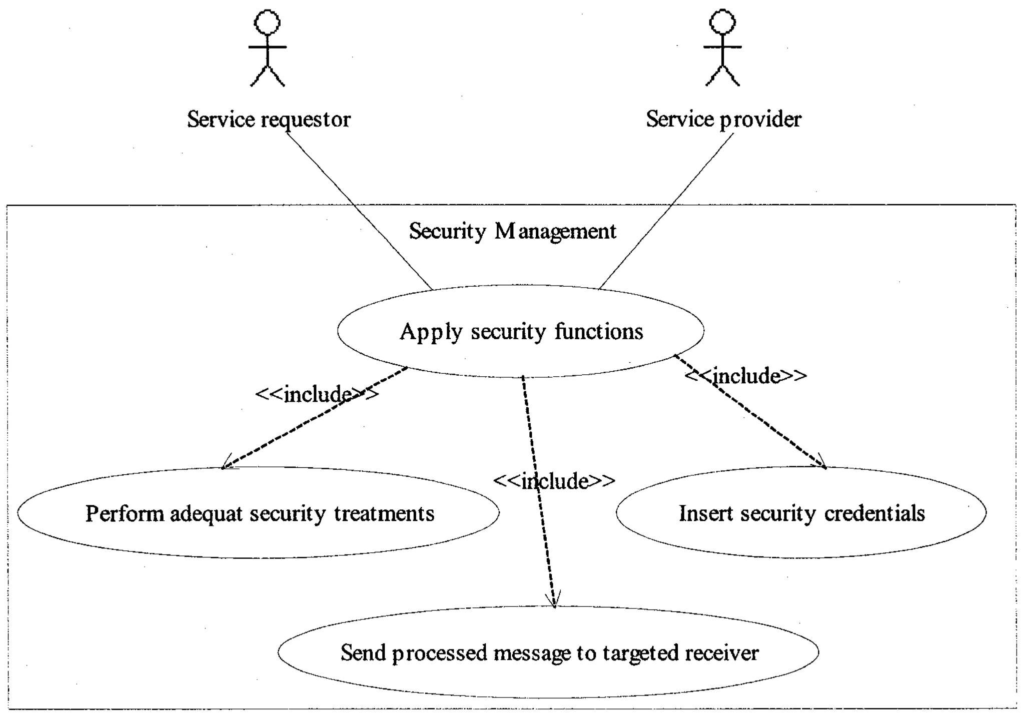


Figure IV.9: "Manage security" use case

b. Sequence diagram

1- The service requestor invokes a service by sending a SOAP request to the framework.
2- The framework processes the request (Authenticates the requestor, verifies resources authorization, digitally sign and/or encrypt the request, ...)
3- The framework inserts the appropriate security credentials into the SOAP request (such as authentication assertion, authorization assertion, ...)
4- The new request is transferred to the service provider.
5- The framework receives the service provider response.

6- The framework applies to the response the reverse of the treatment applied to the request.

7- The output response is sent to the requestor

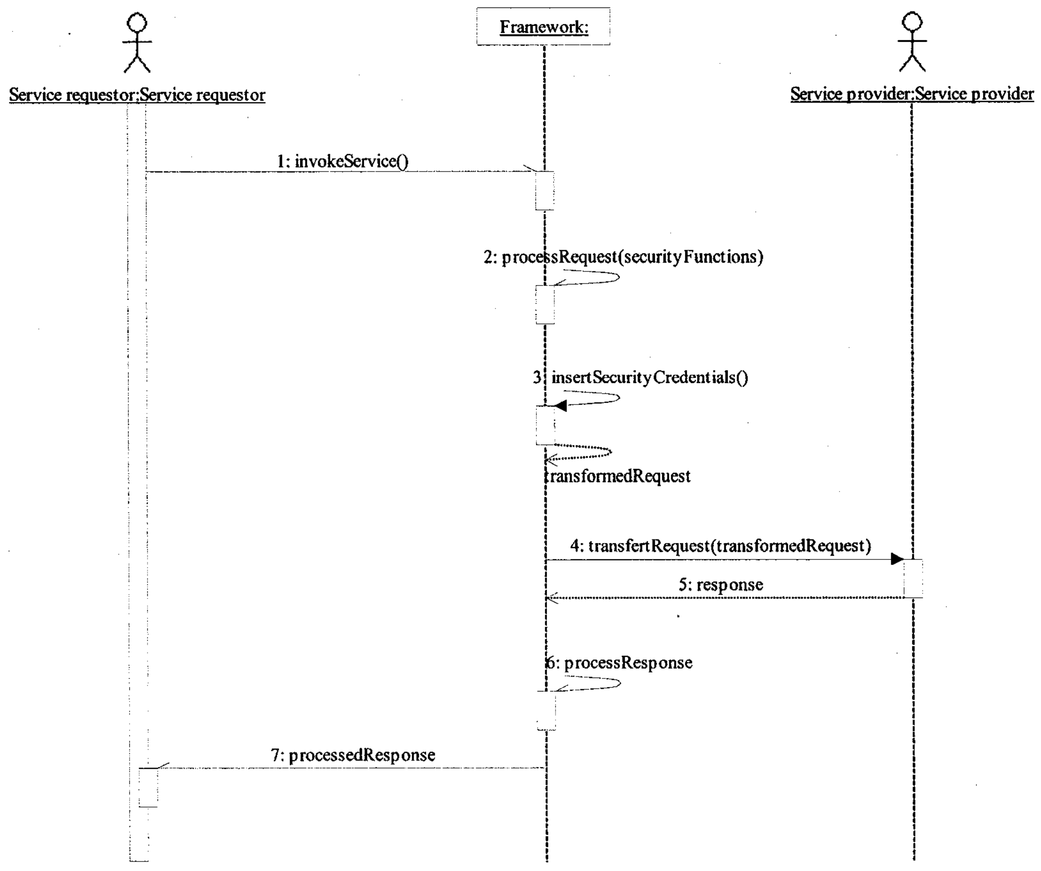

Framework:
Service requestor:Service requestor
Service provider:Service provider
1: invokeService()
2: processRequest(securityFunctions)
3: insertSecurityCredentials()
transformedRequest
4: transfertRequest(transformedRequest)
5: response
6: processResponse
7: processedResponse

Figure IV.10: "Manage security" sequence diagram

In order to ensure the plug-and-play requirement, the security service has to be implemented using the security proxy model. Furthermore, the security server must adhere to the WS-Security specification and uses SAML assertions to ensure message security.

IV.3.2 Network identity management server

The most appropriate mechanism for the network identity management is Single Sign On. So, when attempting to access a web service, the service requestor authenticates to the framework and the framework communicates the authentication assertion to the entity that hosts access to the service (the service provider).

This can be achieved by either inserting the entire authentication assertion into the request to transfer to the service provider or only by inserting its reference.

a. Use case

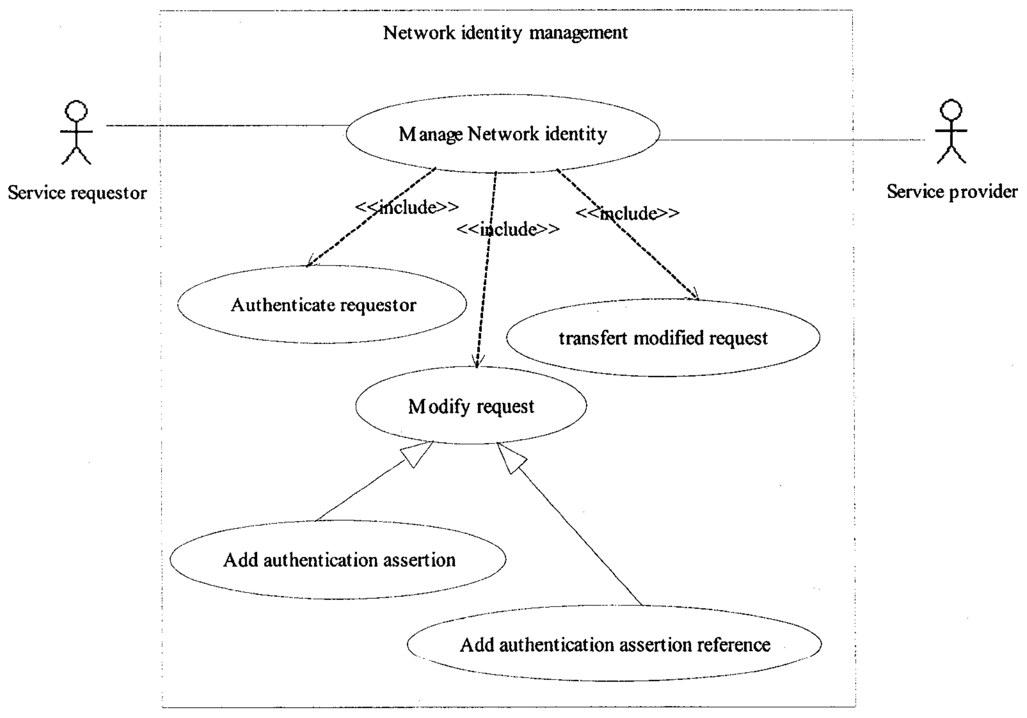


Figure IV.11: Network identity management use case

b. Sequence diagram

1- The service requestor invokes a service

2- The framework authenticates the service requestor

3- The framework adds the authentication assertion to the request (or possibly the authentication assertion reference).

4- The framework sends the request to the service provider, and receives the response.

5- The framework transfers the response to the service requestor.

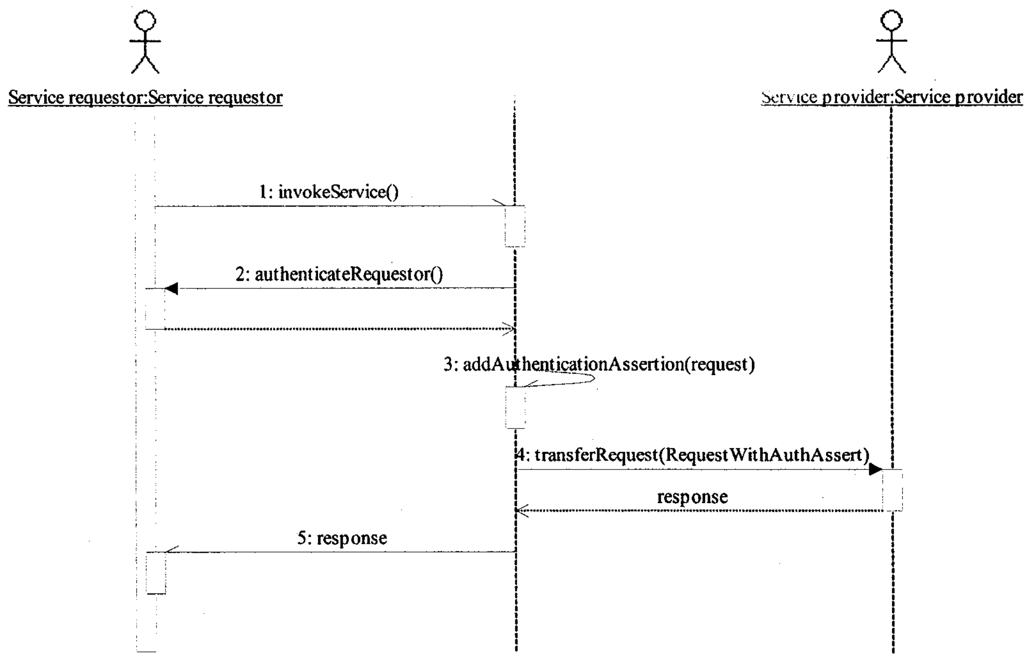

Service requestor:Service requestor
Service provider:Service provider
1: invokeService()
2: authenticateRequestor()
3: addAuthenticationAssertion(request)
4: transferRequest(RequestWithAuthAssert)
response
5: response

Figure IV.12: Network identity management sequence diagram

IV.3.3 Service Level Agreement management server

To manage the service providing conditions and service quality levels, the framework must be in charge of the SLA parameters negotiations, and SLA creation. It has also to enable the web service provider and the consuming application to commit to the created contract.

The committing function may be delegated to the security server since it implies some security functions such as signing, integrity and confidentiality insurance of the SLA. Other security considerations apply to the SLA commitment including the non repudiation that requires key management. The framework may save the SLA violations for future use.

Once the SLA is negotiated, created and committed to, the framework must have the ability to provision the agreed SLA somewhere in the system, so that it can be evaluated during the service execution. This aims to enable the SLA enforcement.

a. Use case

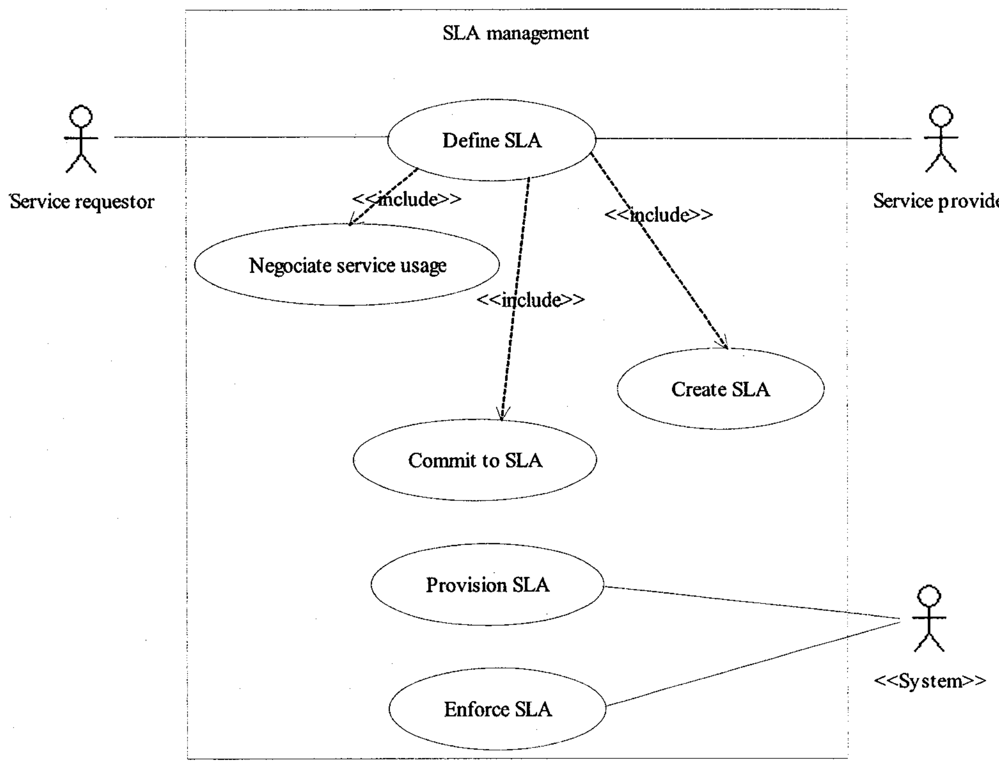


Figure IV.13: Service Level Agreement use case

b. Sequence diagram

1- The service requestor invokes a service

2- The framework negotiates the SLA parameters with the service requestor (the service capabilities to be supplied and under which condition the service is supplied)

3- The framework creates the SLA

4- The service requestor and the service provider commit to the created SLA (by digitally signing and exchanging the signed SLA for instance)

5- The framework provision the SLA

6- During the execution, the framework verifies if the web service is executed according to the associated SLA.

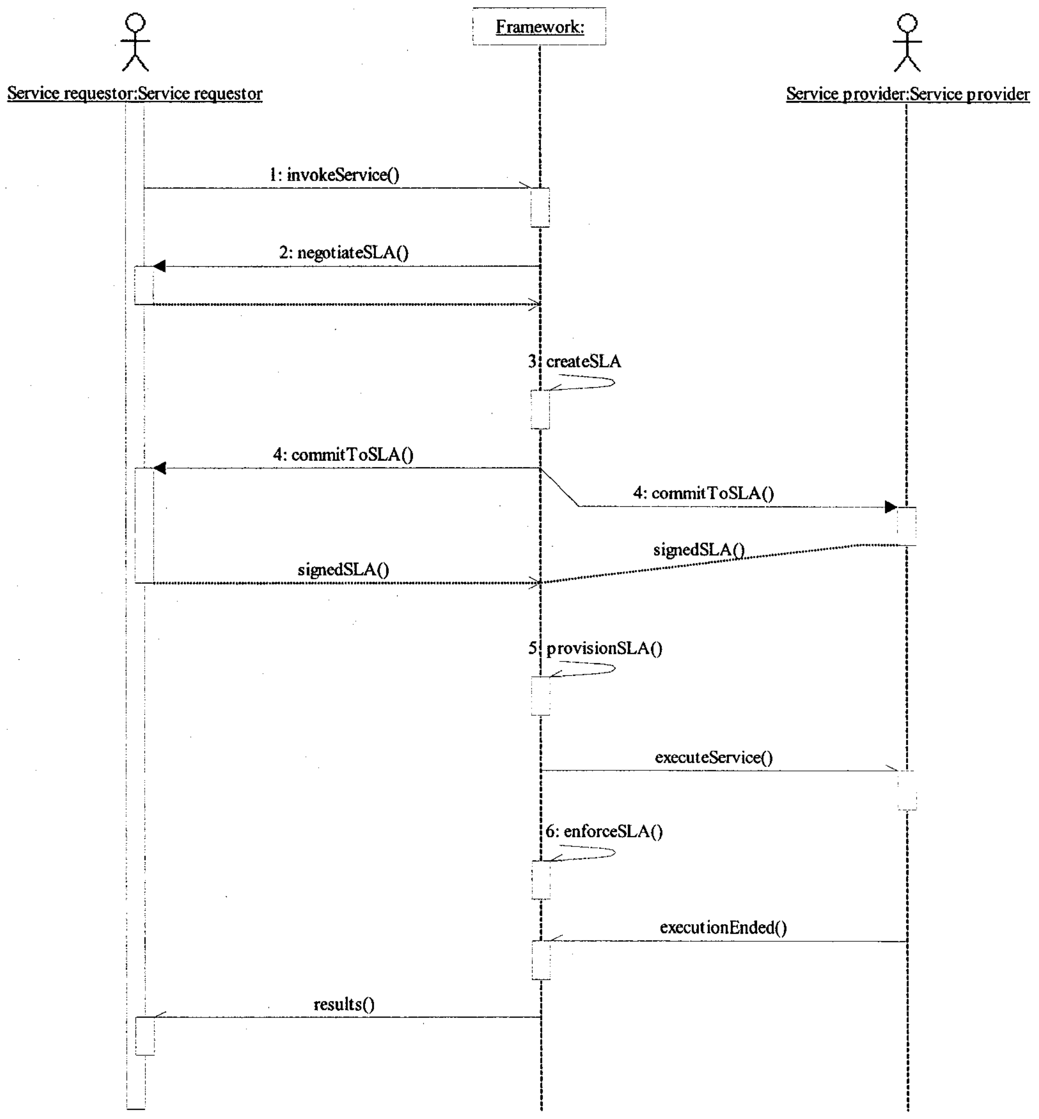


Figure IV.14: Service Level Agreement sequence diagram

IV.3.4 Charging server

In the case of a non-free service, the framework has to be able to charge the service requestor for the service usage. It must provide the service provider with a mechanism to enable service usage metering and accounting.

a. Use Case

When receiving a service invocation request, the framework negotiate the charging contract with the requestor (pre-paid or post-paid charging, service usage or content based charging, ...) and initiates the charging session. After the service execution, the framework terminates the charging session and saves the charging information. This information may be used to compute corresponding charges.

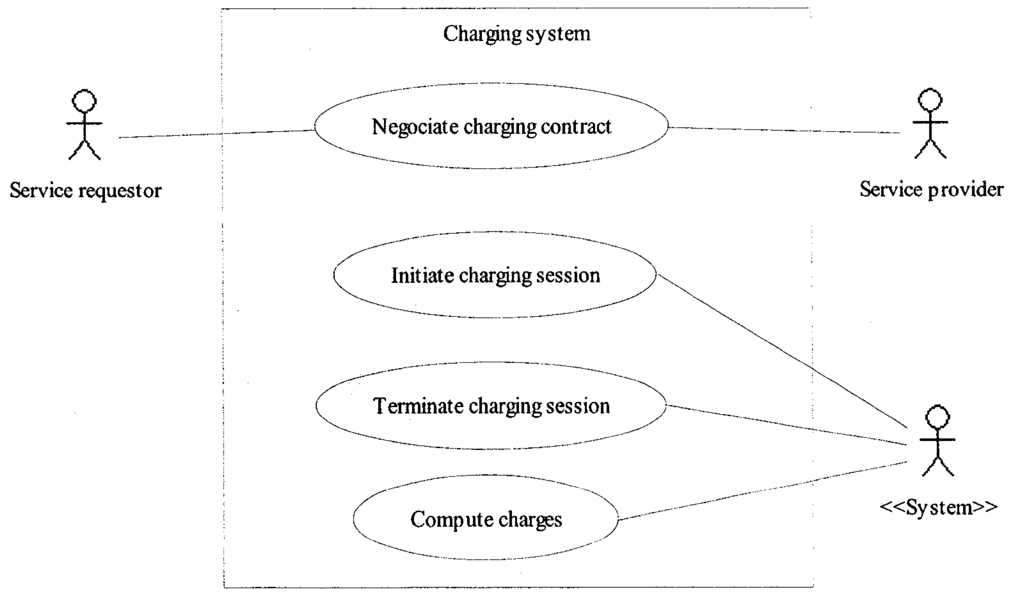

Charging system
Negociate charging contract
Service requestor
Service provider
Initiate charging session
Terminate charging session
Compute charges
<<System>>

Figure IV.15: Charging use case

b. Sequence diagram

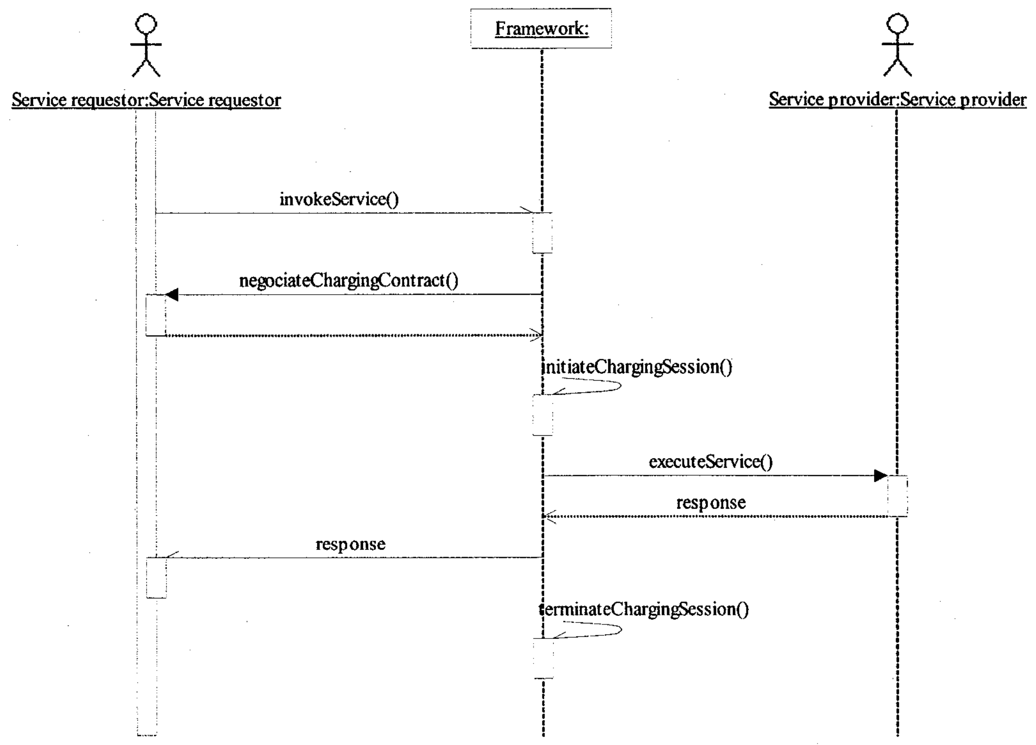

Framework:
Service requestor:Service requestor
Service provider:Service provider
invokeService()
negociateChargingContract()
initiateChargingSession()
executeService()
response
response
terminateChargingSession()

Figure IV.16: Charging sequence diagram

IV.3.5 Policy server

The policies governing the execution of the framework functions are captured in policy server. To identify the appropriate policies, the framework can either use the local policy server, or the service provider policy server (figure IV.17). A third party policy server can also be used. In the first case (IV.17a), the service provider must have a complete access to the framework policy server, in order to maintain policy information accurate and up to date. In the second case (IV.17b), the framework must have read access to the service provider policy server to get necessary information. In this case, the local policy server holds only the address of the remote server.

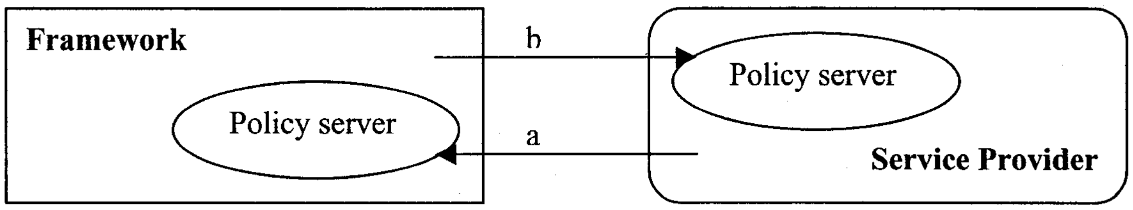


Figure IV.17: Open policy access interface

The policy server holds information about:

- The policy server to use.
- The common functions to be performed by the framework and their execution order.
- The way to perform each function and the detail of the related means (e.g. authentication policies, signing certificate, encryption key, authorization policies, ...)
- Service registration and discovery policies if these are not provided by the registry server.

IV.3.6 Registry server

For service registration, publication and discovery, the framework can use an existing UDDI implementation. The registry server database can be used to save other information such as SLA violations ones.

a. Registration scenario

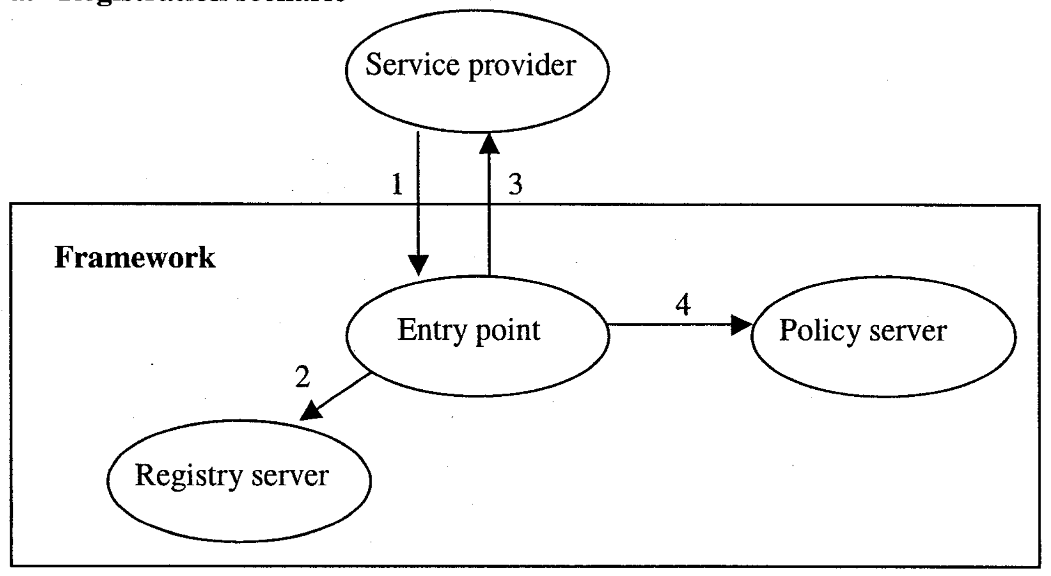


Figure IV.18: Service registration scenario

1- The service requestor creates a new web service and asks the framework to register and publish it.

2- The registry server makes the new service accessible for client applications.

3- The entry point asks for new service usage policies (policies may also be given and registered out of bound).

4- The entry point registers the policies or the service provider policy server reference if the latter is to be used.

b. Discovery scenario

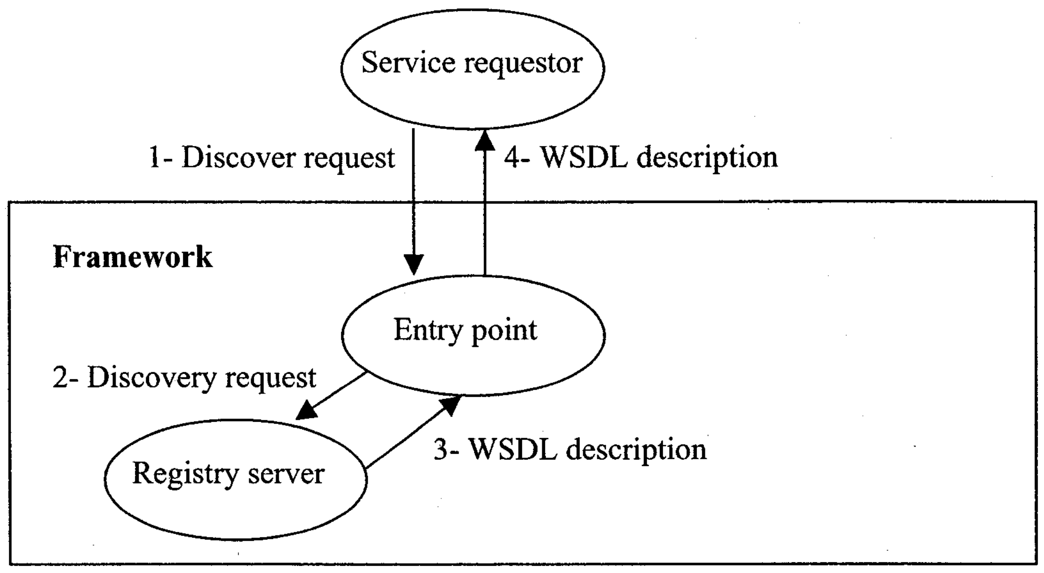


Figure IV.19: Service discovery scenario

For registration and discovery, the registry server can contact the policy server for authorization policies, if these are not provided by the registry server.

IV.3.7 Privacy management server

The privacy management server has the responsibility of protecting the service requestor's privacy. The privacy policies can be saved in the policy server or directly in the privacy server. These policies can be expressed using the Privacy Preferences Project (P3P) [30] of W3C.

The P3P specification enables the expression of the privacy practices in a standard format [30]. It also provides mechanisms for automatic retrieval and easy interpretation of these practices by user agents.

IV.3.8 Framework interface

The framework interface is the entry point to the framework. It federates the other functional entities execution and communication. It is the unique interface that may be seen and accessed from outside the framework.

IV.4 Framework Architecture

After describing the different framework components, we will see how these components are integrated together. Before that, we will have a look at the way a web service can be executed using the framework.

IV.4.1 Service execution scenario

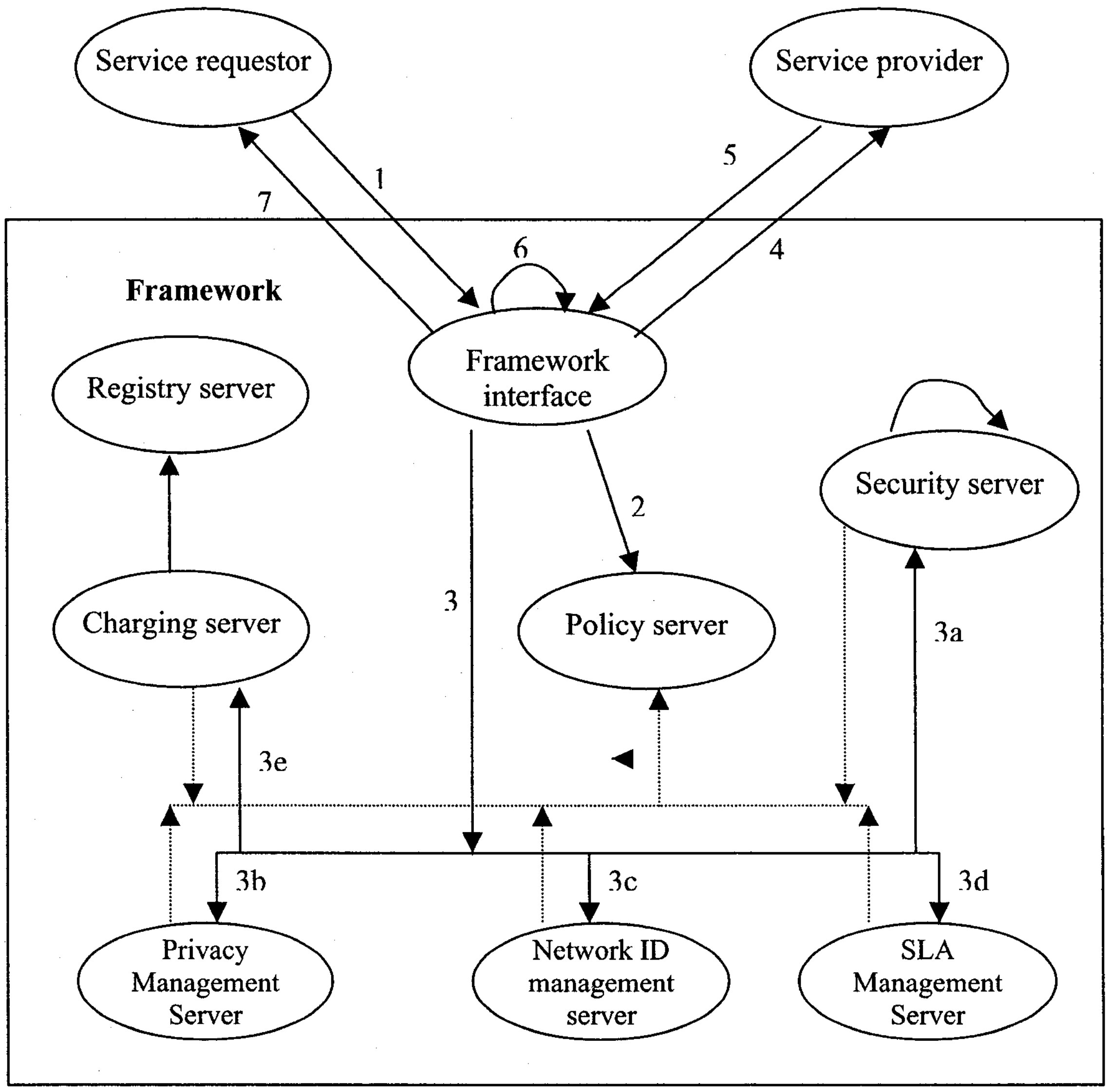


Figure IV.20: Framework execution scenario

The typical execution scenario of the framework when a web service is invoked is as follow:

1- The service requestor invokes a service by sending a SOAP request message to the framework.

2- The framework contacts the policy server to find out the functions that are delegated to the framework and their execution order.

3- The framework executes the required functions (3a and/or 3b and/or 3c and/or 3d and/or 3e) according to the order specified in the policy server and modifies the SOAP request accordingly (i.e. add adequate security assertions to the message). Some functions are only initiated, like SLA enforcement and charging, and terminated when appropriate during the service execution.

   Each functional entity contacts the policy server to identify how to perform its actions. For instance, the security server contacts the policy server to determine which security functions are to be performed (e.g. authentication, authorization) and via which means (i.e. authentication policies, authorization policies, signing certificate, encryption key …).

4- The framework transfers the processed request to the service provider.

5- The service provider sends the response to the framework

6- The framework processes the response according to the delegated functions.

7- The framework transmits the results to the requestor.

IV.4.2 Architecture

The framework architecture is shown in the figure below (figure IV.21). Each common function is implemented as a web service. An internal registry is used for these Web services. They can be accessed only by the framework's components, except the framework interface which can be accessed by external components.

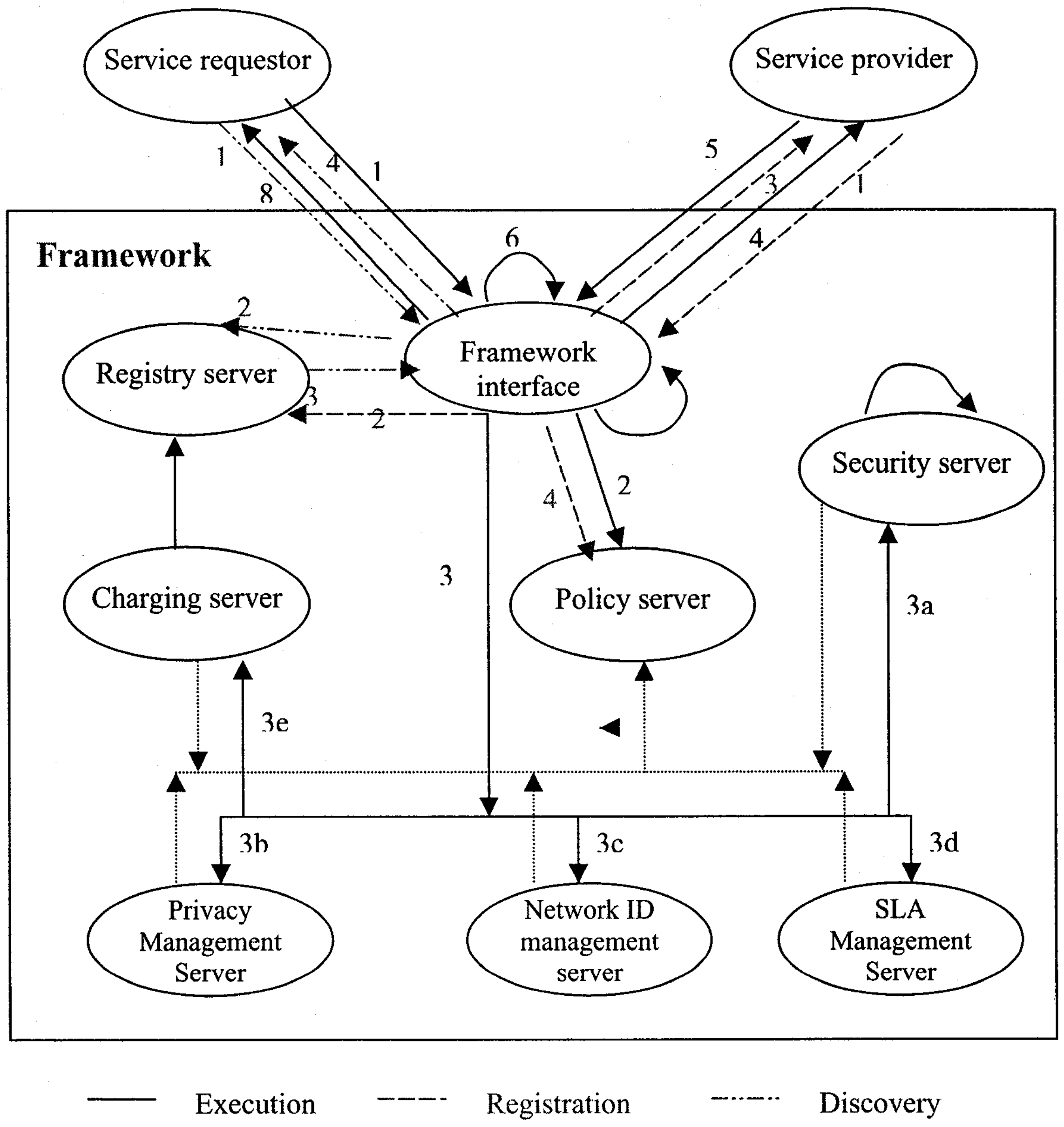


Figure IV.21: Framework architecture

IV.5 Framework functional entities interaction

The functional entities inter-communication is based on functions calls. Functions parameters depend on the function to be executed. The basic exchanges during the web service execution are described hereafter. Other parameters may be exchanged according to the use case.

- **Service requestor-Framework interface:** The service requestor invocation request provides the ***serviceId***, the ***methodId*** and the ***providerId*** parameters. The ***requestorId*** parameter can also be deduced from the request source. The *methodId* parameter refers to the web service method to be executed. The *providerId* parameter refers to the entity providing the service identified by the *serviceId* parameter.

- **Framework interface-Registry server:** Each time the framework interface needs a given functional entity reference, it is gotten to communicate with the registry server. It can use the *getFunctionalEntityRef(functionDescription)* function, where the ***functionDescription*** may be "security", "charging", "policy" or any other functional entity description.

- **Framework interface-Policy server:** The main parameter supplied by the framework interface when accessing the policy server is the ***serviceId***. The ***requestorId*** and **methodId** parameters may also be necessary if the policies associated to the web service depend on them. These three parameters are the principal ones provided by the other functional entities when communicating with the policy server.

- **Framework interface-Security server:** If only authentication is to be achieved by the security server, it's obvious that the required parameters are ***requestorId*** and ***serviceId***. The *serviceId* is used to determine the authentication policies (authentication certificate, authentication algorithm ...) associated to the invoked service. These policies may be related to the service provider rather than being related to a single web service. However, the same parameter is used, since a given web service is associated to a unique service provider.

In the case authorization function is also delegated to the framework, the ***methodId*** parameter is required as well. In other cases, where data integrity, data confidentiality and/or non-repudiation are to be ensured, the whole message (inbound or outbound message) is to be communicated to the security server.

- **Framework interface-SLA management server:** To set up the SLA management, the framework provides the SLA management entity with the ***serviceId*** and ***requestorId*** parameters.

- **Framework interface-Network ID management server:** The only needed parameter is the ***requestorId***.

- **Framework interface-Charging server:** ***requestorId*** and ***serviceId*** are communicated to the charging service to activate the charging session. Moreover, the ***execution ended message*** may have to be transferred to the same server, in order to terminate the charging session.

IV.6 Conclusion

This chapter gave a detailed description of the architecture of the framework we are proposing for common functions. It presented the global architecture of the framework and its functional entities. Then, it gave an insight to how these different entities interact between them.

The following chapter will explain how the framework execution can be optimized. It starts by looking at some use cases of using the framework in order to come up with a solution leveraging the most possible cases.

Chapter 5:

Optimization

This chapter is about the optimization of the common functions execution. It intends to give an answer to the question: *"What is the optimal way to execute the framework functional entities during a given client application execution"*. It starts with some particular case studies. After that, it outlines some useful remarks deduced from the studied cases. Then, it defines the common functions execution scope, and terminates by some general rules to monitor the framework execution.

V.1 Particular case studies

To help understanding the web services execution scenarios using the framework, we will start with looking at some case studies of using them.

V.1.1 First case study

Given a web service *ws* offering three functions f1, f2 and f3. The *ws* is provided by the web service provider *SP*. The *SP* delegates just the authentication function to the framework. A service requestor *SR* wants to execute f1 and f2. To model this problem, we will use the notation given here-after [Figure V.1]. This notation is used in the rest of this document to modelize this kind of problems.

$$\begin{tikzcd}
SR \arrow[r, "executes"] & ws(f1,f2) & SP \arrow[l, "provides"'] \arrow[r, "delegates"] & FW\{Auth\}
\end{tikzcd}$$

Figure V.1: First case study of web services execution

In this case, the best solution is to have a direct communication between the service provider and the service requestor, after the execution of the authentication function. The scenario of this case is as follows:

1. The SR asks for the execution of the function f1, on the web service *ws*.
2. The framework authenticates the SR
3. The framework redirects the request to the SP

After that, the SR continues the communication directly with the SP.

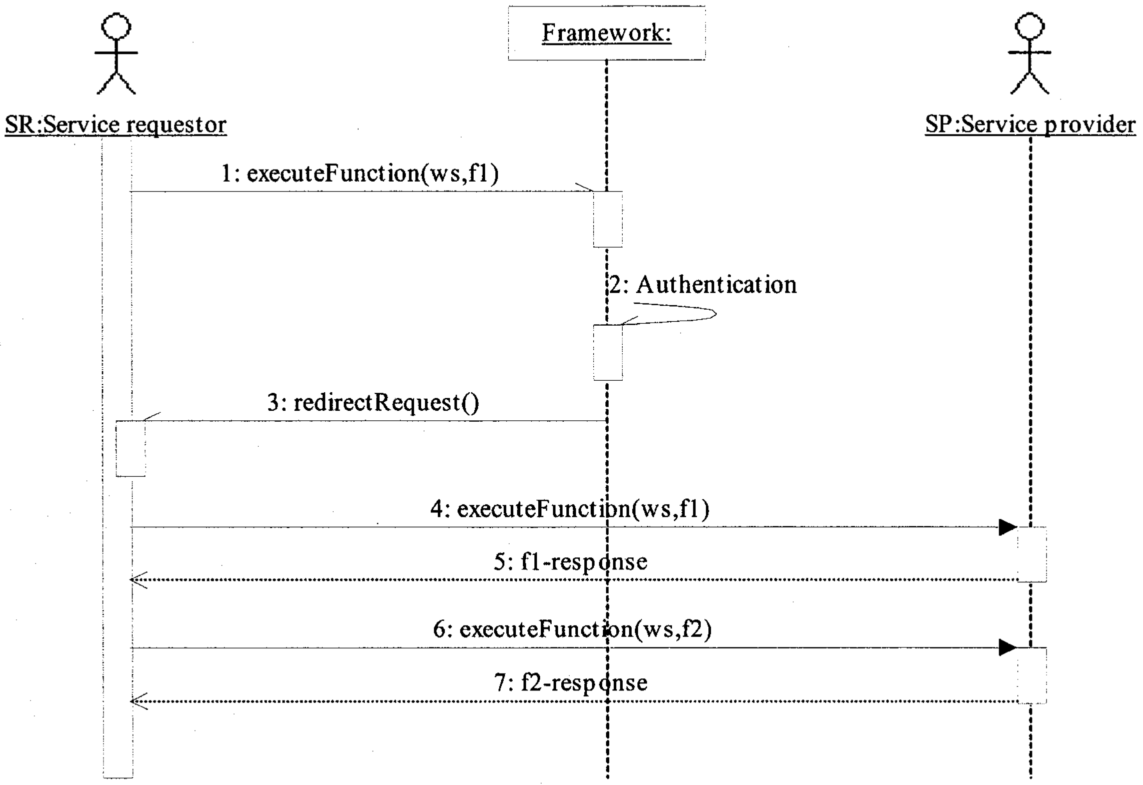


Figure V.2: Sequence diagram of the first case study of web services execution

V.1.2 Second case study

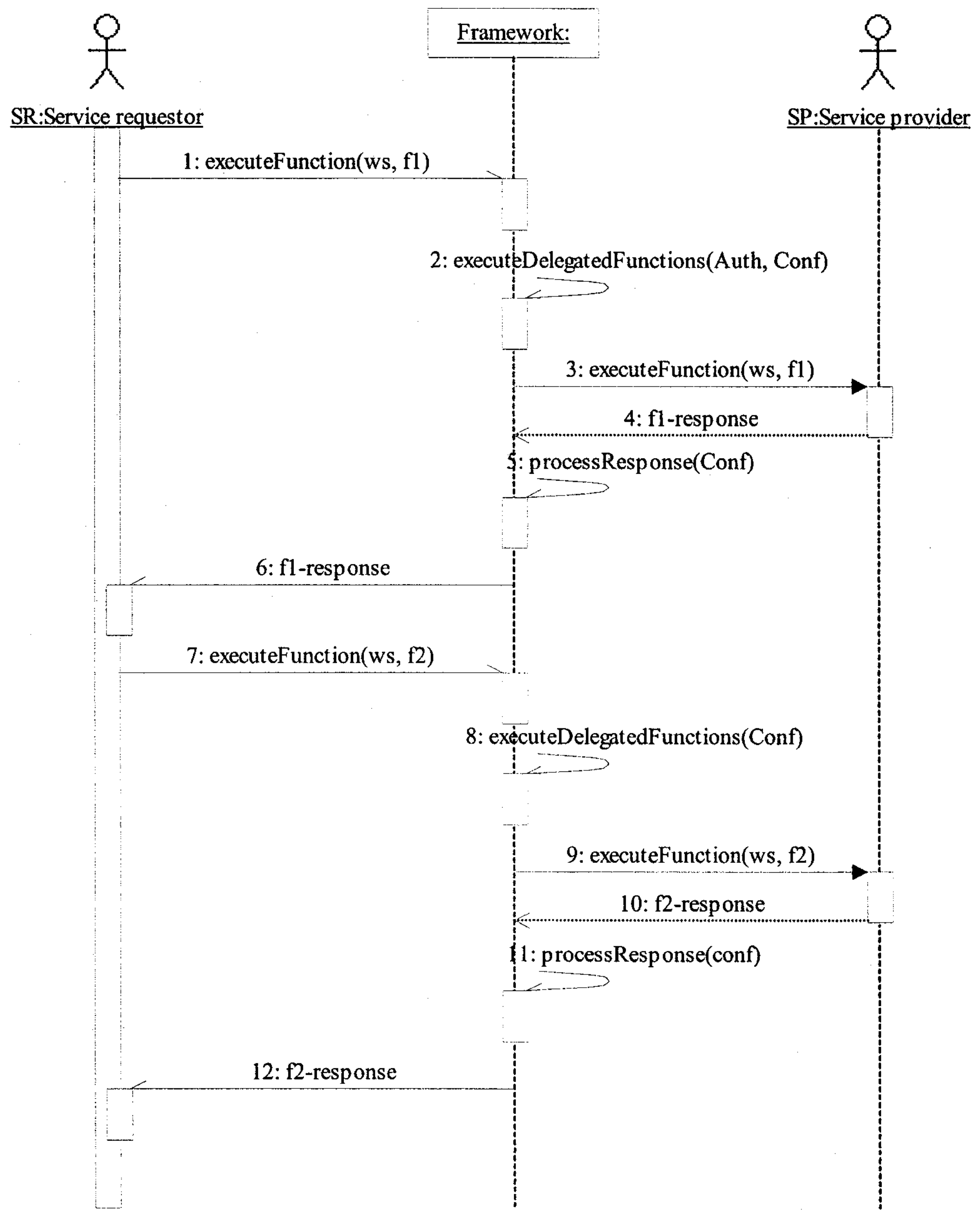
Now, consider the same problem as in the first case and suppose that even the data confidentiality is delegated to the framework. In this case, the framework has to process each message exchanged between the SR and the SP. Then, the whole communication has to pass through it.

```
        executes                    provides    delegates

SR-------------->ws(f1,f2) <--------- SP ------------- > FW{Auth, Data Confidentiality}
```

Figure V.3: Second case study of web services execution

The sequence diagram of this case is as follow:



Framework:
SR:Service requestor
SP:Service provider
1: executeFunction(ws, f1)
2: executeDelegatedFunctions(Auth, Conf)
3: executeFunction(ws, f1)
4: f1-response
5: processResponse(Conf)
6: f1-response
7: executeFunction(ws, f2)
8: executeDelegatedFunctions(Conf)
9: executeFunction(ws, f2)
10: f2-response
11: processResponse(conf)
12: f2-response

Figure V.4: Sequence diagram of the second case study of web services execution

The SR asks for the execution of the function f1, on the service *ws*. The framework authenticates the SR and processes the request to ensure data confidentiality. In step three, the framework transfers the processed request to the SP. It receives the SP response in the 4th step, processes it (always to ensure the confidentiality) and transfers it to the SR.

When the SR asks for the f2 execution in step 7, the framework executes only the confidentiality function, which means that it doesn't re-authenticate the SR. After that, from step 9 to step 12, the framework continues the interaction the same way as seen for the f1 execution.

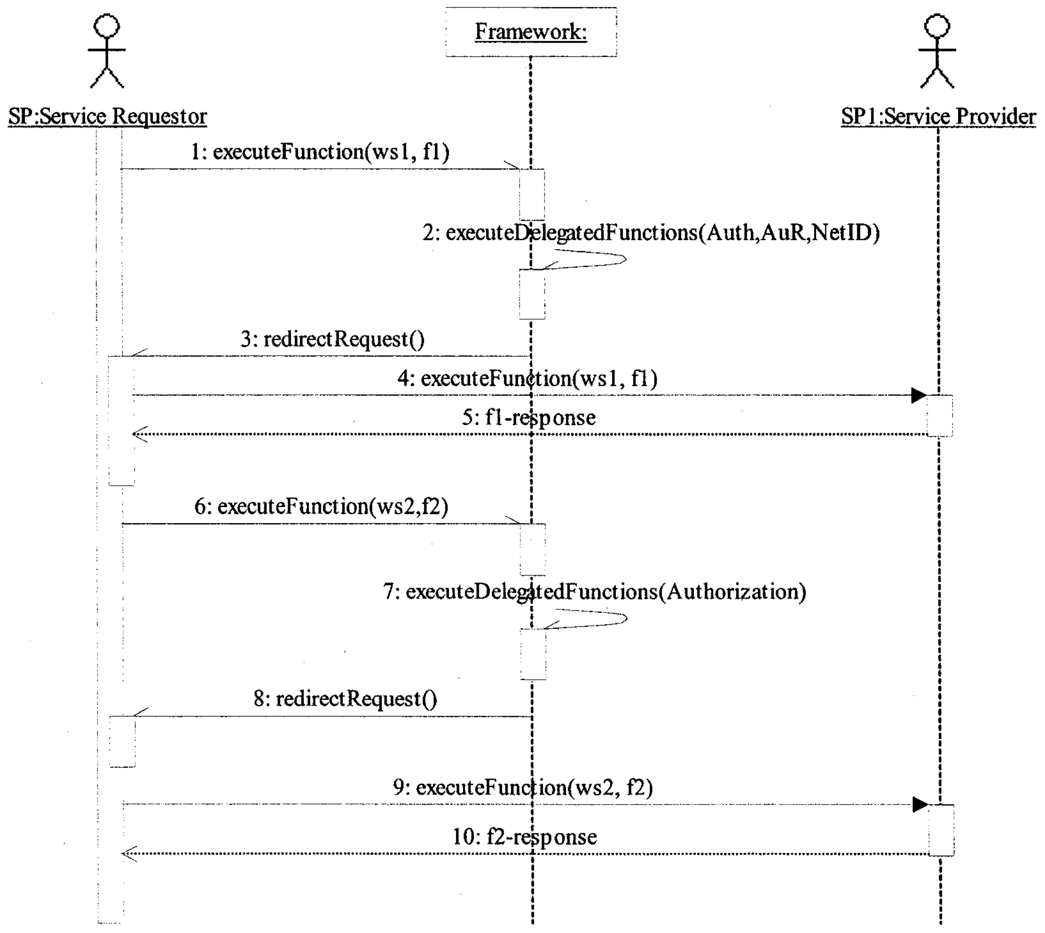
V.1.3 Third case study

Now, consider a more complicated use case. Assume that in the same transaction, the SR tends to execute three functions, f1, f2 and f3. The three functions are offered by three different services ws1, ws2 and ws3 respectively. ws1 and ws2 are provided by the service provider SP1, and ws3 is provided by SP2. SP1 delegates authentication, authorization and network identity management to the framework. SP2 delegates the data confidentiality function too. The problem modeling is as follows:

```
        executes              provides    delegates
RS  ---------------> ws1(f1) <---------- SP1 ------------- >  FW{Auth, Authorization,
                                                                  NetID}
RS  --------------> ws2(f2)  <--------- SP1 ------------- >  FW{Auth, Authorization,
                                                                  NetID}
RS  --------------> ws3(f3)  <--------- SP2 ---------->  FW{Auth, Authorization, NetID,
                                                                Conf}
```

Figure V.5: Third case study of web services execution

The execution sequence diagram of this case is shown in figure V.6a and V.6b. In the first figure, the SR asks for f1 execution. The framework executes all the delegated functions associated to the service provider SP1, and redirects the SR to the SP. After that, the SR asks for f2 execution in step 6. Since the Web Service ws1 is provided by the same service provider as the previous service, the framework executes only the authorization function.



Framework:
SP:Service Requestor
SP1:Service Provider
1: executeFunction(ws1, f1)
2: executeDelegatedFunctions(Auth,AuR,NetID)
3: redirectRequest()
4: executeFunction(ws1, f1)
5: f1-response
6: executeFunction(ws2,f2)
7: executeDelegatedFunctions(Authorization)
8: redirectRequest()
9: executeFunction(ws2, f2)
10: f2-response

Figure V.6a: Sequence diagram of the third case study: interacting with the same SP

When asked for the f3 execution, and since the two service providers delegate network id management to the framework, the framework doesn't re-authenticate the SR. Instead, it uses the network id management function to authenticate the SR to the service provider SP2. However, the framework has to keep the session history in order to know that the SP has already been authenticated within the same session.

The framework executes also the data confidentiality function. After that, the framework transfers the processed request to the SP, processes the received response to ensure data confidentiality, and sends the processed response to the SR (figure V.6b).

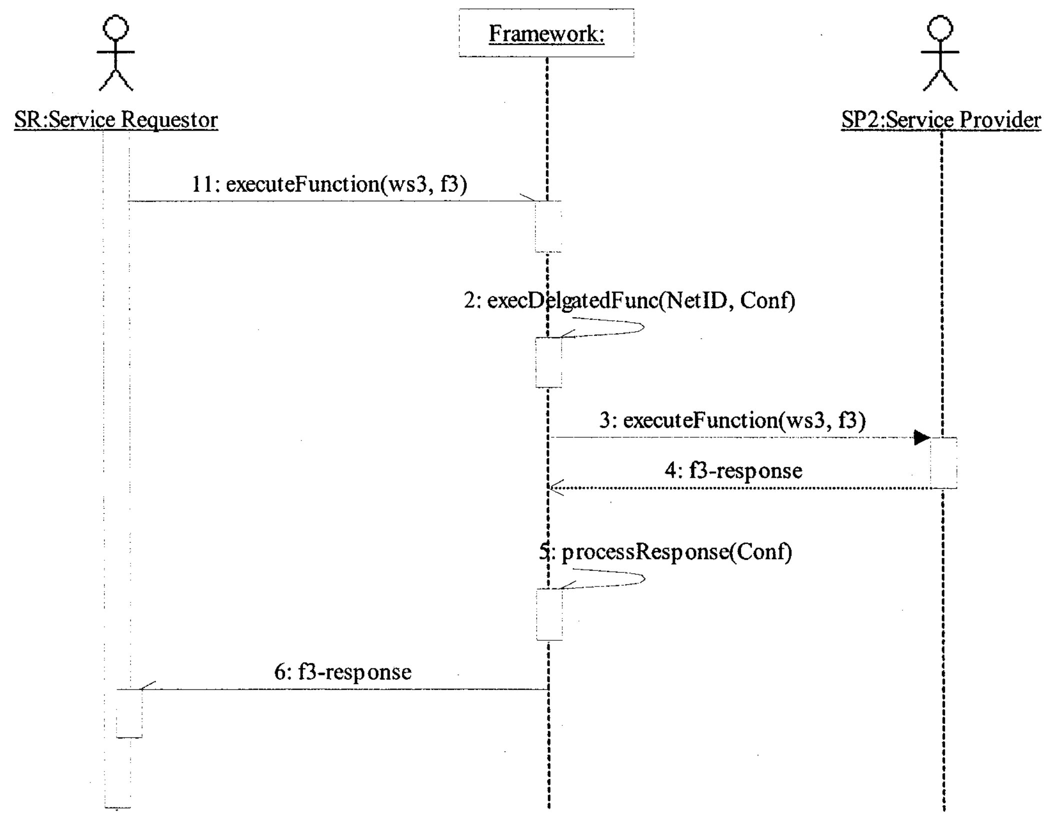


Figure V.6b: Sequence diagram of the third case study: interacting with a different SP

V.2 Use cases analysis

From the previous three use cases, we can deduce that depending on the use case, a given common function may be executed many times or only once during a request processing. For instance, in the second use case, the data confidentiality function is executed each time a message is received or is to be sent. In the third use case, the authentication function is executed once during the whole communication between the requestor and the provider.

We can also point out that in some cases; it is more efficient that the service requestor communicates with the service provider without going throw the framework, after the latter has executed the delegated functions. Such a case is for instance when the framework has just to authenticate the service requestor.

There are two ways to give the requestor the possibility to directly communicate with the provider. The first one is to firstly make the requestor go throw the framework and redirect the request to the service provider after the delegated functions have been executed. The second approach is to send the request directly to the service provider who will ask the framework to execute the delegated functions for him. The first solution keeps the use of the framework transparent to the service requestor and the service provider. The second one requires the service provider to be aware of the process.

V.3 Functional entities execution optimization

Based on the analysis of the studied use cases, we will give some solutions on how to optimize the execution of the framework functional entities.

V.3.1 Common functions execution scope

To ensure that common functions are executed only when needed, we have defined the function execution scope concept. It is the scope on which a given function is executed once. Six function execution scopes are defined and they are:

- Message Scope *(MS)*: a function *f* is said to have a message scope if it's executed once each time the framework has to process a new message (*e.g. Data Integrity, Data confidentiality*)
- Function Scope *(FS)*: relates to the function that is executed once each time a new web service function is invoked *(e.g. Authorization,* if the authorization policies are related to the functions rather that to the service)
- Service Scope *(SS)*: relates to the function that is executed once each time a new service is invoked (*e.g. Authorization,* if the authorization policies are related to the service)
- Service Provider Scope *(PS)*: relates to the function that is executed once within the interaction with the same service provider (*e.g. Authentication*).
- Application Scope *(AS)*: relates to the function that is executed once within the whole client application execution *(e.g. Authentication,* in case the network ID management is used)
- Event related Scope ***(ES)*:** relates to the function whose execution starts and ends

when some events arise (e.g. starts with the application beginning and ends with the application ending (*e.g. charging, SLA enforcement*)).

V.3.2 General rules about the framework execution

During a client application execution, the framework functional entities are executed according to theirs scopes. The general rules associated with the functions scopes are:

- The message scope function (MSF) is applied to each exchanged message. In situation when a MSF function is delegated to the framework, the whole communication between the requestor and the provider has to go through the framework.
- The function scope function (FSF), the service scope function (SSF), the provider scope function (PSF) and the application scope function (ASF) have to be executed once within the same corresponding scope. They are executed once for each function provided by the web service, web service, service provider and client application respectively.
- The event scope functions (ESF) have to be executed and/or terminated when the triggering event arises.

In the case of the two last rules, three possibilities can be considered. The more appropriate one to use depends on the use case. The first possibility is to make the whole communication pass through the framework which executes the common functions when needed. The second possibility is to make the service requestor interact directly with the service provider, and when a common function has to be executed, the latter asks the framework to perform this function. This possibility may be very efficient for instance when the charging mechanism used is "pay per use".

The third possibility is an hybrid of the two previous approaches. The communication alternates between going through the framework and being redirected for a direct interaction between the requestor and the provider. In this case the framework executes some delegated functions and redirects the SR to the SP. The major problem here is how to know if the SR has to directly interact with the SP for all the remaining client application execution, or only for a given scope. The second problem is, if the redirection is total, then how to ensure this total redirection. The next sub-section tries to answer these questions.

V.3.3 Partial and total communication redirection

To invoke a given web service, the service requestor starts by binding to the web service end- point. This end-point may be retrieved from the web service WSDL file or given to the requestor by other means. In case the framework is to be used for common functions, two solutions are possible. As an end-point, we can either give the framework URL or simply give the service provider URL.

In the first case (figure V.7), the framework is used as end-point of the service to execute. Then, the service requestor (SR) starts by binding to the framework. This means that each new function call passes through the framework. Therefore, for a global redirection of the communication to the service provider (SP), the framework has to change the binding URL on the fly.

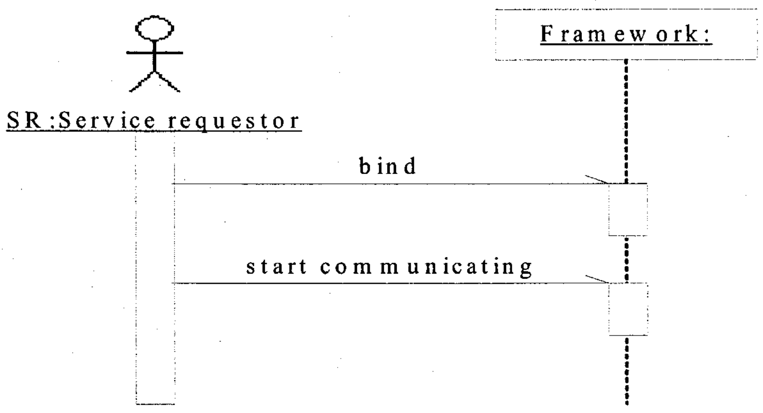


Figure IV.7: The service end-point is the framework

When the end-point is the service provider, we have to determine if the service requestor intervention is needed for the execution of the delegated functions or not. If no SR intervention is needed such as for charging, the SR can bind to the SP and start communicating directly with it. When a common function needs to be executed, the SP asks the framework to do it, receives the execution result and continues interacting directly with the SR (figure V.8).

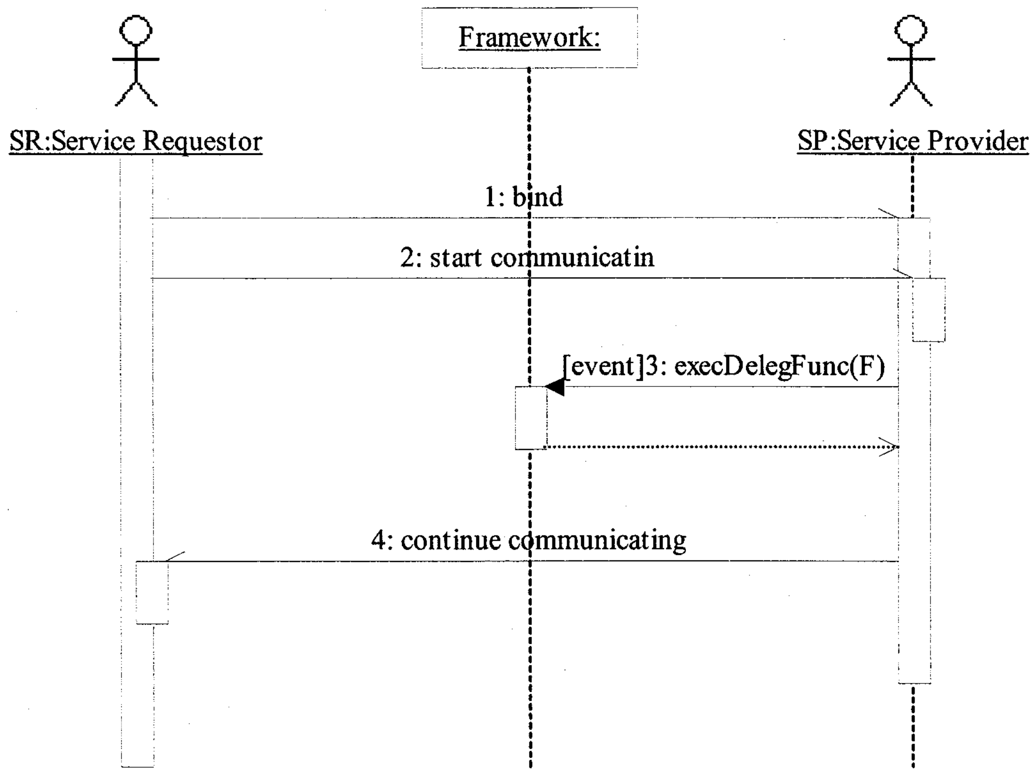


Figure V.8: No SR intervention is required for common functions execution

When a SR intervention is required (e.g. for authentication), the framework has to be given the possibility to directly interact with the SR for the execution of the concerned common function (the one that requires SR intervention). When a common function execution is needed, the service provider redirects the service requestor to the framework. The framework executes the required function by interacting with the SR and redirects the communication to the SP (figure V.9).

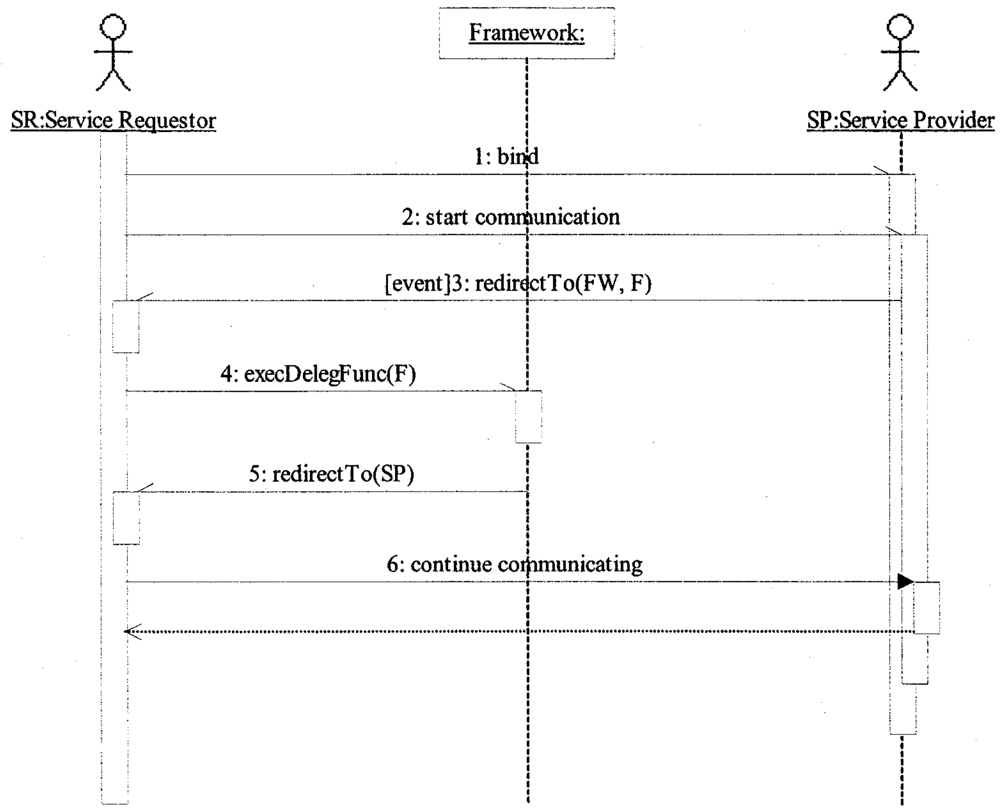


Figure V.9: The SR intervention is required for the common function execution

Now the question is: how to use these different information resources to automatically monitor a client application execution.

V.4 Client application execution monitoring

To automatically monitor a client application execution, the following steps are to be followed:

- Specify the execution scope of each common function to be delegated to the framework.

- Configure the session timeout to be used. This time out is used to determine when a client application execution session has to be terminated. The session timeout parameter can be different for each service provider and/or web service.
- Specify the service provider and the framework URLs.
- Specify for each common function and service provider combination if the communication is to be redirected to the service provider after the execution of this function.
- At execution time, the framework
    - Creates a new session when the first invocation of a service is received.
    - Executes the delegated functions according to their execution scopes.
    - Redirects the communication between the SR and the framework to the SP if needed.
    - Terminates the session after the configured time out.

V.5 Conclusion

The proposed architecture for the common functions framework was introduced and an approach for optimizing the framework execution was presented. Now we have to prove that what is proposed is valid. The following chapter presents a proof of concept prototype.

Chapter 6:

Implementation

As a proof of concept, we have built and tested a framework prototype using a call control scenario. This chapter gives an overall overview of this prototype. It starts by presenting the implemented prototype. Then, it gives an execution scenario and it ends with some performance measures.

VI.1 Prototype

As a prototype, we have implemented the framework interface and the part of the policy server related to the delegation. For each other common function (security, SLA, charging, ...), a simple web service is implemented to display that the function has been invoked. For testing, existing web service and client application are used.

VI.1.1 Prototype architecture

The general architecture of the implemented prototype is given in the figure VI.1. The framework has to:

- Get the request
- Create new session when necessary
- Find out the common functions to be executed and theirs execution scopes
- Process the request by executing necessary functions
- Call the web service and gets the service execution results

- Send results to the client application.
- Terminate the existing sessions after a configurable timeout.

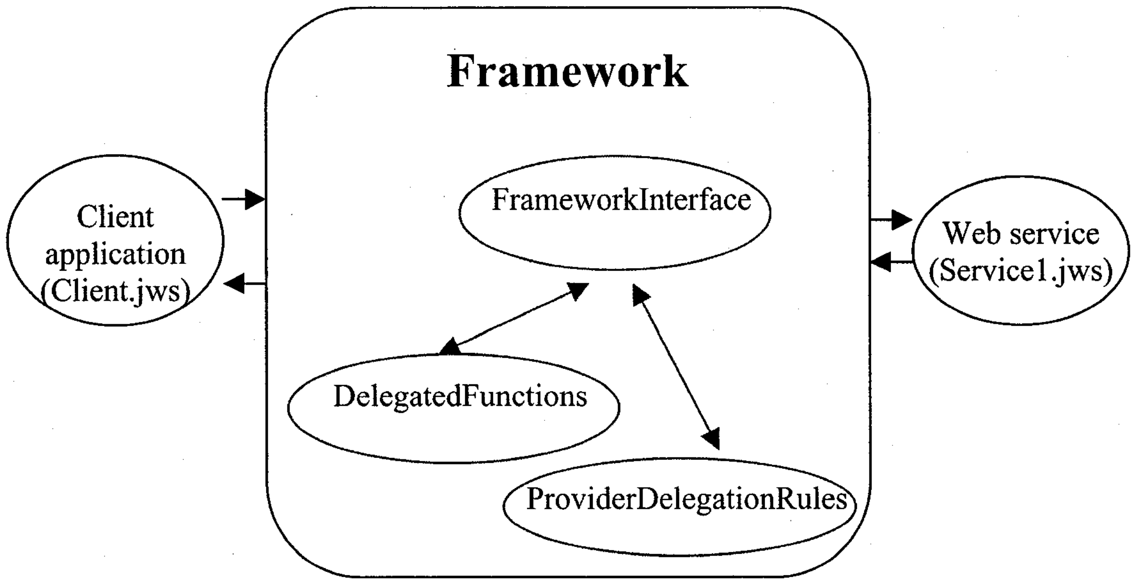


Figure VI.1: General prototype architecture

The whole communication between the service requestor and the service provider goes through the framework. The framework URL is given as end-point of the used web service. It replaces the service provider URL in the associated WSDL file. The effective service provider URL is given in the web service table description in the database. The framework interface and the delegation rules management function are implemented as web services.

VI.1.2 Database schema

The implemented policies are simple and don't require the use of a policy server. Then, a relational database is used for policies provisioning. This database is used to keep information about the common functions to be executed by the framework and the relating delegation information. The delegation information is captured into delegation rules. The figure VI.2 below gives an overall overview of the tables used and theirs relationships.

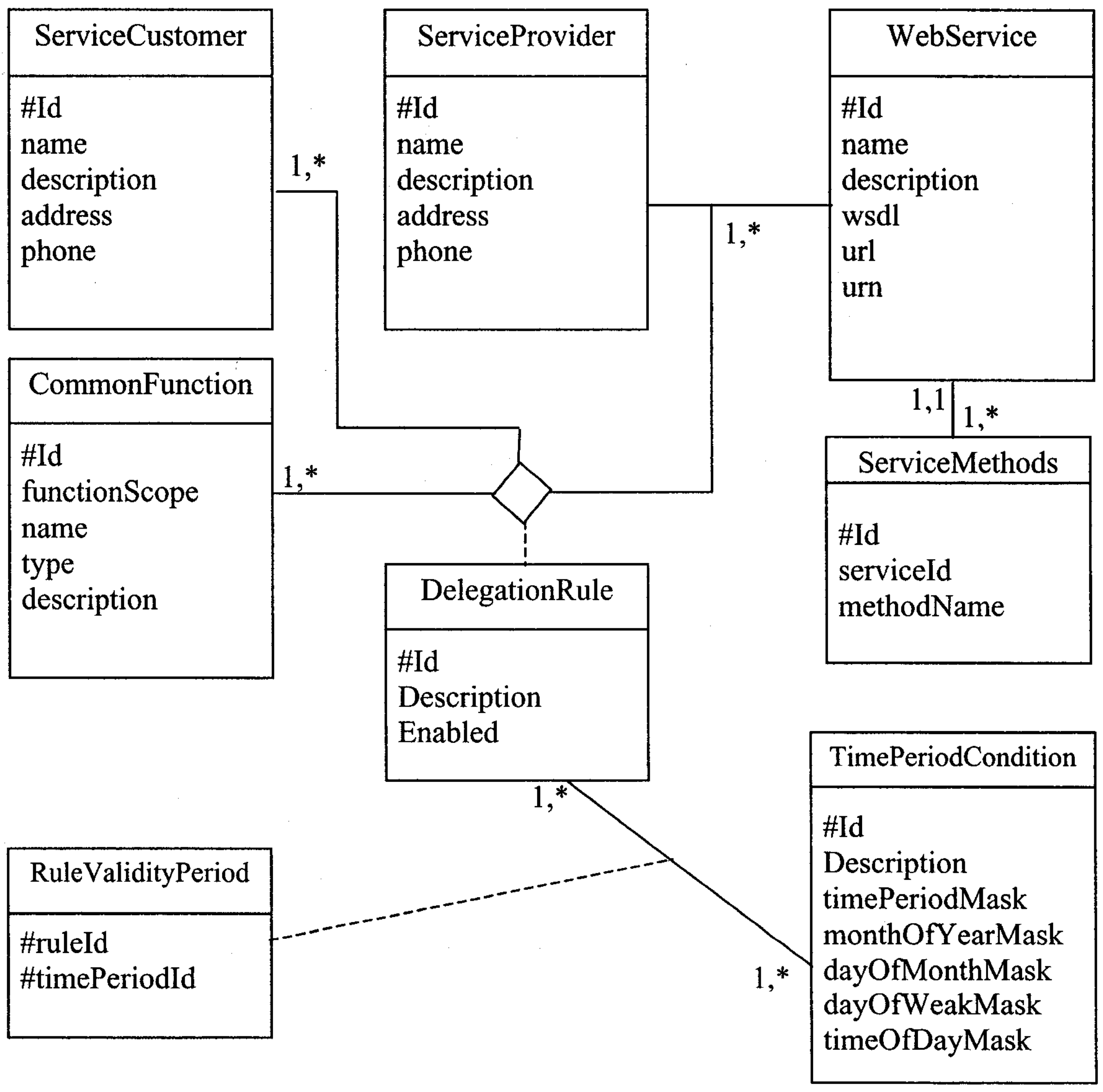


Figure VI.2: Delegation rules database

The delegation rule is an association between the service requestor, the service provider, the web service and the common function. For each delegation rule at least one delegation condition is associated. It presents the condition(s) under which the delegation rule is valid, meaning that the corresponding common function is delegated to the framework. The conditions taken into account are time period relating. A given function is executed by the framework if at least one associated time period condition is valid.

From the previous diagram, we can deduce that the *DelegationRule* table is as follow:

DelegationRule
#Id
description
enabled
wServiceId
providerId
customerId
delegatedFunctionId

Figure VI.3: Delegation rule table

If a given delegation rule is applicable to all customers, the customerId (in DelegatinRule) is set to null. The same for wServiceId, providerId and delegatedFunctionId. For a given provider, we can have a delegation rule which can be applied to all customers and web services, and other specific rules.

However, if for a given provider we have a rule such that:

- Enabled = true
- customerId = null
- wServiceId = null
- delegatedFuncId = null

which means that all functions are delegated to the framework, we can't have another delegation rule for this provider.

VI.1.3 Function scopes and parameters format used

The execution scopes of the common functions used are presented in the table VI.1 here after.

Common function	**Execution scope**	**Comments**
Authentication	PS	Executed the first time is required AND each time a web service belonging to a different provider that doesn't delegate the Network ID to the framework is invoked.
Integrity	MS	Executed once for each exchanged message.
SLA	MS	Executed once for each exchanged message.
Authorization	FS	Executed once each time a different function is called.
Charging	ES	Executed at each function call: it's just like charging the client application for the usage number.
Privacy	AS	Executed once during each session.
Network Identity	AS	Executed once during each session.

Table VI.1: Common function execution scopes used

For the time period condition attributes, the used formats are presented in the table VI.2 bellow:

Attribute	**Format**	**Default**
timePeriodMask	yyyymmddThhmmss/ yyyymmddThhmmss	THISANDPRIOR/ THISANDFUTURE
monthOfYearMask	bbbbbbbbbbbb	111111111111
dayOfMonthMask	bbbbbbbbbbbbbbbbbbbbbbbb bbbbbbb	1111111111111111111111111111111
dayOfWeakMask	bbbbbbb	1111111
timeOfDayMask	Thhmmss/Thhmmss	T000000/T235959

Table VI.2: Time period condition attributes format

- timePeriodMask presents the mask of the period during which the delegation rule is valid. It includes the starting and the ending dates of the period. The default value is "THISANDPRIOR/THISANDFUTURE" which means that the rule is valid on every time.
- monthOfYearMask is a sequence of 12 bits. If a bit is set to 1 it means that the rule is valid on the corresponding month.
- dayOfMonthMask is a sequence of 31 bits. If a month has less than 31 day, no existing days are set to 0. The activated days are those whose corresponding bits are set to one.
- dayOfWeakMask is a sequence of 7 bits. The day one is Sunday and the activated days are those whose corresponding bits are set to one.

- timeOfDayMask presents the time period of the day on which the rule is valid. By default, the rule is valid for the whole day.

VI.1.4 Implementation classes

The main implementation classes are presented in the figure VI.4. The framework interface web service provides methods to process the incoming requests. When a request is received, the "processRequest" method is invoked. This method:

- Checks if there is a current valid session for the current client
- Gets the list of delegated functions
- Executes these functions according to theirs scopes by calling the corresponding server (e.g. security server, charging server, …) using the "callProcessingServer" method.

The "IDelegationManagement" interface is the most important part of the delegation management. It includes the main methods to manage the delegated functions. It provides necessary means:

- To get and to set the delegated functions list
- To add and to remove a given function to or from the delegated functions list
- To remove all the delegated functions from the list.

The "ProviderDelegationRules" class implements "IDelegationManagement" and provides additional methods to get the delegated functions according to the associated delegation rules. A given common function is added to the delegated functions list only if it is valid at the processing time.

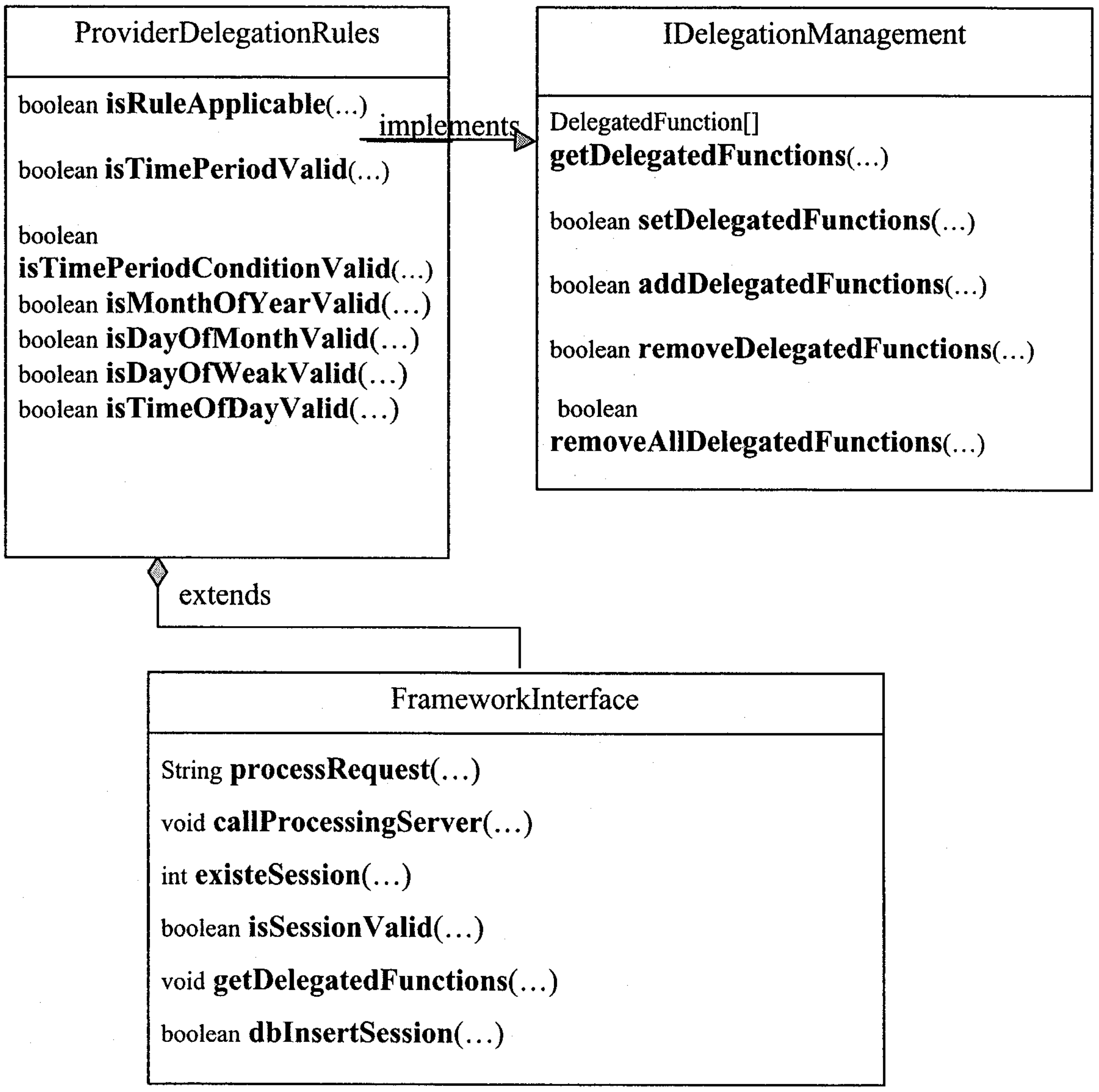

ProviderDelegationRules
boolean isRuleApplicable(...)
boolean isTimePeriodValid(...)
boolean isTimePeriodConditionValid(...)
boolean isMonthOfYearValid(...)
boolean isDayOfMonthValid(...)
boolean isDayOfWeakValid(...)
boolean isTimeOfDayValid(...)
implements
IDelegationManagement
DelegatedFunction[] getDelegatedFunctions(...)
boolean setDelegatedFunctions(...)
boolean addDelegatedFunctions(...)
boolean removeDelegatedFunctions(...)
boolean removeAllDelegatedFunctions(...)
extends
FrameworkInterface
String processRequest(...)
void callProcessingServer(...)
int existeSession(...)
boolean isSessionValid(...)
void getDelegatedFunctions(...)
boolean dbInsertSession(...)

Figure VI.4: Implementation classes

VI.1.5 Session management implementation

For session management, a session table is used (figure VI.5). This table includes especially information about the client to which this session belongs, the called service and the starting date of the session. The table includes also the id of the provider to whom the invoked service belongs and the name of the called function. When a request is received, if no valid session exists, a new one is created. The session time out is a configurable parameter which is set for tests to 30 minutes.

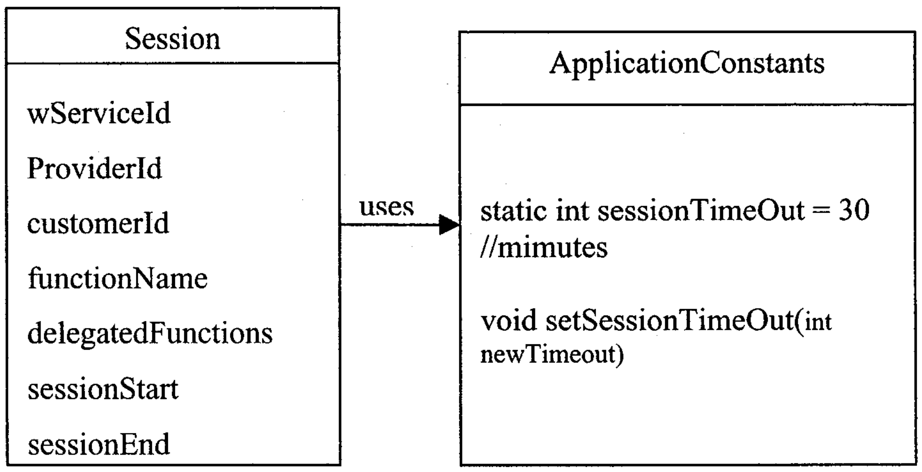


Figure VI.5: Session management table

To verify if a valid session exists, we use the "**existeSession**" method of the "**frameworkInterface**" web service. This method returns:

- 0 if no session exists for the current customer or if a session exists but it is no more valid.
- 1 if a session exists for the current customer, and the same function has already been executed within this session.

- 2 if a session exist for the current customer, and another function is executed on the same service that the current function to be executed belongs to.
- 3 if a session exist for the current customer, and the invoked service belongs to a service provider whose different service has been invoked during the same session.
- 4 if a session exist for the current customer, but no service belonging to the current provider has been invoked during this session.

These results are used to automatically monitor the common functions execution. For instance, assume that a PSF function f is included in the delegated functions list. If the "existeSession" method returns 4, the function f is executed, otherwise, it's not. The following table gives a summary of when each delegated common function is executed according to "existeSession" method results.

Exist session result	**Delegated function scope**	**Function executed**
	MS	
0	FS	YES
	SS	
	PS	
	AS	
	MS	YES
	FS	
1	SS	
	PS	NO
	AS	
	MS	YES
	FS	
	SS	
2	PS	NO

	AS	
	MS	
	FS	YES
3	SS	
	PS	
	AS	NO
	MS	
	FS	YES
4	SS	
	PS	YES
	AS	NO

Table VI.3: Common functions execution conditions

ES functions are not included in this table, seen that their execution is related to some triggering events. Some of these events can be deduced from the previous results. For instance, if the charging mechanism used is "pay-per-use", the charging function is executed each time a new request is received (whatever the "existSession" result is).

VI.1.6 Platform used

As development platform we are using:

- WebLogic platform 8.1 with service pack 2. We are especially using:
    - The application server
    - The development environment (workshop)
    - The UDDI server.
    - The PointBase database.
- Pentium 4, CPU 2.66 GHz, 512 MB of RAM.

VI.2 Testing scenario

For test purposes, we are using an existing web service and an existing client application that uses this service. The introduction of the use of the framework is transparent to the web service and the client application. Indeed, the only thing to change is the web service binding address in the associated WSDL file. This sub-section presents the applications used for the testing, the database data used and the testing scenario execution.

VI.2.1 Applications used

To test the implemented prototype, a call control scenario is used. The client application is calling a conferencing web service by going through the framework. This application is implemented using *JBUILDER* 9.0, whilst the conferencing web service is implemented using *WebLogic platform 8.1 with service pack 1.* The framework application, the conferencing web service and the client application are running on different machines.

The conferencing web service allows the client application to:

- Initiate and/or terminate a conference between a given number of end users.
- Create and/or end a sub-conference.
- Move a given user from one sub-conference to another or from the main conference to a sub-conference and vise-versa.
- Remove a user from the main conference or from a sub-conference.

VI.2.2 Database data used

The database data related to the conferencing web service are given here-after (tables VI.4, VI.5, VI.6 and VI.7). The conferencing web service WSDL file is changed by replacing the conferencing web service binding address by the framework binding address. The conferencing web service binding address is given in the "webService" table, under the URL column.

ID	3
NAME	ConferenceService
DESCRIPTION	Synchronous conferencing web service
WSDL	http://142.133.72.82:7001/call_control/ConfInt/Service/ConferenceService.wsdl
URL	http://142.133.72.82:7001/call_control/ConfInt/Service/ConferenceService.jws

Table VI.4: Conferencing web service information

ID	SERVICEID	METHODNAME	PROVIDERID
1	3	setConfiguration	2
2	3	initiateConf	2
3	3	endConf	2
4	3	addUser	2
5	3	removeUser	2
6	3	initiateSubConf	2
7	3	endSubConf	2
8	3	moveUser	2

Table VI.5: Conferencing web service methods

ID	NAME	DESCRIPTION	PHONE	ADDRESS
2	ConferenceClient	A Jbuilder application calling the conferencing service developed in the ericsson lab		Ericsson Montreal, canada

Table VI.6: Conferencing web service provider information

ID	NAME	DESCRIPTION	PHONE	ADDRESS
2	ConferenceClient	A Jbuilder application calling the conferencing service developed in the ericsson lab		Ericsson Montreal, canada

Table VI.7: Conferencing web service customer information

When executing the conferencing web service, all the common functions are delegated to the framework (table VI.8). The time period condition associated with this delegation rule is always valid (table VI.9). This means that all the common functions needed for the conferencing web service execution are always delegated to the framework. The information relating to these common functions and their execution order is given in the table (VI.10).

ID	16
CUSTOMERID	NULL
PROVIDERID	2
WSERVICEID	NULL
DELEGATEDFUNCID	NULL
ENABLED	TRUE
DESCRIPTION	Delegation rule for provider 3 valid for all web services and customers

Table VI.8: Delegation rule associated to the conferencing web service usage

ID	1
DESCRIPTION	Time Condition which is always valid
TIMEPERIODMASK	THISANDPRIOR/THISANDFUTURE
MONTHOFYEARMASK	111111111111
DAYOFMONTHMASK	1111111111111111111111111111111
DAYOFWEAKMASK	1111111
TIMEOFDAYMASK	T000000/T235959

Table VI.9: Time period condition associated to the conferencing web service usage

ID	NAME	DESCRIPTION	TYPE	FUNCTION-SCOPE	EXECUTION-ORDER
1	Authentication	Authenticates customers	Security	PSF	1
2	Integrity	Ensure messages exchanged integrity	Security	MSF	4
3	SLA	Service Level Agreement function	SLA	MSF	6
4	Charging	Charge the user for service usage	Charging	ESF	7
6	Privacy	Protects user privacy according to the existing privacy regulations	Privacy	ASF	5
7	NetworkId	Manages Network Identity	NetworkId	ASF	2
8	Authorization	Determines if the customer is authorized to execute a given function	Security	FSF	3

Table VI.10: Common functions info

VI.2.3 Client application execution scenario used

One of the testing scenarios used is as follow:

1. The client application asks for conference initiation by invoking the *intiateConf* method. The conference is initiated between three users given as *initiateConf* parameters.

2. A new user is added to the conference using the addUser method.
3. A new sub-conference is created via the initiateSubConf method.
4. Two users are moved to the created sub-conference using the moveUser method.
5. One user in the sub-conference is moved back to the main conference using the moveUser method.
6. The other user in the sub-conference is removed from the sub-conference using the removeUser method.
7. The sub-conference is ended via the endSubConf method.
8. The main conference is ended via the endConf method.

In the first step, the framework executes all the common functions according to their execution order. In the other steps, the common functions are executed according to their execution scopes. The list of the common functions executed at each step is given in the table here after. They are presented in the order of theirs execution.

Step	**Common functions executed Description**
1	• Authentication • NetworkId • Authorization • Integrity • Privacy • SLA • Charging

2,3, 4,6,7,8	• Authorization • Integrity • SLA • Charging
5	• Integrity • SLA • Charging

Table VI.11: Common functions execution

VI.3 Some performance measurements:

The performance measurements performed aims to measure the overhead introduced by the use of the framework for providing common functions. They were taken in the week-end in order to minimize the overhead of other applications. The machines used are connected to a 100 Mb/s Ethernet LAN segment.

The taken measurements and their analysis are based on the assumption that the deployment pattern of the web service and the required common functions is the same with and without using the framework.

VI.3.1 Time delay overhead

To get the time delay overhead generated, we have measured the execution time delay of each function provided by the conferencing web service used (initiateConf, endConf, addUser, removeUser, initiateSubConf, endSubConf, moveUser). The execution time delay of a given function is calculated as the time duration between the function call and the reception of the feedback message from the web service.

Two sets of measures were taken. The first one is without using the framework. The second one is by going through the framework. Each set consists of 17 trials. The overhead generated by the framework usage is calculated as the delay difference between the two measurements sets.

For each measurements set, the measurements were taken in three cases. They include measurements for a conference with three, four and five participants. The measuring scenario is based on the following conditions: Participants are always online and available at conference initiation and they always accept the invitation. The sequence of actions executed for time delay measurements is as follow:

1. The conference is initiated with the corresponding number of users (3, 4 or 5).
2. A participant is removed and added again to the conference
3. A sub-conference is created with random participants
4. A participant is moved from the sub-conference to the main conference
5. The sub-conference is ended.
6. The main conference is ended.

The figure VI.6 here-after presents the collected data and the generated delay overhead representation. The figure VI.6.a presents the measured average time delay without using the framework. The figure VI.6.b presents the percentage of the average time delay of each function execution with using the framework, according to the average time delay of the execution of the same function without using the framework. The figure VI.6.c presents the time delay overhead introduced by the framework usage for each executed function. The figure VI.6.d presents the percentage of the time delay overhead introduced by the framework usage for each function, according to the average time delay of the same function execution without using the framework. Another graph [figure VI.7] presents the framework intern processing delay, according to the web service function executed and the number of participants.

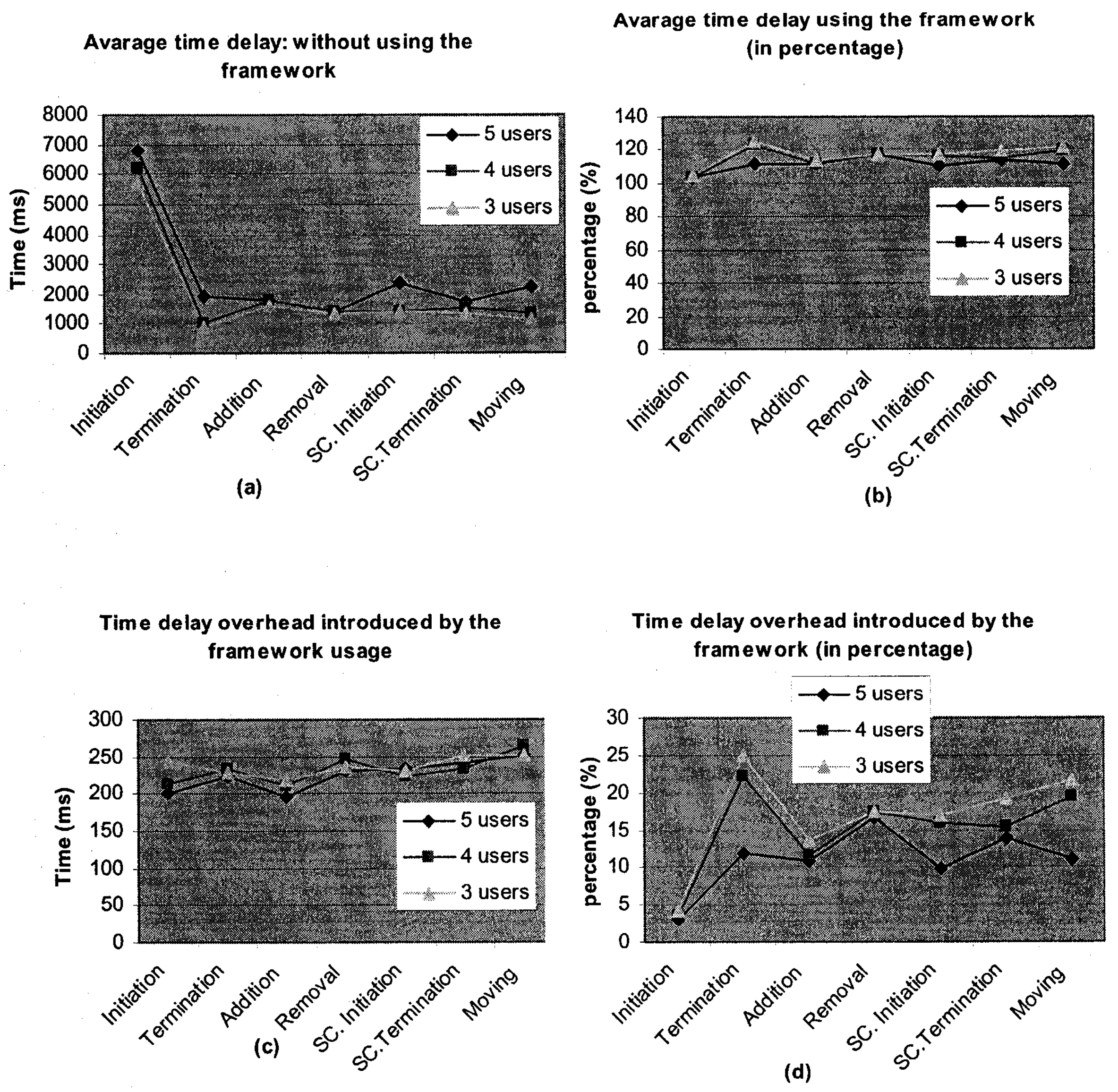


Figure VI.6: Average time delay and time delay overhead: (a) average time delay of functions execution without using the framework. (b) Average time delay of functions execution using the framework (c) time delay overhead generated by the framework usage (d) percentage of the time delay overhead generated by the framework usage

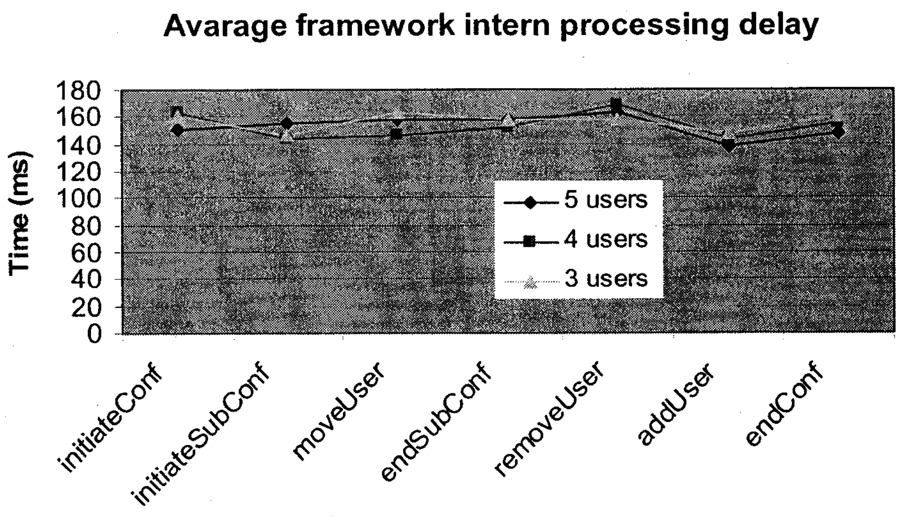


Figure VI.7: Average framework intern processing delay

VI.3.2 Network load overhead

In the implemented prototype, the common functions are not implemented. Each common function is presented by a simple web service that just prints out that the function has been called. So, no changes are made to the interchanged messages (figure VI.7). The framework uses exactly the same received SOAP message to invoke the conferencing web service. The same thing is done for response messages received from the web service.

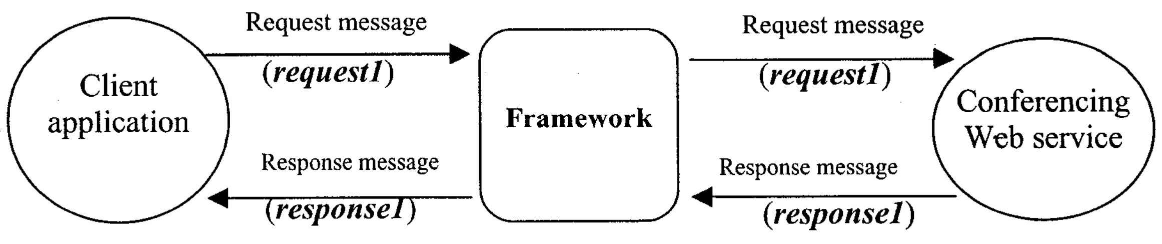


Figure VI.7: Exchanged messages during the conferencing web service execution

So, no network overhead is generated in case of the implemented prototype. However, an additional network load overhead may be generated in case the common functions are implemented. The generated overhead will depend on the way these functions are implemented and deployed.

This overhead will be minimized in case the common functions implementations reside in the same server as the framework implementation. In this case, the only overhead generated will be due to the modification of the incoming messages, before being transferred to their destination. For instance, the security server may add some security assertions to the message to transmit. Nevertheless, this overhead will be more and less the same, if in the initial configuration (without using the framework) the corresponding function is executed outside the server hosting the web service implementation.

In case the common functions and the framework implementations are deployed in a distributed environment (figure VI.8), another type of load overhead will be generated. It results from the communication between the framework and the common functions implementations. This will generate also an additional time delay overhead.

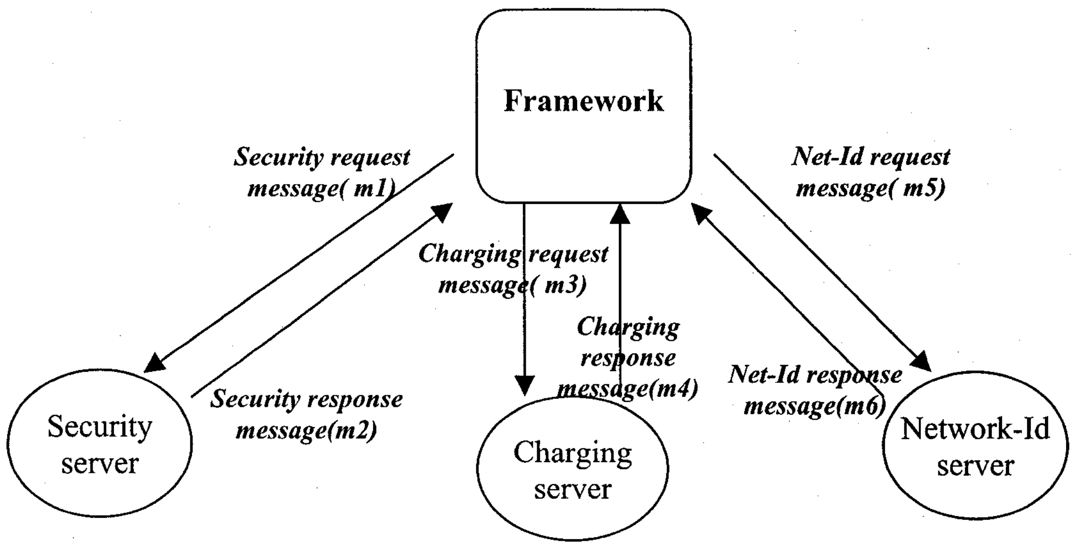

Framework
Security request message(m1)
Security response message(m2)
Charging request message(m3)
Charging response message(m4)
Net-Id request message(m5)
Net-Id response message(m6)
Security server
Charging server
Network-Id server

Figure VI.8: Example of exchanged messages in distributed environment

VI.3.3 Measurement analysis

As we can notice in the figure (VI.6c), the time delay overhead induced is slightly affected by the function executed or the number of participants. In fact, almost the same set of actions is performed in the framework side in each case. Indeed, when a request is received, the framework interrogates the database to execute the same set of operations (determines if a valid session exists and identifies the delegated functions list), executes the required web service function and transmits the execution results to the client application. The taken measurements show that the framework intern processing delay is almost constant with respect to the function to execute and the number of participant. The average framework intern processing delay is 154 ms.

The slight variations observed on the time delay overhead result from the variation of the establishment delays of each TCP connection, and from the differences in database manipulations. The average time delay overhead is 232 ms. This represents 10% overhead of the average time delay when the framework is not used. This overhead will not be affected if the common functions are implemented. In fact, the required common functions have to be executed even if the framework is not used.

The time delay introduced by the framework is barely noticed by the client application's users. Therefore, the use of the framework does not penalize the system's performance. Furthermore, the use of the framework frees the applications developers of conceiving and developing solutions for common functions. This will save time and footprint.

Chapter 7:

Conclusion and future work

Using web services for applications development in next generation telecommunications networks raises two main issues. The first one concerns the definition of web services for making telecommunications capabilities available to applications, and the second one relates to enabling the use of Web services in telecommunications by providing common supporting functions. In this chapter, we will summarize the contributions of this thesis and resume how it handles the issues to be resolved. We will also give some hints about the future work.

VII.1 Contributions of this thesis

In this thesis, we have focused on the second issue. We have started by identifying the relating requirements. Then, we have specified a novel architecture that fulfills these requirements. After that, we have optimized the proposed architecture and we have implemented a proof of concept prototype.

The requirements we have identified are based on OMA ones. They are of two types. The first type concerns the common functions of web services when used in telecommunications settings (e.g. security, charging). The second one is the consistency and the easy use of the framework to propose for providing the identified common functions.

One of the main assumptions that the proposed framework architecture is based on is that the network operator may choose to perform some of the common functions itself and delegate the others to the framework.

The proposed architecture is a web service–based architecture, where each common function is presented as a web service. An internal registry is used to keep information about these web services which inter-communicate using functions calls. Functions parameters depend on the function to be executed. The policies that govern the execution of the framework functions are captured into the policy server. To find out the appropriate policies, the framework can either use the local policy server or the one belonging to the service provider.

Depending on the use case, a given common function may be executed many times or only once during a request processing. For instance, the authentication function is executed once during the whole communication between a given requestor and provider, where as the data integrity function has to be executed each time a message is received or has to be sent. To ensure that common functions are executed only when needed, we have defined the function execution scope. It is the scope on which a given function is executed once. For instance, the authentication is a provider scope function and data integrity is a message scope function.

As a prototype, we have implemented the framework entry point and the part of the policy server related to the delegation. The most important part of the delegation management is the “IDelegationManagement” interface. It provides necessary means to get and to set on the fly the delegated functions list, to add and to remove a given function to or from the delegated functions list and to remove all the delegated functions from the list. The prototype has been tested using a call control scenario. Some performance measurements have been taken and they show that the overhead generated by the use of the framework is acceptable.

VII.2 Items for future Work

The defined architecture gives a global solution for providing common functions for using web services in telecommunications settings. However, we still need more details about how each common function can be provided. This can be addressed in future work.

The proposed architecture is more suitable for current centralized telecommunications networks. The common functions framework must be pre-installed in the networks before being utilized. This is not compatible with peer-to-peer and ad-hoc networks principals. In future work, we will look at how to adapt this architecture to peer-to-peer and ad-hoc networks.

REFERENCES

[1] Telcordia Technologies, "Next Generation Network (NGN) Services", White paper, http://www.mobilein.com/NGN_Svcs_WP.pdf

[2] ITU-T Recommendation H.323, "Packet based multimedia communications systems," Geneva, 2002

[3] H. Liu and P. Mouchtaris, "Voice over IP Signaling, H.323 and Beyond," IEEE Communications Magazine, vol. 38, no.10, pp. 142.148, October 2000

[4] J. Rosenberg et al, "Session Initiation Protocol (SIP)", RFC 3261, IETF, June 2002.

[5] R. Braden et al, "Resource ReSerVation Protocol (RSVP)", RFC 2205, Version 1, Functional Specification, September 1997

[6] H. Schulzrinne et al, "RTP: A Transport Protocol for Real-Time Applications", RFC 1889, January 1996

[7] M. Handley and V. Jacobson, "SDP: Session Description Protocol," RFC 2327, IETF, April 1998

[8] Henning Schulzrinne, Jonathan Rosenberg, "The session initiation protocol: Internet-Centric Signaling", IEEE Communications magazine, October 2000

[9] J. Lennox, J. Rosenberg, and H. Schulzrinne, "Common Gateway Interface for SIP," RFC 3050, IETF, January 2001

[10] JCP Java SIP Servlet API, JSR 116 at http://jcp.org/aboutJava/Communityprocess/review/jsr116/

[11] Java Community Process^{SM} , "JAIN Java Call Control (JCC) Application Programming Interface (API)", Version 0.8.4, September 2000.

[12] Parlay 4.1 specifications at http://www.Parlay.org/specs/index.asp

[13] L. Lennox and H. Schulzrinne, "Call Processing Language Framework and Requirements," RFC 2824, IETF, May 2000

[14] W3C, "Web Services Architecture", W3c Working Group Note 11 February 2004. http://www.w3.org/TR/ws-arch/

[15] Adam Bobsworth in ACM Queue, Vol1, No1

[16] W3C web site for XML at http://www.w3c.org/XML/

[17] W3C web site for SOAP at http://www.w3c.org/TR/SOAP/

[18] W3C web site for WSDL at http://www.w3c.org/TR/wsdl/

[19] OASIS standards consortium web site for UDDI at http://www.uddi.org/

[20] "Parlay APIs 4.0, Parlay X Web Services", White paper, version 1.0, December 2002

[21] T. Thomson et al., CPXe: Web services for Internet Imaging, IEEE Computer Magazine, October 2003

[22] N. Alameh, "Chaining Geographical Information Web Services," IEEE Internet Computing Magazine, vol. 7, no. 5, pp. 22-29, September 2003

[23] The OMA Web Services Enabler core specification v1.1 (16-02-2004) at http://member.openmobilealliance.org/ftp/public_documents/mws/2003/

[24] "Web Services Security: SOAP Message Security", Working Draft 18, Friday, 15 July 2003. http://www.oasis-open.org/committees/download.php/1044/WSS.

[25] W3C, "XML-Signature Syntax and Processing", W3C Recommendation 12 February 2002

[26] IETF, "X.509 Internet Public Key Infrastructure Online Certificate Status Protocol – OCSP", RFC 2560

[27] W3C, "XML Key Management Specification (XKMS)", W3C Note 30 March 2001, http://www.w3.org/TR/xkms/

[28] Official OASIS web site: http://www.oasis-open.org/

[29] Liberty Alliance Project web site, http://www.project-liberty.org/

[30] "The Platform for Privacy Preferences 1.0 Specification", W3C recommendation, 16 April 2002. http://www.w3c.org/TR/2002/REC-P3P-20020128/

[31] OMA, "OMA Mobile Web Services Requirements", Draft Version 1.1

[32] "Assertions and Protocol for the OASIS Security Assertion Markup Language (SAML)", OASIS Standard, 5 November 2002. http://www.ouasis-open.org/committees/security/docs/.

[33] OASIS, "Bindings and Profiles for the OASIS Security Assertion Markup Language (SAML)", v1.1, Last call working draft 06, 2 May 2003

[34] OASIS, "Web Services Security: SAML Token Profile", Working Draft 06, 21 February 2003

[35] Web Services Enhancements (WSE) http://msdn.microsoft.com/webservices/building/wse/default.aspx

[36] "Securing Web Services with the Xtradyne Web Services Domain Boundary Controller™" .http://www.xtradyne.com/documents/whitepapers/WS-DBC-WhitePaper.pdf

[37] "The WSLA Framework: Specifying and Monitoring Service Level Agreements for Web Services". Alexander Keller, Heiko Ludwig. Journal of Network and Systems Management (JNSM), Vol. 11, No 1, March 2003

[38] Liberty Alliance Project, "Introduction to the Liberty Alliance Identity Architecture", Revision 1.0, March, 2003

[39] Liberty Alliance Project, "Liberty Identity Personal Profile Service Specification", Version: 1.0-18

[40] Liberty Alliance Project, "Liberty ID-FF Protocols and Schema Specification", Version 1.2-08, 11 April 2003

[41] Liberty Alliance Project, "Liberty ID-FF Bindings and Profiles Specification", Version 1.2-08, 2003

[42] Liberty Alliance Project, "Liberty Discovery Service Specification", Version: 1.0-06, 2003

[43] The Web Services Interoperability organization official website at http://www.ws-i.org/

[44] Gary Ellison, Sun Microsystems, Inc. Liberty Alliance Project> "Liberty ID-WSF Security profiles", version 1.0-08.

[45] F.Belqasmi, R.Glitho, R.Dssouli, "Using web services in telecommunications networks: An Architecture for the common functions", NOTERE 2004.