Implementation of Federated Identity in Multimedia Messaging Service using Liberty Technology

Hong Xia Shi

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

Montreal, Quebec, Canada

March 2005

Your file Votre référence
ISBN: 0-494-04398-9
Our file Notre référence
ISBN: 0-494-04398-9

ABSTRACT

Implementation of Federated Identity in Multimedia Messaging Service using Liberty Technology

Hong Xia Shi

Multimedia messaging services (MMS) enrich the user experience and create a major new source of revenue for network operators as well as for content and service providers. The capability of multimedia messaging service center (MMSC) in delivering different types of digital content, such as music, images, and video, presents a huge opportunity for its use in mobile e-commerce. But in current e-commerce, separate logins are needed to access different service providers. This limitation prevents MMS to closely cooperate with other service providers.

The Liberty Alliance standards are intentded to solve the problem of signing on repeatedly for each service provider. The Liberty Alliance defines the specifications to share identity information across service providers. However, these specifications have not been integrated into MMS.

This thesis is concerned with the design of a system wherein the MMS works with other service providers, MMS acting as Identity provider. When the user federates his/her accout in MMS with the account at any of the service providers, he/she can send a service request to a service provider (SP) without being asked for his/her password with the service provider, and at the same time can have access to the MMS from web page of that service provider.

To demonstrate the feasibility of this design, a simulation system in compliance with Liberty single sign-on and federation protocol is developed in Java using a single service provider based on Liberty identity federation framework (ID-FF).

Acknowledgements

I would like to thank my academic advisors, Dr. M.N.S, Swamy, and Dr. M. Omair, Ahmad, for their valuable guidance, regular helpful suggestions and constant support, throughout the development of this thesis.

I would like to thank the Ericsson Digital ID workgroup for directing me into this area. I would especially like to thank Todd Daley and Makan Pourzandi for their valuable advice and professional support.

I would like to thank my family members for their understanding and loving care.

Contents

List of Figures

List of Tables

ABBREVIATIONS

API	Application Program Interface
BS	Base Station
GGSN	Gateway GPRS Support Node
HLR	Home Location Register
HTTP	HyperText Transfer Protocol
ID-FF	Identity Federation Framework
IDP	Identity Provider
ISDN	Integrated Services Digital Network
J2EE	Java 2 (Platform) Enterprise Edition
LAN	Local Area Network
MMS	Multimedia Messaging Service
MMSC	Multimedia Messaging Service Center
MSISDN	Mobile Subscriber ISDN Number
PDU	Protocol Data Unit
RN	Radio Network
SAAJ	SOAP with Attachment API for Java
SAML	Security Assertion Markup Language
SGSN	Serving GPRS Support Node
SOAP	Simple Object Access Protocol
SP	Service Provider
SS7	Signaling System 7
SSL	Secure Sockets Layer
SSO	Single Sign-On
URL	Uniform Resource Locator

VAS	Value Added Service
VASP	Value Added Service Provider
VLR	Visitor Location Register
WSP	Wireless Session Protocol

Glossary

Federated Identity

Federated identity infrastructure enables cross-boundary *single sign-on*, dynamic user provisioning and identity attribute sharing. By providing for identity portability, identity federation affords end-users with increased simplicity and control over the movement of personal identity information while simultaneously enabling companies to extend their security perimeter to trusted partners.

Java 2 Platform, Enterprise Edition (J2EE)

It defines the standard for developing component-based multitier enterprise applications. Features include Web services support and development tools (SDK).

JBoss

The JBoss server is the leading Open Source, standards-compliant, J2EE based application server implemented in 100% Pure Java.

Liberty Alliance

The Liberty Alliance provides the technology, knowledge and certifications to build identity into the foundation of mobile and web-based communications and transactions. It defines specifications for Federated Identity and Web Services

Metadata

Metadata describes the attributes of an information bearing object (IBO) - document, data set, database, image, artifact, collection, etc.; metadata acts as a surrogate representation of the IBO. A metadata record can include representations of the content, context, structure, quality, provenance, condition, and other characteristics of an IBO for the purposes of representing the IBO to a potential user - for discovery, evaluation for fitness for use, access, transfer, and citation.

Mobile Station International ISDN Number (MSISDN)

The Mobile Station International ISDN Number is the standard international telephone number used to identify a given subscriber. The number is based on the ITU-T (International Telecommunications Union-Telecommunication Standardization Sector) E.164 standard.

Multimedia Messaging Service (MMS)

It is a messaging service for the mobile environment very similar to Short Message Service (SMS), or text messaging. It provides automatic, immediate delivery of personal multimedia messages from phone to phone or from phone to e-mail. In addition to the familiar text content of text messages, multimedia messages can contain images, graphics, voice, and audio clips.

Multimedia Messaging Service Center (MMSC)

It is the center that provides the MMS. Normally it contains relays, message store(s), subscriber database, etc.

MySQL

MySQL is a relational database management system, which means it stores data in separate tables rather than putting all the data in one big area. This adds flexibility, as well as speed. The SQL part of MySQL stands for "Structured Query Language," which is the most common language used to access databases.

Simple Object Access Protocol (SOAP)

The standard for web services messages. Based on XML, SOAP defines an envelope format and various rules for describing its contents. Seen (with WSDL and UDDI) as one of the three foundation standards of web services, it is the preferred protocol for exchanging web services, but by no means the only one; proponents of REST say that it adds unnecessary complexity.

Single Sign-on (SSO)

It enables a user to access multiple computer platforms or application systems after being authenticated only once.

SourceID

SourceID is an open source multi-protocol project for enabling identity federation and cross-boundary security. SourceID focuses on ease-of-integration and deployment within existing web applications, products or services. SourceID provides a high-level of developer functionality and customization and is designed to shield the integrator and enterprise from needing to understand the complexities of federation, or the rapidly evolving federation standards.

SOAP with Attachment API for Java (SAAJ)

It provides a convenient API for constructing SOAP messages without having to directly create the XML yourself.

Chapter 1 Introduction

1.1 Motivation

With the advance of radio and network technologies, the mobile system has evolved to a ubiquitous personal communication service system that provides users with new features. *Multimedia messaging service* (MMS) is a great success to enable mobile handsets to support multimedia content, such as pictures, animations, music, and video clips [1]. MMS promises a dramatic increase in messaging capabilities that will enrich the user experience and create a major new source of revenue for network operators as well as content and service providers.

At the same time, e-commerce, the whole range of business transactions over an electronic network, is growing very rapidly. With the research on the area of mobile e-commerce [2], the utilization of mobile technologies in e-commerce is also placing greater importance. In addition to buying physical goods, a considerable part of mobile e-commerce consists of the purchase of different types of digital content. Currently, MMS is the best way to deliver the digital content.

It is to think of the opportunities to introduce MMS technologies into mobile e-commerce. Using MMS in e-commerce will also enhance the features of MMS by allowing the MMS system to work with other service providers. However, there are some limitations in e-commerce transactions. The most obvious limitation is that a user has to remember many unique usernames and passwords in order to access different services [3]. Furthermore, prompting a user to login separately to closely affiliated sites creates an awkward user experience.

The limitations of e-commerce will of course affect the cooperation between the MMS system and the service providers. In order to solve this problem, the drawback of the current environment will be analysed and solutions will be proposed in the remaining of this thesis.

The previous discussion suggests that MMS is on a fast growth track. However, the co-operation between MMS operators and other service/content providers needs improvement.

Currently *multimedia messaging service center* (MMSC) has a standalone user database which has no connections to other service providers. When we introduce MMS to e-commerce, the limitations are immediately apparent. Some of the drawbacks are listed below.

- *Limitations of partners*: Currently, MMSC can only work with the content providers that do not need authentications, for example, weather broadcasting. Messages are pushed from the content providers to the MMS users. Content providers cannot yet accept requests directly from an MMS user.
- *Multiple authentications*: When a user is purchasing digital content, such as pictures, music, or videos, he/she needs at least two authentications – one from the MMSC, and the other from the service/content provider.
- *Waste of resources*: MMSC is not aware of whether or not a specific user is registered with the service/content provider. The only thing that it can do is – to always forward the requests to the service/content provider without verifying if the user is allowed to access the content or a service to that provider. In the case where the service/content provider denies the user access, thus, MMSC's forwarding work becomes a waste.

- *Redundant user profiles*: Since users have to register both at the MMSC site and with the service/content providers, the full profiles are stored in two databases. Hence, username, password, first name, last name, sex, mobile number, and home address etc, will be found in each database. In principle, some information can be shared among the service providers, for example, name and home address.
- *Redundant billings*: Users have to pay the bills from MMSC and service/content providers separately.

To overcome the above drawbacks, the limitations of e-commerce have to be eliminated or reduced as much as possible. The *Liberty Alliance* is a consortium that deals with these limitations. This alliance develops specifications for *federated identity* management. The *identity federation* establishes a standards-based mechanism of both sharing and managing identity information among organizational domains [4]. It also enables a cost-efficient means of establishing *single sign-on* to cross-company information.

1.2 Scope and Purpose of the Thesis

Although the Liberty Alliance has developed a business-ready architecture to offer more flexible and efficient identity management, its specifications have not been used in mobile networks. On the other hand, although MMS develops fast, multiple logins prevent MMS from smoothly working with other service providers.

Currently, the developments of *Liberty* specifications and MMS are on two separate tracks and they are not yet benefiting from each other's progress. This thesis aims at making the two tracks to interact by implementing a *single sign-on and identity federation protocol* into MMS, conforming the Liberty Alliance standards.

Specifically, the thesis aims at accomplishing the following:

1) Propose a prototype to implement *Liberty single sign-on and identity federation* in MMS system by

   a) specifying the relationship and communication patterns among the MMS, service providers, and subscribers,

   b) determining the protocols to be used in message format and identity management

2) Define the message flow for the prototype:

   a) Messages and authentication information should go through mobile phones, MMSC, and the service providers.

   b) Determine as to how the federation will be initiated.

3) Simulate and test the prototype elaborated in step 1):

   a) In the simulation system, Kodak is used as an example of a service provider. The system demonstrates that a picture print request is sent to Kodak via MMSC. After Kodak receives the picture data sent by a user, the picture is printed and can be mailed to the user's home address.

   b) The MMSC and Kodak simulators must be designed individually in this thesis project.

   c) In the simulation system, the *Liberty* solution will be built based on the SourceID's existing open source software.

1.3 Contribution of the Thesis

A principal contribution of the thesis is the prototype architecture for insertion of the *Liberty* technology into MMS. This implementation allows MMS to affiliate with other services. Thus, MMS users can access various service providers easily; and at the same time, MMS functionalities will get enhanced and extended.

This thesis also provides a detailed design and simulation for the system. The detailed design contains definitions of *Liberty* roles, communication patterns, message flow, and sequence diagrams.

In the simulation system, the *single sign-on and identity federation protocol* is implemented. With the support of this protocol, user can smoothly access a trusting service site by simply sending MMS messages. The simulation system shows the feasibility of this prototype.

1.4 Thesis Organization

This thesis contains five chapters. Chapter 2 introduces the background knowledge of MMS and *Liberty*. Chapter 3 presents the prototype design including architectural design, Federation flow, and Database design. In chapter 4, a simulation system is built for the proposed prototype. Chapter 5 describes the contribution of this study and gives a few suggestions for possible future investigation.

Chapter 2 Background

2.1 Multimedia Messaging Service

2.1.1 Introduction

The *multimedia messaging service* (MMS) can be regarded as an example of a range of messaging services, such as the *short message service* (SMS), and Internet mail.

Since 2002, the rollout of the service in many countries constituted the first MMS wave. MMS allowed mobile users to exchange multimedia messages with the Internet and mobile domains. Multimedia messages range from simple text messages to sophisticated messaging, for example, a slideshow composed of text, images and audio clips [5].

The future success of MMS is believed to rely on four main enablers:

- Availability and penetration of MMS phones: Mobile users require MMS-enabled phones for composing and sending multimedia messages.
- Device interoperability and service inter-working: The introduction of any new telecommunications service in a multi-vendor environment is always subject to equipment interoperability issues.
- Ease of use: The use of MMS should be very easy. No time should be required browsing through complex phone menu items.
- Added value for the end-user: The user should perceive the added value using the MMS compared to other messaging systems such as SMS or email.

2.1.2 Usage Scenarios

Person-to-person messaging

The use of MMS in the person-to-person scenarios is tightly associated with the availability of multimedia accessories such as a digital camera or a camcorder. The user usually has the possibility of sending the message to one or more recipients belonging to one of the following groups, MMS users who have an MMS phone and the corresponding service subscription, users of legacy handsets who have a legacy phone without support of MMS, and Internet users.

Content-to-person messaging

In the context of MMS, a *value-added service* (VAS) provider is an organization that offers an added-value service based on MMS. A VAS application may provide weather notifications, news updates, and so on, delivered to the phone as a multimedia message. For this purpose, the provider sets up a VAS application, which generates multimedia messages and sends them to one or multiple recipients via the MMS provider infrastructure. In many cases, the user needs to subscribe first to the value-added service in order to receive the corresponding messages. In order to operate a value-added service, the VAS provider (VASP) has to establish a service agreement with the MMS provider. The content-to-person scenario is also referred to as the machine-to-person scenario [6].

2.1.3 Service Architecture

The MMS architecture is divided into two parts. The first part is the software messaging application in the MMS phone. This application is required for the composition, sending and retrieval of multimedia messages. The second part is the external support structure. In addition, other elements in the network infrastructure are required to route messages, to adapt the content of messages to the capabilities of receiving devices, and so on. The MMS client is the software application shipped with the mobile handset. Figure 2.1 shows the general architecture of elements required for the realization of the MMS service.

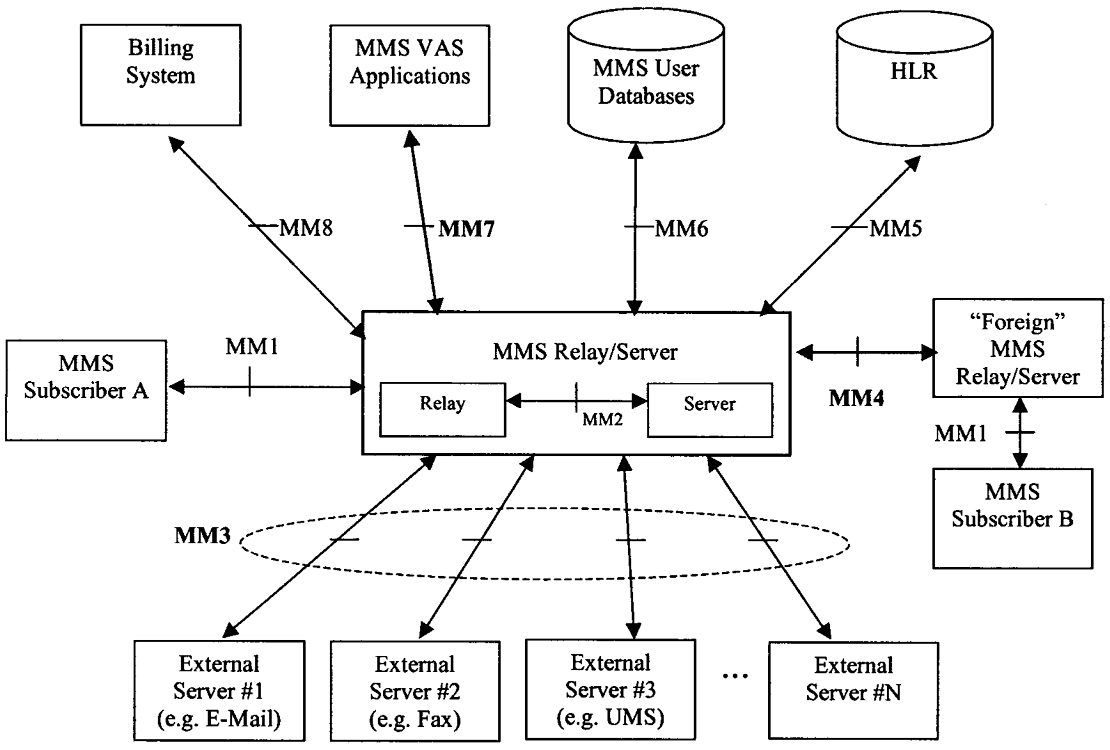


Figure 2.1 MMS architecture [7]

In an MMS environment (MMSE), network elements communicate via a set of interfaces. Each interface supports a number of transactions such as message submission, message retrieval and message forwarding. Each operation is associated with a set of protocol data units with corresponding parameters. Several interfaces have been standardized in order to ensure interoperability between devices produced by various manufacturers [6].

- The MM1 interface is a key interface in the MMS environment. It allows interactions between the MMS client, hosted in the mobile device, and the MMSC. Transactions such as message submission and message retrieval can be invoked over this interface.
- The MM2 interface is the interface between the two internal elements composing the MMSC: the MMS server and the MMS relay.
- The MM3 interface is the interface between an MMSC and external servers. Transactions invoked through this interface allow the exchange of messages between MMSC's and external servers such as email servers and SMS centers. This interface is typically based on existing IP protocols.
- The MM4 interface is the interface between two MMSC's. This interface is necessary for exchanging multimedia messages between distinct MMS environments.
- The MM5 interface enables interactions between the MMSC and other network elements. For instance, an MMSC can request routing information from the *home location register* (HLR) or from a *domain name server* (DNS).
- The MM6 interface allows interactions between the MMSC and user databases.
- The MM7 interface fits between the MMSC and external VAS applications. This interface allows a VAS application to request services from the MMSC (message submission, etc.) and to obtain messages from remote MMS clients.

- The MM8 interface enables interactions between the MMSC and a billing system.

MMSC is a key element in the MMS architecture. The MMSC is responsible for handling transactions from MMS phones and transactions from other messaging systems. The server is also in charge of temporarily storing messages that are awaiting retrieval from recipient MMS clients. Optionally, the server may also support a persistent message store where users can store messages persistently in their MM-Boxes.

The MMS offers several features for the support of person-to-person and content-to-person message scenarios. These features include sending and receiving multimedia messages, notifying a user that a message is awaiting retrieval, forwarding messages and managing a network-based box where messages can be stored over a longer term.

2.1.4 Transaction Flow

Each end-to-end feature offered by the MMS relies on a series of consecutive transactions occurring over one or more of the eight identified MMS interfaces. These transactions allow the transfer of messages and the associated reports between MMS communicating entities including network servers and mobile devices [5], [6]. Person-to-person and content-to-person scenarios are described below.

Person-to-person scenarios

The simplest transaction flow for a message exchange is the one that involves two MMS clients attached to the same MMSE. The process of exchanging a message is composed of four steps as shown in Figure 2.2 and described below.

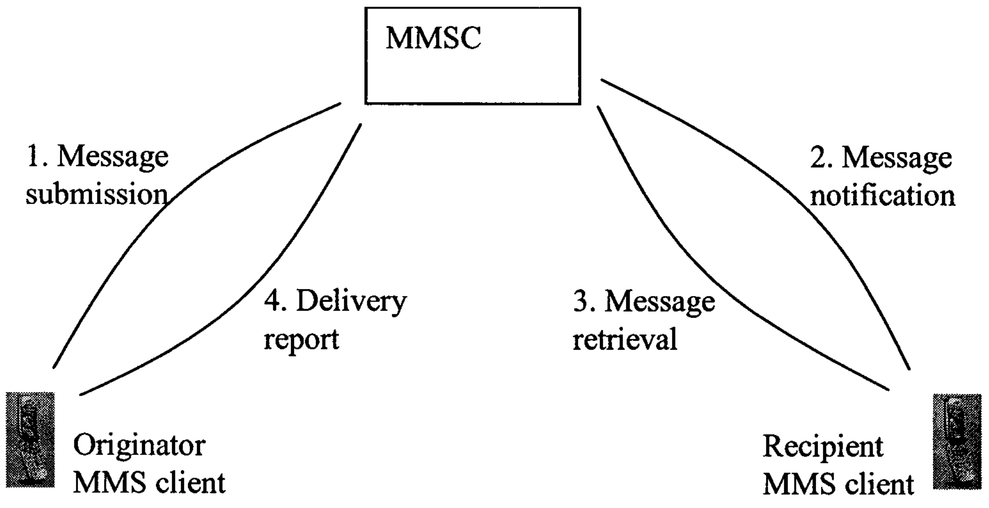


Figure 2.2 General transaction flow/Message exchange –single MMSE

1. Message submission over the originator MM1 interface: This transaction is composed of submission request, *M-send.req PDU* (Protocol Data Unit) and a corresponding submission response, *M-send.conf PDU.*
2. Message notification over the recipient MM1 interface: This transaction is composed of a notification indication *(M-notification.ind PDU)* and a notification response indication *(M-notifyresp.ind PDU).*
3. Message retrieval over the recipient MM1 interface: This transaction is composed of a retrieval request (*WSP/HTTP GET.req PDU*), retrieval response (*M-retrieve.conf PDU*) and optionally a retrieval acknowledgement indication (*M-acknowledge.ind PDU*).
4. Delivery reporting over the originator MM1 interface: A delivery report is conveyed over the MM1 interface only if the generation of a delivery report was requested during message submission and if the recipient did not deny the

generation of such a report. The transaction is composed of a delivery-report indication (*M-delivery.ind PDU*).

The exchange of a message between MMS clients attached to distinct MMSEs involves two MMSCs: the originator MMSC and the recipient MMSC. The two MMSCs are interconnected via the MM4 interface. The process of exchanging a message is composed of four to six steps as shown in Figure 2.3.

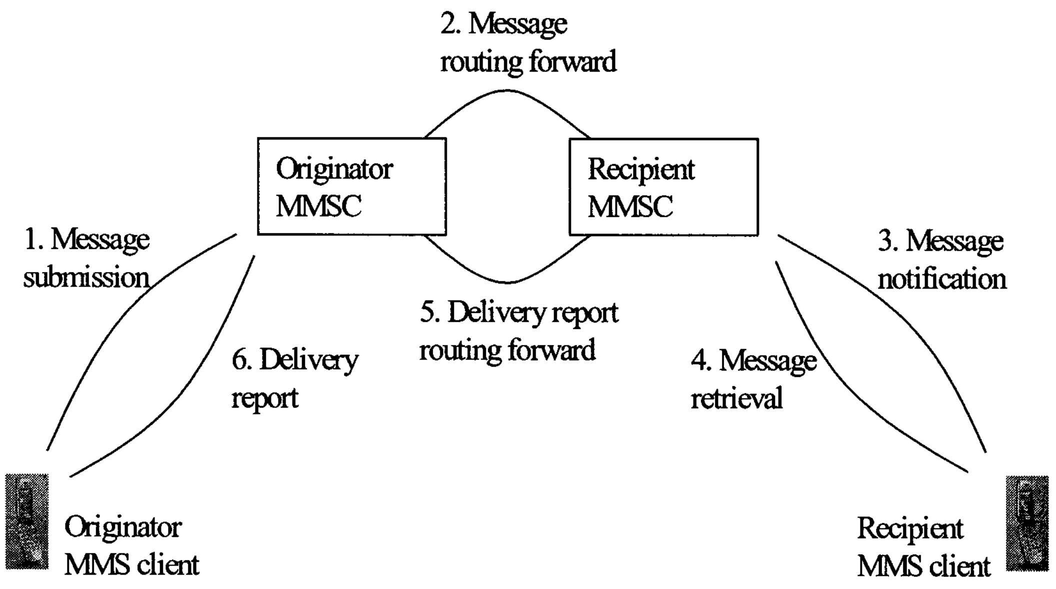


Figure 2.3 General transaction flow/Message exchange – inter-MMSE

Step 2 and step 5 are in addition to single MMSE case. Step 2 (routing forward the message over the MM4 interface) is composed of a forward request (*MM4_forward.req PDU*) and a corresponding forward response (*MM4_forward.res PDU*). Step 5 (routing forward the delivery report over the MM4 interface) is composed of a submission request (*MM4_delivery_report.req PDU*) and a corresponding submission response (*MM4_delivery_report.res PDU*). The delivery report is generated by the recipient MMSC upon confirmation of message retrieval by the recipient MMS client.

Content-to-person scenarios

Content-to-person refers to the scenario where the message originates from a VAS application and is delivered to one or more MMS recipients. For this purpose, the *VAS provider* (VASP) operates a VAS application, usually Internet hosted, and connected to an MMSC via the MM7 interface. In this configuration, the MMSC interacts with recipient MMS clients over the MM1 interface as already described previously.

Figure 2.4 shows interactions among a VAS application, the MMSC and one recipient MMS client for the exchange of a message from the VAS application down to the recipient MMS client.

1. Message submission over the MM7 interface. The transaction is composed of a submission request (*MM7_submit.req PDU*) and a corresponding submission response (*MM7_submit.res PDU*).
2. Message notification over the MM1 interface
3. Message retrieval over the MM1 interface
4. Delivery reporting over the originator MM7 interface: The transaction is composed of a delivery-report request (MM7_delivery_report.req PDU) and a corresponding delivery-report response (MM7_delivery_report.res PDU).
5. Read reporting over the MM1 interface
6. Read reporting over the MM7 interface: The transaction is composed of a read-report request (*MM7_read_reply_report.req PDU*) and a corresponding read-report response (*MM7_read_reply_report.res PDU*).

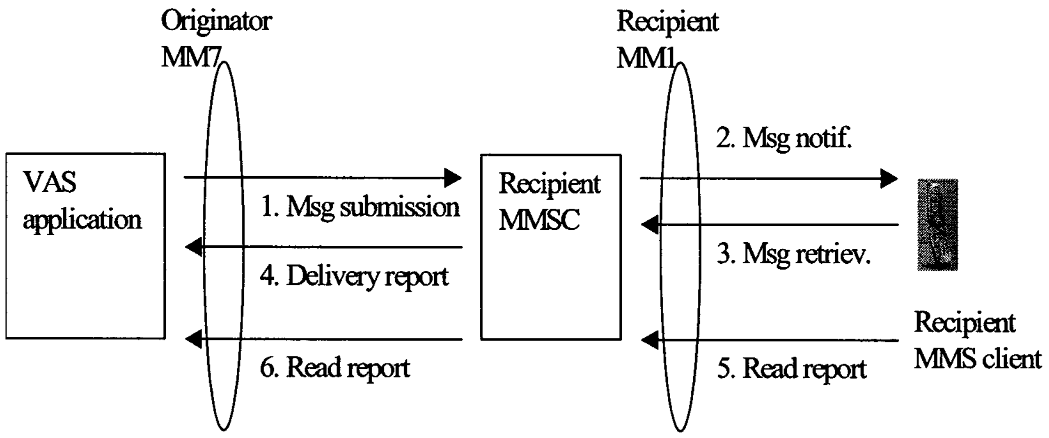


Figure 2.4 General transaction flow/Message exchange – VASP-mobile

In the reverse direction, the MMS client also has the possibility of submitting a message addressed to a VAS application. This may be reply related to a previous message originated by the VAS application or it can be a new unrelated message. Figure 2.5 shows interactions between the MMS client, the MMSC and VAS application for the exchange of a message from the MMS client up to the VAS application.

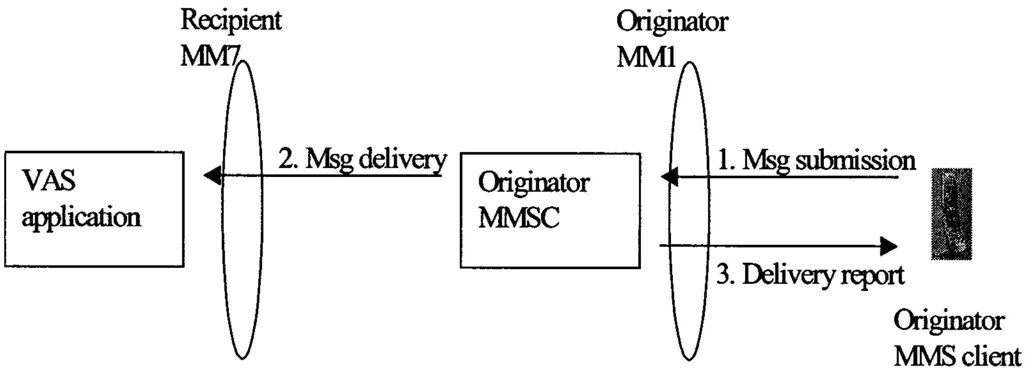


Figure 2.5 General transaction flow/Message exchange – mobile-VASP

1. Message submission over the originator MM1 interface

2. Message delivery over the MM7 interface. This transaction is composed of a delivery request (*MM7_deliver.req PDU*) and a corresponding delivery response (*MM7_deliver.res PDU*).
3. Delivery reporting over the MM1 interface.

2.1.5 Interface Description

As described in Figure 2.1, there are eight interfaces, MM1 through MM8, in MMS. In this thesis, we only introduce MM1 and MM7 interfaces which are closely related to the proposed system. The description of other interfaces, can be found in [5], [6], and [7].

MM1 interface, MMS client – MMSC

The MM1 interface allows interactions between the MMS client and the MMSC. Several primitives, also known as *protocol data units* (PDUs), can be invoked over this interface for transactions such as message notification, message submission, message retrieval, and so on. Three types of PDUs can be exchanged via the MM1 interface.

- *Request*: A request PDU is invoked from an MMS entity (MMS client or MMSC) to request a service to be provided by another MMS entity (MMS client or MMSC). In the context, the serving MMS entity, which accepts or rejects the request, notifies the requesting MMS entity of the request status with a confirmation PDU. A request PDU is often marked with a transaction identifier. The name of a request PDU is suffixed with ".req" (e.g. M-send.req)

- *Confirmation/response*: A confirmation PDU is invoked by serving MMS entity to confirm the status of the corresponding request PDU. The name of a confirmation PDU is suffixed with ".conf" (e.g. M-send.conf)
- *Indication*: An indication PDU is invoked by an MMS entity to notify another MMS entity of the occurrence of an event (message notification, reports, etc.). The name of an indication PDU is suffixed with ".ind" (e.g. M-notification.ind). An indication is not confirmed.

Descriptions of transactions/PDUs over MM1 interface can be found in [5].

MM7 interface, MMSC – VAS application

The MM7 interface enables interactions between VAS applications and an MMSC.

MM7 interface is based on the *simple object access protocol* (SOAP) with HTTP at the transport layer.

SOAP is a lightweight protocol for the exchange of information in distributed environments such as the MMSE. All SOAP messages are represented using XML. The SOAP specifications consist of three distinct parts:

- Envelope: This part defines a framework for describing the content of a SOAP message and how to process it.
- Set of encoding rules: Encoding rules are used for expressing instances of application-defined data types.
- Convention for representing remote procedure calls: This convention helps entities in a distributed environment to request services from each other in an inter-operable manner.

SOAP may be used over a variety of transport protocols. In the MMSE, SOAP is used over the HTTP transport protocol for the realization of the MM7 interface. With this configuration, MM7 request messages are transferred in *HTTP Post* requests, whereas the corresponding MM7 response messages are transferred as part of *HTTP Response* messages.

A SOAP message, represented using XML, as shown in Figure 2.6, consists of a *SOAP envelope*, a *SOAP header*, a *SOAP body* and an optional *SOAP attachment.* For messages containing a *SOAP envelope* only, the media type text/xml is used. If the *SOAP message* also contains an attachment, then the media type multipart/related is used and the *SOAP envelope* is identified with the *Start parameter* of content type. Each part of the SOAP message has, at least, the two parameters *Content-Type* and *Content-ID.*

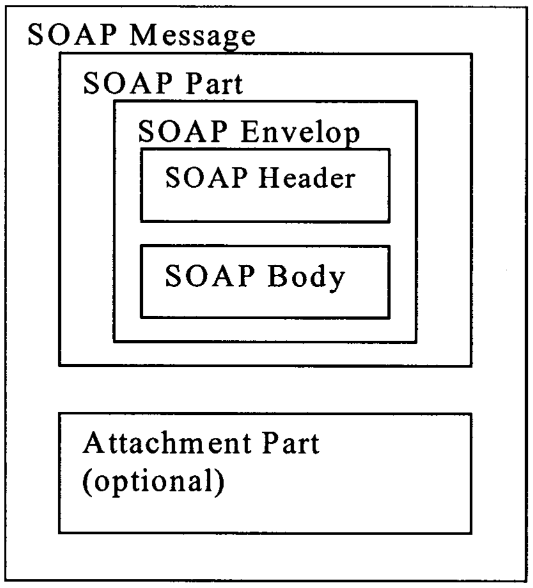


Figure 2.6 Structure of a SOAP message

The *SOAP envelope* is the first element to appear in *HTTP Post* requests and in the corresponding responses. The *SOAPAction* parameter is set to the “Null string”. The MMSC or the VAS server is identified uniquely with a URL placed in the host header field of the *HTTP Post* method [8], [9].

2.2 Liberty Alliance

2.2.1 Introduction

From the moment a person is born, he/she has an “identity.” The identity starts with their name on a birth certificate and evolves over time as labels, interactions, and relationships are associated with that person. As people grow, they interact with an ever-larger group of individuals and organizations.

On reaching adulthood, pieces of their identities are scattered across a range of entities: banks, credit card companies, national IDs, and the places where they work, to name just a few. The Internet has become one of the prime vehicles for business, community and personal interactions, and it is fragmenting this identity even further. Pieces of their identity are doled out across the many computer systems and networks used by employers, Internet service providers, instant messaging applications, and online business and content providers, as shown in Figure 2.7. All this occurs with little coordination, interaction, or control on the part of the individual.

People have to repeatedly enter the same information within the workplace and in personal business dealings. Everyone concerned may also have to deal with identity abuse [10].

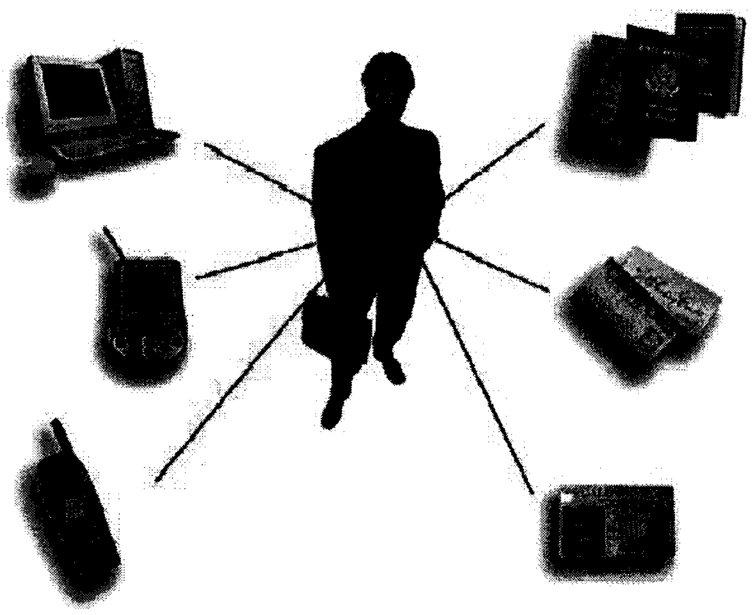


Figure 2.7 The various forms of identification [11]

The *Liberty Alliance Project* is an attempt to deal with these problems. The Liberty Alliance is non-profit and government organization. The consortium is committed to developing an open standard for federated network identity that supports all current and emerging network devices. Federated identity offers businesses, governments, employees and consumers a more convenient and secure way to control identity information in the digital economy, and is a key component to drive the use of e-commerce, personalized data services, as well as web-based services. Membership is open to all commercial and non-commercial organizations [11].

2.2.2 The Need for Federated Network Identity

To address the inefficiencies and complications of network identity management for businesses and consumers, there is a strong need for *a* federated network identity infrastructure that would allow users to “link” elements of their identity between accounts without centrally storing all of their personal information.

There are many benefits to a federated network identity infrastructure. Some of the main ones are as follows.

- It provides the end user a far more satisfactory online experience, as well as new levels of personalization, security, and control over identity information.
- It enables the IT manager to more easily and securely register accounts and provide access to the right resources.
- It enables businesses to create new relationships with one another and to realize business objectives faster, more securely, and at a lower cost.

The Liberty Alliance’s vision is one of a networked world in which individuals and businesses can more easily interact with one another, while respecting the privacy and security of shared identity information.

2.2.3 The Liberty Actors

Liberty defines three actors: *principals*, *service providers*, and *identity providers*:

- A *principal* is a user.
- A *service provider* (SP) is an organization offering Web-based services to *principals*. It can be car rental or airline, etc.
- An *identity provider* (IDP) is a service provider offering additional services and incentives so that other service providers affiliate with the *identity provider* and *principals* choose to use the service provider as their *identity provider*. The important service an *Identity provider* gives is authentication of the *principal*.

2.2.4 Liberty Protocols

A set of protocols collectively provides a solution for identity federation management, cross-domain authentication, and session management. The *Liberty* protocol suite consists of the following protocols [12]:

- *Single sign-on and federation*: The protocol by which identities are federated and by which *single sign-on* occurs.
- *Name registration*: The protocol by which a provider can register an alternative opaque handle (or name identifier) for a *principal.*
- *Federation termination Notification*: The protocol by which a provider can notify another provider that a particular identity federation has been terminated (also known as de-federation).
- *Single logout*: The protocol by which providers notify each other of logout events.
- *Name identifier mapping*: The protocol by which service providers can obtain (often encrypted) name identifiers corresponding to an identity federation in which they do not participate.

Single sign-on and federation is the main protocol used for this thesis. It will be broken down below. For details on the other protocols, please refer to [5], [12], and [13].

Single sign-on and federation protocol

The *single sign-on and federation* protocol defines a request and response protocol by which *single sign-on and identity federation* occurs. The protocol is conducted between a service provider and one or more identity providers. The protocol works as follows:

1. This step is optional. A service provider issues an `<AuthnRequest>` to an identity provider, instructing the identity provider to provide an authentication assertion to the service provider. Optionally, the service provider may request that the identity be federated.

2. The identity provider responds with either an `<AuthnResponse>` containing authentication assertions to the service provider or an artifact that can be de-referenced into an authentication assertion. In addition, the identity provider potentially federates the *principal*'s identity at the identity provider with the *principal*'s identity at the service provider.

Note that under certain conditions, an identity provider may unilaterally (without receiving an authentication request) issue an authentication response to a service provider.

The identity provider may be proxying for an *authenticating identity provider*, in which case, this protocol may be repeated between the recipient of the original `<AuthnRequest>`, and other identity providers.

The service provider issues an initial `<AuthnRequest>` to an identity provider. A set of parameters is included in the request that allows the service provider to specify desired behavior at the identity providers in processing the request. A requester can control the following identity provider behaviors:

- Prompt the principal for credentials if the principal is not presently authenticated.
- Prompt the principal for credentials, even if the principal is presently authenticated.

- Federate the principal's identity at the identity provider with the principal's identity at the requester.
- Issue an anonymous and temporary identifier for the principal to the service provider.
- Use a specific protocol profile in responding to the request.
- Use a specific authentication context (for example, smartcard-based authentication vs. username/password-based authentication).
- Restrict the ability of the recipient to proxy the authentication request to additional identity providers.

As mentioned above, the response is either an `<AuthnResponse>` element containing a set of authentication assertions or a set of artifacts the service provider can dereference into a set of authentication assertions.

2.2.5 Liberty Profiles

This section defines the *Liberty* profiles for the use of request and response messages given in [12]. The combination of message content specification and message transport mechanisms for a single client type (that is, user agent) is termed a *Liberty profile*. The profiles are grouped into categories according to the protocol message intent.

The following profile categories are defined in by Liberty Alliance [13]:

- *Single sign-on and federation*: The profiles by which a service provider obtains an authentication assertion from an identity provider facilitating *single sign-on and identity federation*.

- *Name registration*: The profiles by which service providers and identity providers specify the name identifier to be used when communicating with each other about the principal.
- *Federation termination notification*: The profiles by which service providers and identity providers are notified of federation termination.
- *Single logout*: The profiles by which service providers and identity providers are notified of authenticated session termination.
- *Identity provider introduction*: The profile by which a service provider discovers which identity providers a *principal* may be using.
- *Name identifier mapping*: The profile by which a service provider may obtain a NameIdentifier with which to refer to a principal at a SAML Authority.
- *Name identifier encryption*: The profile by which one provider may encrypt a NameIdentifier to permit it to pass through a third-party without revealing the actual value until received by the intended provider.

Basic single sign-on and federation profile

Since *single sign-on and federation* is the most important protocol in this thesis, we focus on *single sign-on and federation* profile.

This section defines the profiles by which a service provider obtains an authentication assertion of a user agent from an identity provider to facilitate *single sign-on*. In addition, the *single sign-on* profiles can be used as a means of federating an identity from a service provider to an identity provider through the use of the `<NameIDPolicy>` element in the `<lib:AuthnRequest>` protocol message as specified in [12].

The *single sign-on* profiles make use of the following metadata [14] elements:

- *ProviderID:* Used to uniquely identify the service provider to the identity provider and is documented in these profiles as "service provider ID."
- *AffiliationID:* Used to uniquely identify an affiliation group to the identity provider and is documented in these profiles as "affiliation ID."
- *SingleSignOnServiceUR*L: The URL at the identity provider that the service provider should use when sending *single sign-on* and federation requests. It is documented in these profiles as "*single sign-on* service URL."
- *AssertionConsumerServiceUR*L: The URL(s) at the service provider that an identity provider should use when sending *single sign-on* or federation responses. It is documented in these profiles as "assertion consumer service URL."
- *SOAPEndpoint:* The SOAP endpoint location at the service provider or identity provider to which *Liberty* SOAP messages are sent.

All *single sign-on* profiles can be described by one interaction diagram, provided that different message types are optional in different profiles and that the actual content of the messages may differ. Where interactions and messages differ or are optional, they are designated and detailed within the specific *single sign-on* profiles. Figure 2.8 represents the basic template of interactions for achieving *single sign-on*. This should be used as the baseline for all *single sign-on* profiles.

In Figure 2.8, Steps 1 through 5 can be considered typical but optional. An identity provider may initiate a SSO profile by unilaterally creating a `<lib:AuthnResponse>` or artifact, and then proceeding with Step 6, as discussed in [13].

It should be noted that multiple identity providers may be involved in the authentication of the *principal*. Although a single identity provider is depicted in the profiles, that identity provider may interact with other identity providers to authenticate the *principal* using the proxying method described in [13] and [15], and the profiles as noted below. In such situations, these profiles would be used by the identity provider originally contacted by the requesting service provider to communicate with other identity providers.

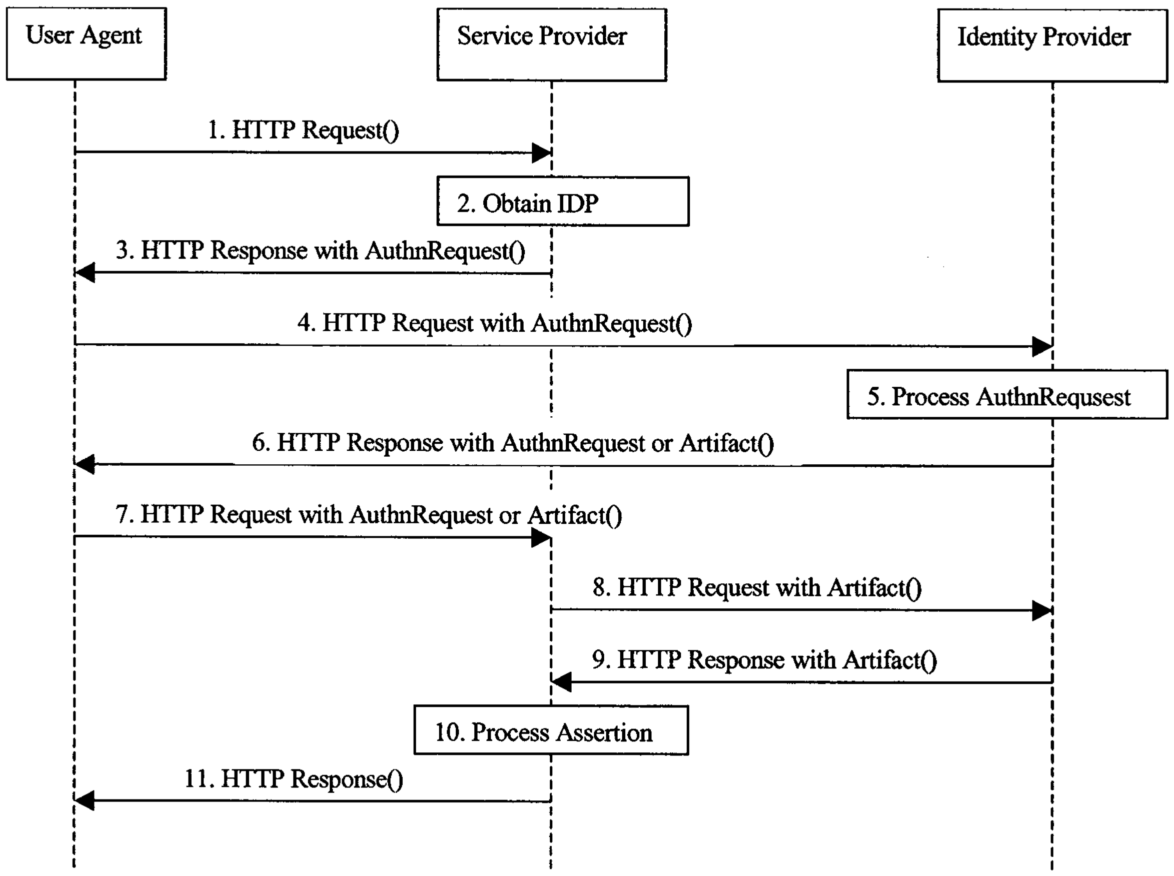

User Agent
Service Provider
Identity Provider
1. HTTP Request()
2. Obtain IDP
3. HTTP Response with AuthnRequest()
4. HTTP Request with AuthnRequest()
5. Process AuthnRequsest
6. HTTP Response with AuthnRequest or Artifact()
7. HTTP Request with AuthnRequest or Artifact()
8. HTTP Request with Artifact()
9. HTTP Response with Artifact()
10. Process Assertion
11. HTTP Response()

Figure 2.8 Basic single sign-on profile

Step 1: HTTP request

In this step, the user agent accesses the intersite transfer service at the service provider with information about the desired target attached to the URL. Typically, access to the intersite transfer service occurs via a redirection by the service provider in response to a user agent request for a restricted resource.

It is recommended that the HTTP request be made over either SSL 3.0 [16] or TSL 1.0 [17] to maintain confidentiality and message integrity in this first step.

Step 2: Obtain identity provider

In this step, the service provider obtains the address of the appropriate identity provider to whom the user agent will be redirected in step 3. The means by which the identity provider address is obtained is implementation-dependent and up to the service provider. The service provider may use the *Liberty* identity provider introduction profile in this step.

Step 3: HTTP response with <AuthnRequest>

In this step, the service provider's intersite transfer service responds and sends the user agent to the *single sign-on* service URL at the identity provider. The form and contents of the HTTP response in this step are profile-dependent.

Step 4: HTTP request with <AuthnRequest>

In this step, the user agent accesses the identity provider's *single sign-on* service URL with the `<lib:AuthnRequest>` information. This request may be a GET or POST request; providers must support both methods. As described later, such a POST must contain an `LAREQ` form element containing the XML protocol request in base64-encoded format.

Step 5: Processing <AuthnRequest>

In this step, the identity provider must process the `<lib:AuthnRequest>` message according to the rules specified in [12].

If the *principal* has not yet been authenticated with the identity provider, authentication at the identity provider may occur in this step. The identity provider may obtain consent from the principal for federation, or otherwise consult the principal. To this end, the identify provider may return to the HTTP request any HTTP response. Including but not limited to HTTP authentication, HTTP redirect, or content. The identity provider should respect the HTTP User-Agent and Accept headers and should avoid responding with content-types that the User-Agent may not be able to accept. Authentication of the principal by the identity provider is dependent upon the `<lib:AuthnRequest>` message content.

In case the identity provider responds to the user agent with a form, it is recommended that the `<input>` parameters of the form be named whenever possible.

Step 6: HTTP response with <AuthnResponse> or artifact

The identity provider must respond to the user agent with a `<lib:AuthnResponse>`, a SAML artifact, or an error. The form and contents of the HTTP response in this step are profile-dependent.

Step 7: HTTP request with <AuthnResponse> or artifact

The user agent accesses the assertion consumer service URL at the service provider with a `<lib:AuthnResponse>` or a SAML artifact. This request may be a GET or POST request; providers must support both methods. As described later, such a POST must contain an LARES form element containing the XML protocol request or artifact in base64-encoded format.

Step 8: HTTP request with artifact

This step is required only for *single sign-on* profiles that use a SAML artifact.

In this step the service provider, in effect, dereferences the single SAML artifact in its possession to acquire the authentication assertion that corresponds to the artifact.

The service provider must send a `<samlp:Request>` SOAP message to the identity provider's SOAP endpoint, requesting the assertion by supplying the SAML assertion artifact in the `<samlp:AssertionArtifact>` element.

The service provider must provide a mechanism for the identity provider to authenticate the service provider.

Step 9: HTTP response with assertion

This step is required only for *single sign-on* profiles that use a SAML artifact. In this step if the identity provider is able to find or construct the requested assertion, it responds with a `<samlp:Response>` SOAP message containing the requested `<saml:Assertion>`. Otherwise, it returns an appropriate status code, as defined within the SOAP binding for SAML in [15].

Step 10: Process assertion

The service provider processes the `<saml:Assertion>` returned in the `<samlp:Response>` or `<lib:AuthnResponse>` protocol message to determine its validity and how to respond to the *principal*'s original request. The signature on the `<saml:Assertion>` must be verified.

The service provider processing of the assertion must adhere to the rules defined in [15] for messages such as assertion `<saml:Conditions>` and `<saml:Advice>`.

The service provider may obtain authentication context information for the *principal*'s current session from the `<lib:AuthnContext>` element contained in `<saml:Advice>`. Similarly, the information in the `<lib:RelayState>` element may be obtained and used in further processing of the request, by the service provider.

Step 11: HTTP response

In this step, the user agent is sent an HTTP response that either allows or denies access to the originally requested resource.

2.3 SourceID Liberty 2.0

SourceID is an open source project for federated identity. Sponsored by Ping Identity [18], SourceID provides developers and enterprises with a complete suite of identity federation libraries for SAML, *Liberty*, and WS-Federation. SourceID toolkits focus on ease-of-integration and deployment within existing web applications, products or services.

SourceID's mission is to provide open source tools, applications, and infrastructure for Federated Identity Management. SourceID uses open protocols and standards such as those developed by the Liberty Alliance and other project groups [19].

SourceID Liberty 2.0 is an open source stand-alone identity federation server -- enabling *Liberty Identity Federation Framework* (ID-FF) 1.2 while focusing on extendability and flexibility. SourceID Liberty 2.0's core is a workflow engine. It is a Java application which was developed on the JBoss application server.

2.3.1 Workflow and Open Business Engine

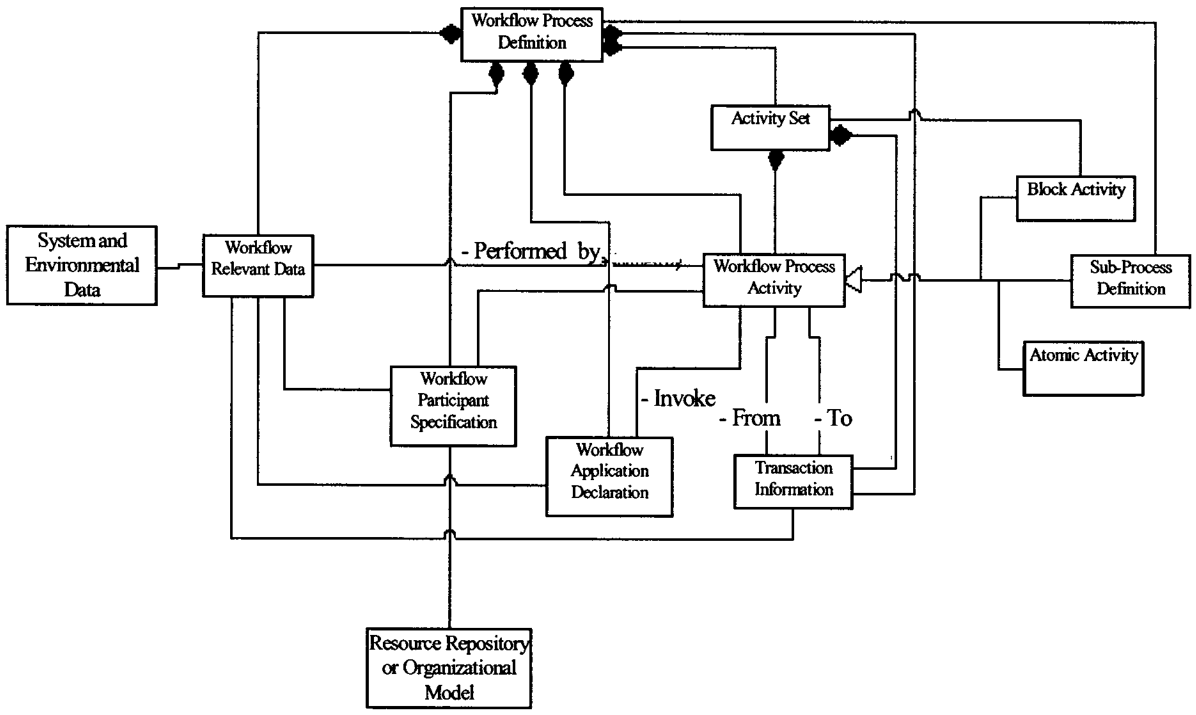
Workflow is the automation of a business process, in whole or in part, during which documents, information or tasks are passed from one participant to another for action, according to a set of procedural rules [20].

Business process is a set of one or more linked procedures or activities that collectively realize a business objective or policy goal, normally within the context of an organizational structure defining functional roles and relationships. It has defined conditions triggering its initiation in each new instance (e.g. the arrival of a claim) and defined outputs at its completion.

The automation of a business process is defined within a *Process Definition*, which identifies the various process activities, procedural rules and associated control data used to manage the workflow during the process enactment.

Workflow management coalition (WfMC) is a nonprofit, Brussels-based standards organization focused on defining workflow standards. The WfMC has described the process definition and the interchange of process definitions [21].

The *workflow process definition* provides contextual information that applies to other entities within the process. It is a container for the process itself and provides information associated with administration or to be used during process execution.



Workflow Process Definition
Activity Set
Block Activity
System and Environmental Data
Workflow Relevant Data
- Performed by
Workflow Process Activity
Sub-Process Definition
Atomic Activity
Workflow Participant Specification
- Invoke
- From
- To
Workflow Application Declaration
Transaction Information
Resource Repository or Organizational Model

Figure 2.9 Entities in workflow process definition [22]

According to the WfMC's description, the *workflow process definition* is shown in Figure 2.9. Figure 2.9 shows the relationships between the basic entities and attributes for the exchange of *process definitions*. For a process definition the following entities must be defined [22], [23]:

Workflow process activity: A process definition consists of one or more activities, each comprising a logical, self-contained unit of work within the process. An activity represents work, which will be processed by a combination of resource and/or computer applications.

Transition information: Activities are related to one another via flow control conditions (transition information). Each individual transition has three elementary properties, the from-activity, the to-activity and the condition under which the transition is made. Transition from one activity to another may be conditional (involving expressions which are evaluated to permit or inhibit the transition) or unconditional. The transitions within a process may result in a sequential or parallel operation of individual activities within the process.

Workflow participant specification: This provides descriptions of resources that can act as the performer of the various activities in the process definition. The particular resources, which can be assigned to perform a specific activity, are specified as an attribute of the activity, participant assignment, that links the activity to the set of resources (within the workflow participant declaration) which may be allocated to it.

Workflow application declaration: This provides descriptions of the IT applications or interfaces which may be invoked by the workflow service to support, or wholly automate, the processing associated with each activity, and identified within the activity by an application assignment attribute (or attributes).

Workflow relevant data: This defines the data that is created and used within each process instance during process execution. The data is made available to activities or applications executed during the workflow and may be used to pass persistent information or intermediate results between activities and/or for evaluation in conditional expressions such as in transitions or participant assignment.

As indicated in Figure 2.9, the process model includes various entities whose scope may be wider than a single process definition. In particular the definitions of participants, applications and workflow relevant data may be referenced from a number of process definitions. The meta-model assumes the use of a common process definition repository, associated with the workflow management system, to hold the various entity types comprising the process definition. Within the repository itself and to support the efficient transfer of process definition data to/from the repository, the concept of a *package* is introduced, which acts as a container for the grouping of common data entities from a number of different process definitions, to avoid redefinition within each individual process definition.

The package provides a container to hold a number of common attributes from the workflow process definition entity (author, version, status, etc.). Each process definition contained within the package will automatically inherit any common attributes from the package, unless they are separately re-specified locally within the process definition.

Within a *package*, the scope of the definitions of some entities is global and these entities can be referenced from all workflow process definitions (and associated activities and transitions) contained within the package. Those entities are *workflow participant specification, workflow application declaration*, and *workflow relevant data*. The package definition model is illustrated in Figure 2.10.

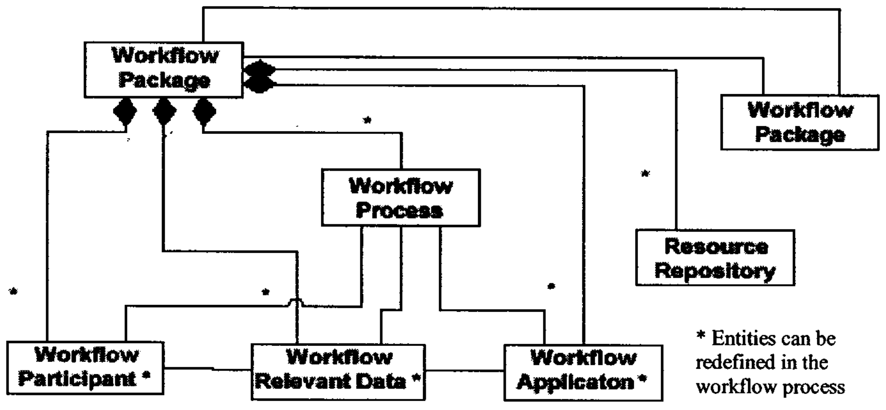


Figure 2.10 Package definition [22]

The *open business engine* (OBE) is a Java workflow engine implementing WfMC. It is designed to act as a general-purpose application controller. Highly modular and configurable, it suits J2EE or embedded deployment.

All changes to a process or activity persist in real-time. Therefore, the OBE engine does not run in a thread, it is simply a group of APIs and common objects that handle the flow. When a change to the workflow is made, the engine then processes that change. When finished, the engine returns. OBE workflow engine uses XPDL as its process definition language.

The OBE workflow engine currently implements *NO, ROUTE, TOOL:Procedure*, and *SUB-FLOW* type activities [23].

NO: Just as is says, no activity. This describes a 'manual' activity.

ROUTE: A route activity is used for simply routing to other activities via the transitions.

TOOL:

Application: An external application is invoked. - *Not currently implemented.*

Procedure: OBE implements procedures as internal service calls.

SUB-FLOW: Sub-flows can be defined as synchronous or asynchronous.

2.3.2 System Architecture

SourceID Liberty 2.0 is a Java implementation of the Liberty Alliance ID-FF 1.2 protocols. It has a workflow-based architecture that provides a great deal of flexibility and extensibility. A server-based application, it focuses on making it possible to seamlessly adapt the Liberty 1.2 federation protocols to existing infrastructure. From a developer and deployment standpoint, the SourceID Liberty 2.0 adapter tier is a critical aspect of the architecture.

The adapter tier is a set of Java interfaces (see Figure 2.11) that allow developers to customize how data is stored and how interactions with the web application occur. Developers implement these interfaces in order to integrated SourceID with their application environment [24].

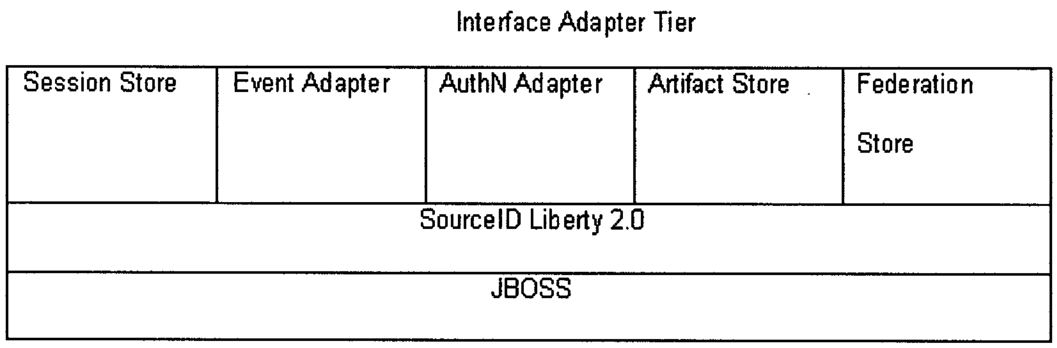


Figure 2.11 SourceID Liberty 2.0 B high level architecture

Session Store: It is a mechanism that the SourceID implementation uses to track which users have logged in and logged out. An in-memory implementation is provided in SourceID Liberty 2.0 (org.sourceid.idff12.adapters.impl.SimpleSessionStore). No additional configureation is required to use this default adapter.

Event Adapter: It is a notification mechanism that is used to update the local session system when a SSO event occurs. Separate adaptor instances must be created for handling IdP and/or SP side behaviour. An implementation of this adaptor must be provided for a specific deployment (no default implementation is provided).

AuthN Adapter: SourceID uses this interface to retrieve the session identifier provided in a previous call to the on SessionCreated method on the EventAdapter interface. The session identifier is used by SourceID to track state information about a user's current session so that functionality such as *single log out* works correctly. Separate adaptor instances must be created for handling IdP and/or SP side behaviour. An implementation of this adaptor must be provided for a specific deployment (no default implementation is provided).

Artifact Store: It supports the artifact profile defined in the *Liberty* specification by keeping track of an associated array of artifacts to assertions. An in-memory implementation is provided with SourceID Liberty 2.0 (org.sourceid.idff12.adapters.impl.SimpleArtifactSotre). No additional configuration is required to use this default adapter.

Federation Store: It keeps track of information about account linkages. Hides all implementation details of mapping user account identifiers to pseudonyms. An in-memory implementation is provided with SourceID Liberty 2.0 (org.sourceid.idff12.adapters.imple.SimpleFederationStore). No additional configuration is required to use this default adapter.

The package of SourceID Liberty 2.0 is composed of 13 workflow processes shown in Figure 2.12 and described below.

sp-authn: SP SSO/Fed. It handles *single sign-on and federation* on *service provider.*

idp-authn: IDP SSO/Fed. It handles *single sign-on and federation* on *Identity provider.*

idp-artifact: IDP Artifact. This process handles the artifacts on *Identity provider.*

idp-slo: IDP SLO. This process handles *single logout* on *Identity provider.*

idp-slo-return-http-get: IDP SLO with HTTP Get. It handles Single Log-out by accessing *Identity provider*'s return URL in GET.

sp-slo-http: SP HTTP SLO. It handles HTTP based *single logout* on *service provider.*

sp-slo-soap: SP SOAP SLO. It handles SOAP based *single logout* on *service provider.*

sp-slo-init: SP SLO Initiator. it initiates the *single logout* on *service provider.*

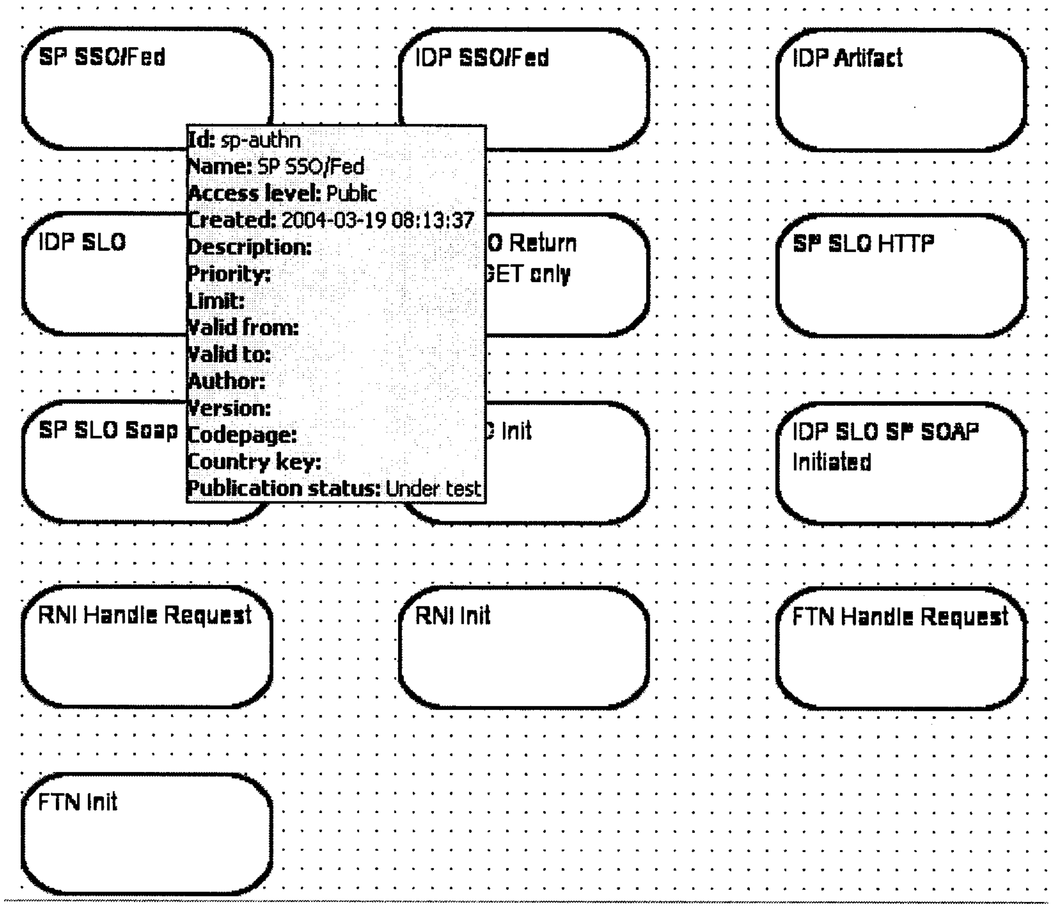
idp-slo-sp-init-soap: IDP SOAP SLO Initiator. This process initiates SOAP based *single logout* on *identity provider.*

rni-handle-request: RNI Request Handler. This process handles the *redirection* request.

rni-init: RNI Initiator. This process initiates the *redirection*.

ftn-handle-request: FTN Request Handler. it handles *federation termination* request.

ftn-init: FTN Initiator. This process initiates the *federation termination*.



Id: sp-authn
Name: SP SSO/Fed
Access level: Public
Created: 2004-03-19 08:13:37
Description:
Priority:
Limit:
Valid from:
Valid to:
Author:
Version:
Codepage:
Country key:
Publication status: Under test

Figure 2.12 Workflow processes in SourceID Liberty 2.0

The graphical and XPDL description of each process can be found in the file idff12.xpdl.

To look more details into the workflow, process SP SSO/Fed is taken as the example. This process is illustrated in Figure 2.13.

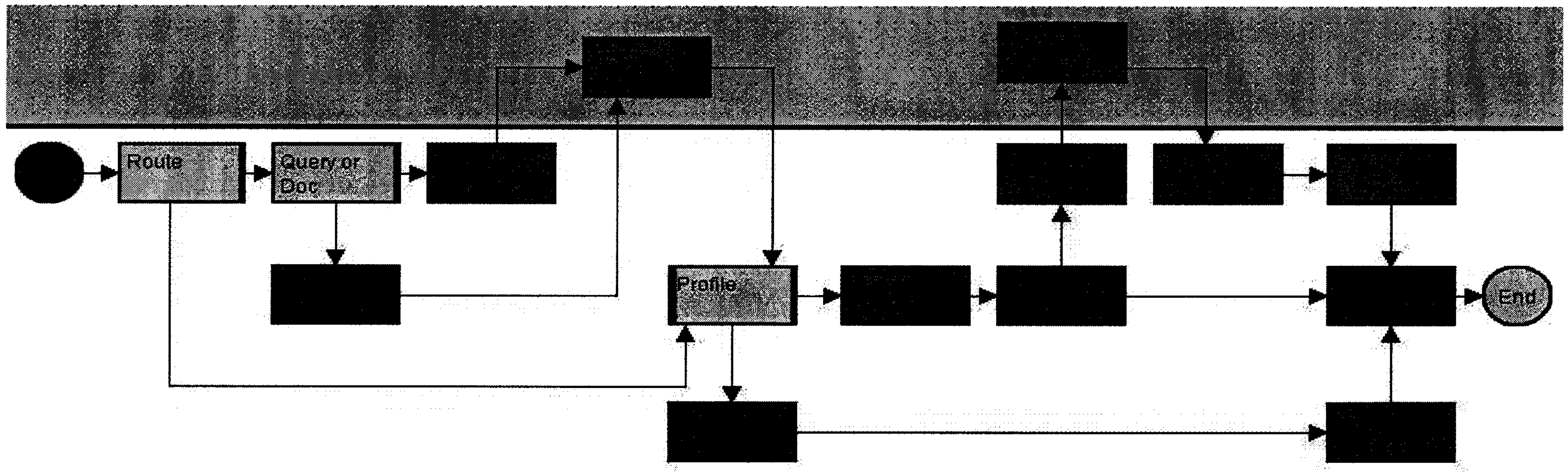
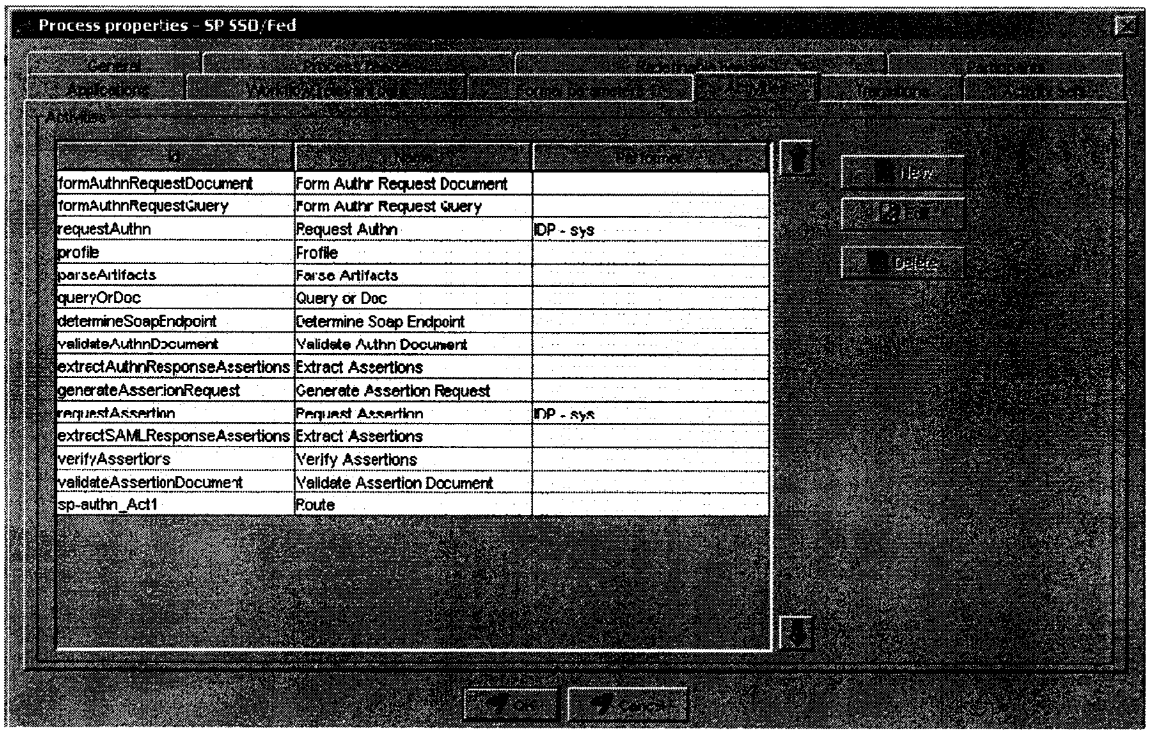


Figure 2.13 The process of SP SSO/Fed

A workflow editor can provide more information on these activities and applications. Figure 2.14 illustrates the layout of workflow editor.



Id	Name	Performer
formAuthnRequestDocument	Form Authr Request Document	
formAuthnRequestGuery	Form Authr Request Guery	
requestAuthn	Request Authn	IDP - sys
profile	Frofile	
parseArtifacts	Farse Artifacts	
queryOrDoc	Query or Doc	
determineSoapEndpoint	Determine Soap Endpoint	
validateAuthnDocument	Validate Authn Document	
extractAuthnResponseAssertions	Extract Assertions	
generateAsserionRequest	Generate Assertion Request	
requestAssertion	Request Assertion	IDP - sys
extractSAMLResponseAssertions	Extract Assertions	
verifyAssertiors	Verify Assertions	
validateAssertionDocument	Validate Assertion Document	
sp-authn_Act1	Route	

Figure 2.14 Breaking down a process definition in Workflow Editor

2.4 Conclusion

In this chapter, the background information about MMS and the *Liberty* standard has been introduced. MMS has been running commercially for a couple of years; therefore, its standards are quite well integrated and developed. Among the interfaces in MMS, MM7 between the MMSC and the content providers is the most important for this thesis. The Liberty Alliance defines specifications for the protocols, such as *identity federation* and *single sign-on*, Federation Termination, etc. It also defines various profiles for each protocol, which provides the flexibility for implementations. SourceID provides the open source framework for our later simulation of our proposed system.

Based on the background information provided in this chapter, in the following chapter, a model as to how an MMSC can work with other service/content providers will be developed.

Chapter 3 Development of an MMS-Based System for Integrated Services

3.1 Introduction

As discussed in Section 1.1, there exist some drawbacks in the current co-operation between MMS and other service/content providers. The limitations of e-commerce are the main cause of these drawbacks. As explained in Chapter 2, the Liberty Alliance specifications are intended to overcome these limitations. The purpose of this thesis is to develop a prototype to implement Liberty *single sing-on and identity federation* into MMS, so that those drawbacks can be solved. In this chapter, we address the problem of implementing *Liberty* protocols into MMS.

Compared with the existing systems, the proposed solution adds new functionalities into the MMS components and modifies some of the message flows so that *identity federation* and *single sign-on* can be integrated into MMS. These additions and modifications are based on Liberty Alliance specifications.

In order to facilitate the proposed solution, we start with specific example of service providers, Kodak, and later generalize to other services. In our proposed system, basically, a user sends a simple SMS or MMS message, which in turn authenticates the consumer's device and enables him/her to sign up for the Kodak service. After the user's request is processed, a simple return message is relayed to the handset in an MMS (or SMS) response. The Figure 3.1 depicts the proposed system integrating a service provider (in our case, Kodak) with MMS. It can be seen that single sign-on and federation protocol enables the user to access services from both MMS and Kodak with only one login.

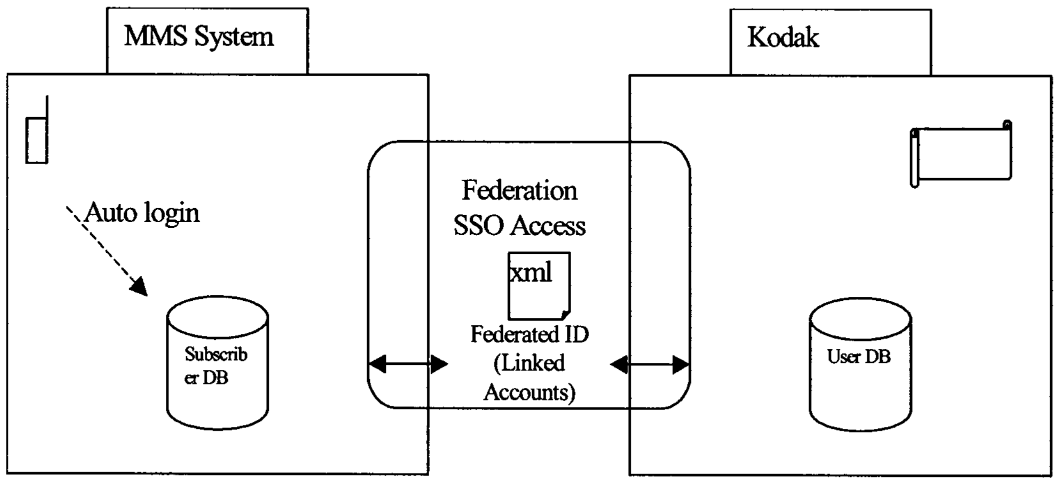


Figure 3.1 Proposed integration of Kodak with MMS system

3.2 System Specification

To bring the Kodak service into the proposed system, the main components and system roles are described below.

3.2.1 Multimedia Messaging Service Centre

The MMSC, as the multimedia message forwarder and storage facility, retains all of its current functionalities. The necessary additional functionalities are listed below:

- After a message is stored, relay will check if the destination is one of the trusted service providers.
- Relay also checks if the originator has been federated with the service provider.
- The MMSC should be able to initiate or terminate federation upon user's request with other service providers.

- In the MMSC's subscriber database, the subscriber table contains the fields for federation.
- A VASP table contains the fields necessary to indicate whether a trust relationship is established between the VASP and the MMSC or not.

In Liberty specifications, identity providers and service providers affiliate together into a *circle of trust* [25]. In our circle of trust, the MMSC acts as both *identity provider* and *service provider.* On one hand, the MMSC creates, maintains, and manages identity information for mobile users and provides mobile user authentication to other service providers (in our case, Kodak). On the other hand, MMSC also provides multimedia messaging service to mobile users.

The multimedia messaging service is always provided by mobile operators. The MMSC usually has a completed subscriber database. In our system, users' requests are sent as/with multimedia messages. Therefore, it is convenient to use an MMSC as the *identity provider*.

The implementation of *Liberty* affects the following components of MMSC:

Proxy-Relay: Federation verification needs to be added into this component. In the current MMSC, relay receives and forwards the message from/to the subscribers. With implementation of *Liberty*, relay needs to verify if the subscriber has federated his/her accounts with the service provider.

Message Store: The schedule of retry or delete messages according to whether the user is federated or not needs to be added. In case of failure to deliver a message to service providers, message store need to retry delivery if the user is federated, or delete the message if he/she is not federated.

Subscriber Database: Extra tables and fields to mark federated/not federated, and with whom need to be added. The subscriber table should contain a field indicating whether this subscriber federates his/her account with other service providers. Then, in the federation table, records tell with which providers he/she is federated. The federated providers may number more than one.

3.2.2 Service Provider

Note that for the purposes of this thesis, Kodak is used just as the name of a service provider. It does not literally mean that a software is being developed for Kodak Company. Kodak image service is a good example of one of the hundreds of services available. Kodak is not the only possible service provider -- obviously, we can also have X Taxi Company or Y Music Centre as service providers too.

The basic components that a Kodak service needs are:

Identity Management Component: This component integrates the *Liberty* protocols and profiles to perform authentication, *single sign-on and identity federation*, *federation termination*, etc.

Application Box: This is the service provided; in our case the company receives the request from a mobile subscriber, prints the picture, sends the print to the customer, etc.

User Database: User profile/information is stored in the database. Mirroring a similar table contained by the subscriber database at the MMSC, the user table should contain a field indicating whether this subscriber federates his/her account with other service providers. The federation table also has the similar characteristics of the table at the MMSC.

The *service provider* (in this thesis, Kodak) should be able to initiate or terminate federation as well as an MMSC. Kodak acts as a VASP connected to an MMSC. On the

MMSC side, Kodak would be registered in the database (VASP table) and marked as a "trusted" partner, and would be used as a company providing only digital picture printing service.

3.3 System Architecture

MMSC and service providers affiliate together in a circle of trust (Figure 3.2). This circle of trust enables mobile subscribers to transact business in a secure and apparently seamless environment.

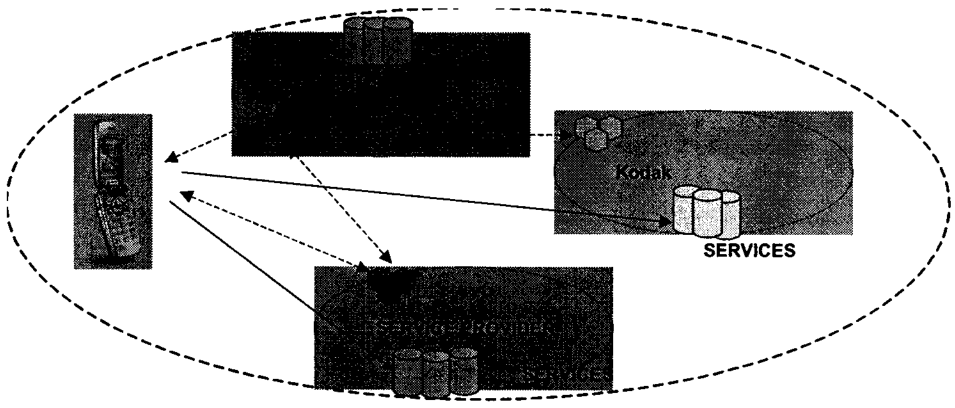


Figure 3.2 A circle of trust formed by MMSC and other service providers

3.3.1 User Accesses

In our system, a user needs to access both the MMSC and Kodak to order a photo print. By accessing MMSC, the user uploads his/her pictures to Kodak; and by accessing Kodak, the user can have his/her picture printed. When the user accesses MMSC via the mobile operator's Core Network as shown in Figure 3.3, the access-based authentication done by the Core Network can be reused at the service layer. This is achieved by exploiting the MSISDN forwarding mechanism. Thus, the end-user experiences only a service delivery, seamlessly and transparently incorporating authentication. In other words, if the subscriber does not pass the checking done in Core Network, he/she cannot send any message to MMSC. On the terminal, he/she will receive a notification "not allowed to send MM".

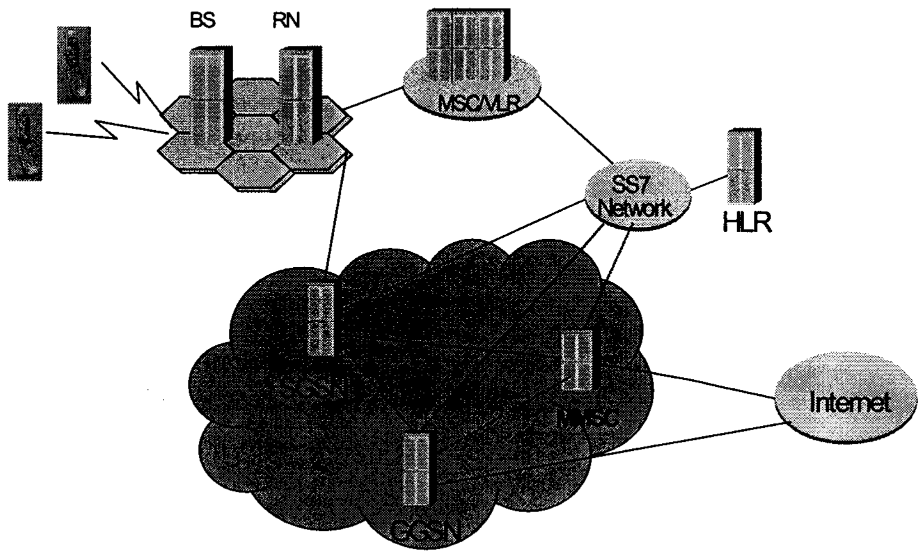


Figure 3.3 MMSC connecting to core network (via SGSN or GGSN)

The MMSC can accept the message from an originator if he/she passes the checking step in Core Network. Before it proceeds with a transaction, however, MMSC may have to do some additional verification against this originator. For example, it may be required to check if the originator is in the blacklist, is a pre-paid subscriber, has sufficient credit to send this message, etc. The additional checking is usually done in the MMSC subscriber database as illustrated in Figure 3.4.

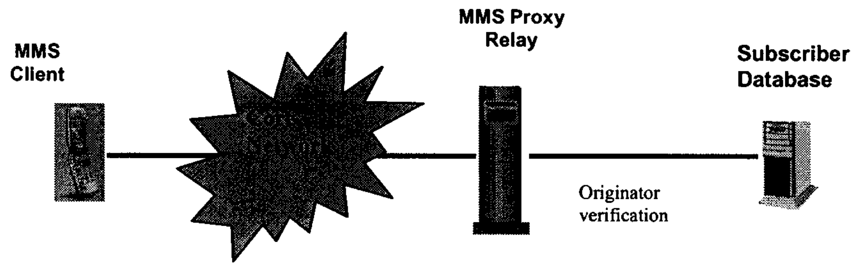


Figure 3.4 Mobile user accessing MMSC

Traditionally, the user accesses Kodak via Internet as shown in Figure 3.5. The authentication mechanism functions through the use of a username and a password.

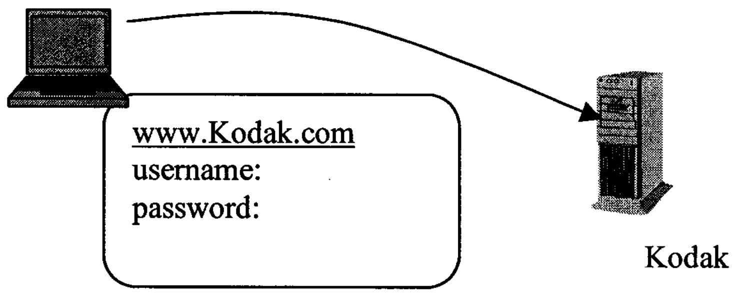


Figure 3.5 User accessing Kodak

3.3.2 User Experiences

Once the MMSC and Kodak are federated by the *Liberty* technology, users can benefit from s*ingle sign-on and identity federation.*

Identity federation user experience

There are two options available for a federation user: explicit and implicit. They both lead to user accounts linked between the MMSC and Kodak, as illustrated in Figure 3.6.

Explicit: An *identity federation* user experience can begin either from the MMSC or from Kodak. The MMSC sends notification to the user asking if he/she consents to federate with his/her Kodak account (Alternatively, when user logs onto the Kodak site, he/she could be asked whether he/she agrees to federate with his/her MMSC account). Upon answering "yes", the user gets his/her MMSC identity (MSISDN) federated with his/her Kodak identity (user@Kodak).

Implicit: The user does not feel the federation initiation. Federation between an MMSC and Kodak is created by a mutual agreement, after which all MMS subscribers are considered federated with Kodak by default.

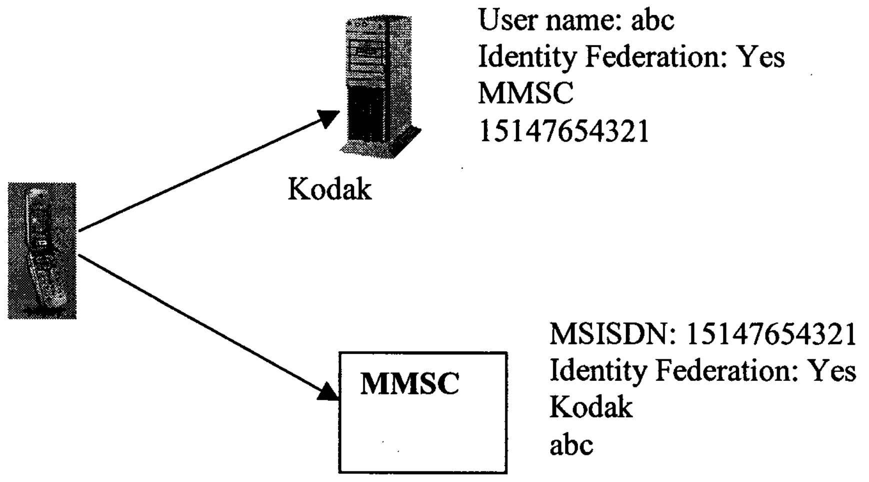


Figure 3.6 Identity federation user experience

Single sign-on user experience

Accessing Kodak from MMSC: *Single sign-on* user experience enables the users to access Kodak via multimedia messages. It means that, after federation, and upon being authenticated in a Core Network, users can send requests to Kodak or receive responses from Kodak via the MMSC. This user experience is shown in Figure 3.7.

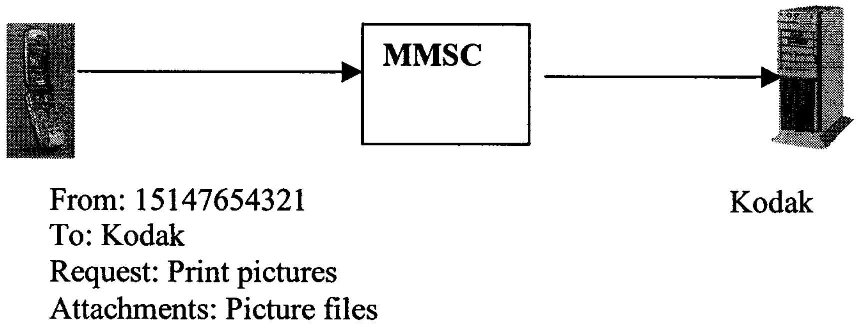


Figure 3.7 Single sign-on user experience (access via MMS)

Accessing MMSC from Kodak: *Single sign-on* user experience can also enable the users to access the MMSC via Kodak, for example, to check their messages via Internet, as shown in Figure 3.8. This is optional depending on whether or not the MMSC allows users to download the messages from Internet.

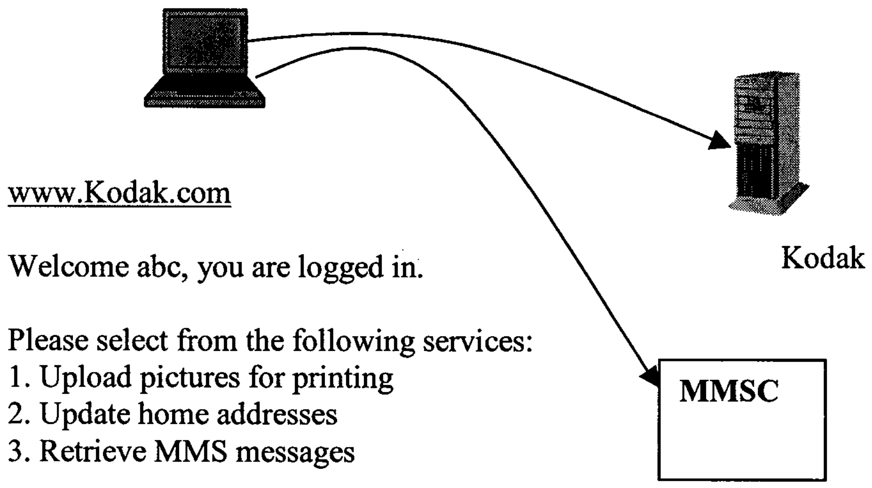

www.Kodak.com
Welcome abc, you are logged in.
Please select from the following services:
1. Upload pictures for printing
2. Update home addresses
3. Retrieve MMS messages
Kodak
MMSC

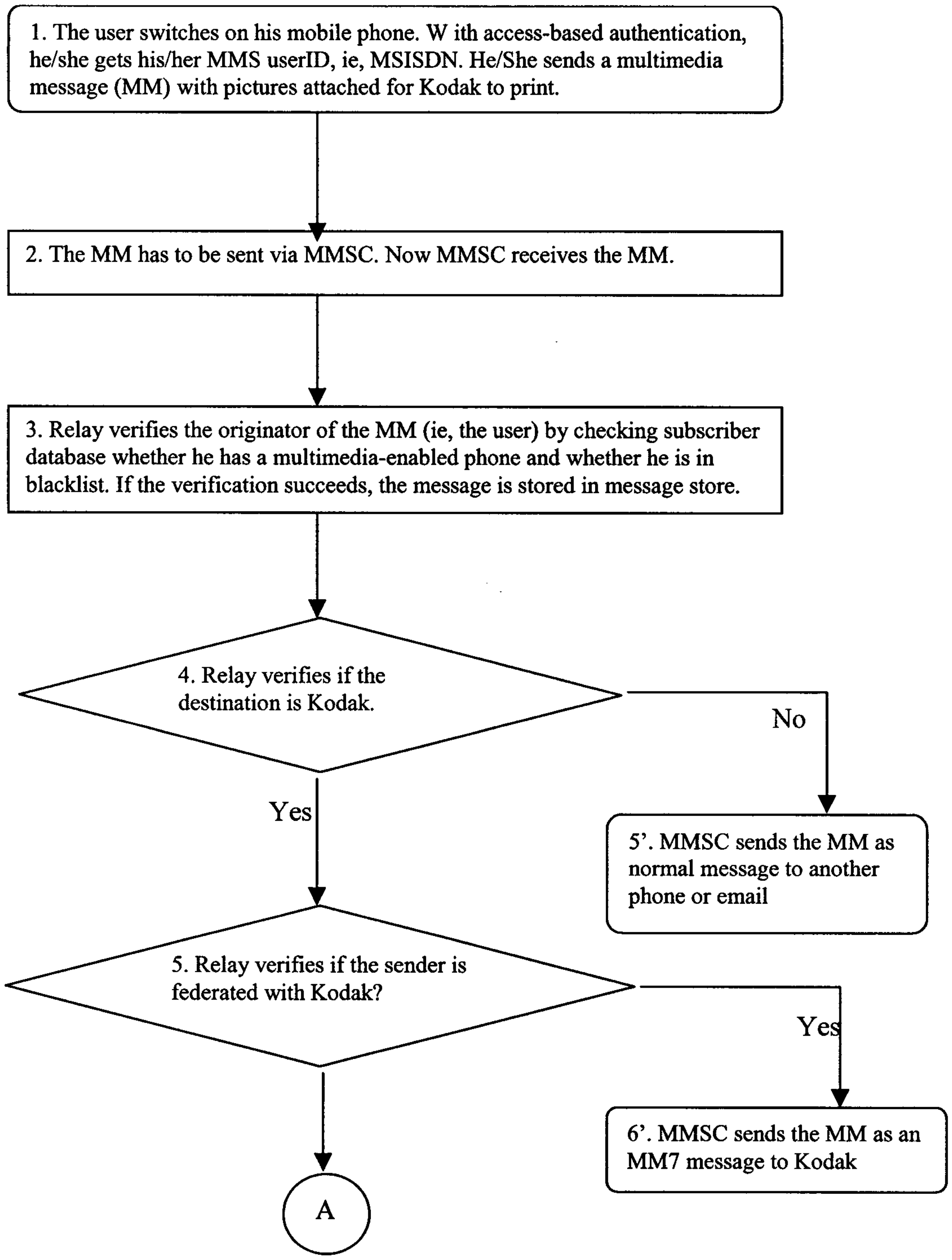
Figure 3.8 Single sign-on user experience (access via Kodak)

3.3.3 Message Flow Charts

Just as for the user, operators can also have two types of federation, explicit and implicit. The message flows will again be different with different federation types.

Explicit federation

Explicit federation can be initiated from both sites. Scenario 1 is shown in Figure 3.9, in which the federation is initiated from an MMSC.


1. The user switches on his mobile phone. W ith access-based authentication, he/she gets his/her MMS userID, ie, MSISDN. He/She sends a multimedia message (MM) with pictures attached for Kodak to print.
2. The MM has to be sent via MMSC. Now MMSC receives the MM.
3. Relay verifies the originator of the MM (ie, the user) by checking subscriber database whether he has a multimedia-enabled phone and whether he is in blacklist. If the verification succeeds, the message is stored in message store.
4. Relay verifies if the destination is Kodak.
No
Yes
5'. MMSC sends the MM as normal message to another phone or email
5. Relay verifies if the sender is federated with Kodak?
Yes
6'. MMSC sends the MM as an MM7 message to Kodak
A

(continued on Page 51)

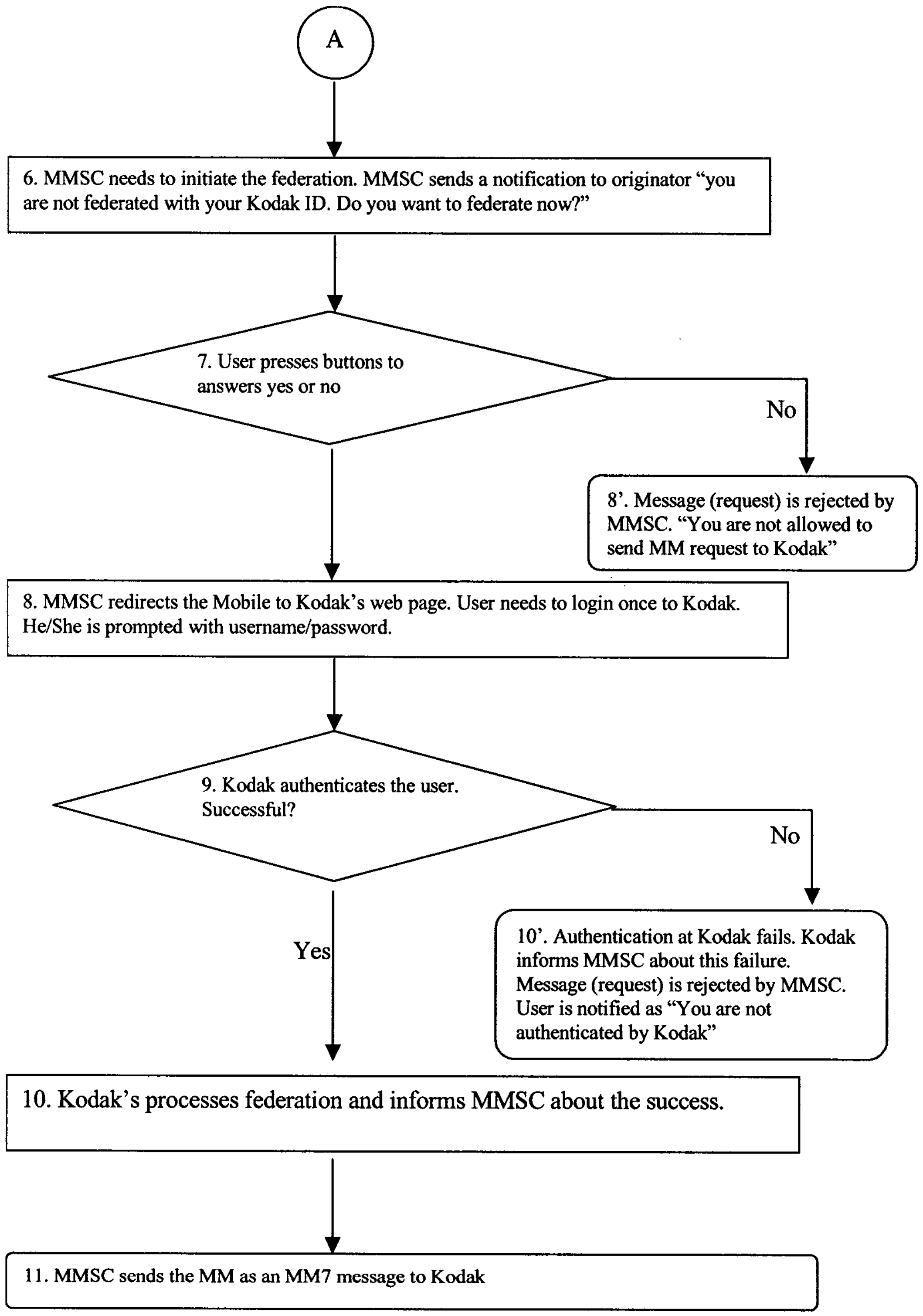

A
6. MMSC needs to initiate the federation. MMSC sends a notification to originator "you are not federated with your Kodak ID. Do you want to federate now?"
7. User presses buttons to answers yes or no
No
8'. Message (request) is rejected by MMSC. "You are not allowed to send MM request to Kodak"
8. MMSC redirects the Mobile to Kodak's web page. User needs to login once to Kodak. He/She is prompted with username/password.
9. Kodak authenticates the user. Successful?
No
Yes
10'. Authentication at Kodak fails. Kodak informs MMSC about this failure. Message (request) is rejected by MMSC. User is notified as "You are not authenticated by Kodak"
10. Kodak's processes federation and informs MMSC about the success.
11. MMSC sends the MM as an MM7 message to Kodak

Figure 3.9 Explicit federation: MMSC-initiated federation message flow

Scenario 2 is shown in Figure 3.10, in which the federation is initiated from Kodak's web page. User logs onto Kodak's web page from his/her mobile phone.

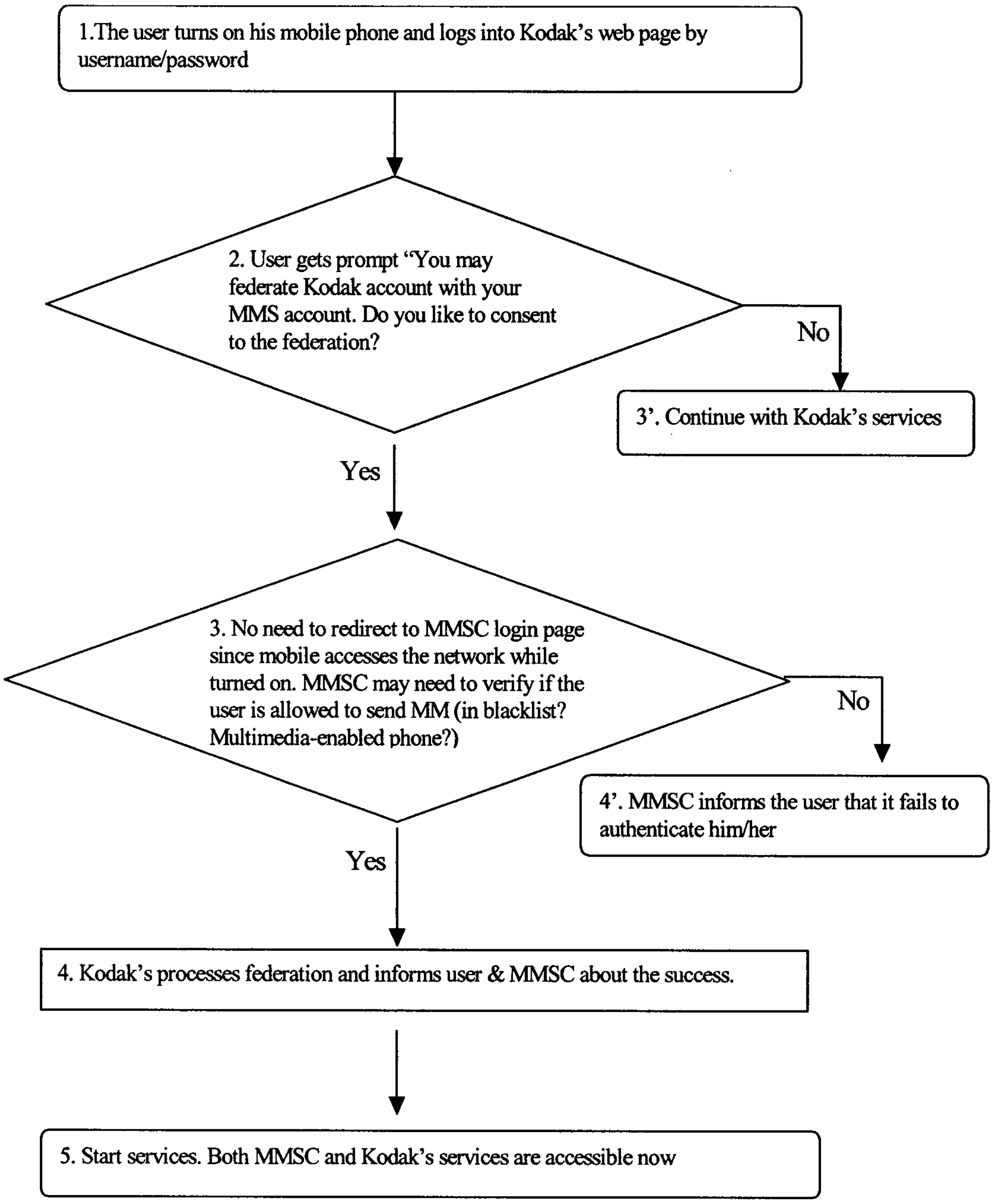

1.The user turns on his mobile phone and logs into Kodak's web page by username/password
2. User gets prompt "You may federate Kodak account with your MMS account. Do you like to consent to the federation?
No
3'. Continue with Kodak's services
Yes
3. No need to redirect to MMSC login page since mobile accesses the network while turned on. MMSC may need to verify if the user is allowed to send MM (in blacklist? Multimedia-enabled phone?)
No
4'. MMSC informs the user that it fails to authenticate him/her
Yes
4. Kodak's processes federation and informs user & MMSC about the success.
5. Start services. Both MMSC and Kodak's services are accessible now

Figure 3.10 Explicit federation: Kodak-initiated federation message flow

Note that the use of a PC always leads to Scenario 2. When the user logs in to Kodak from a computer, the browser will direct the user for the federation. In this case, Step 3 will be different, in that the user may need to provide login information in order to be authenticated once by MMSC. There are two options:

1. The user needs to type his/her username and password if pre-assigned by the MMSC.

2. There is no additional username and password. User just uses his/her MSISDN for authentication. This option has security risk.

Implicit federation

By using implicit federation, user interactions are not involved in the federation procedure. Federation is done when the MMS operator and Kodak agree to co-operate to provide the users with enhanced services.

Whenever the user sends a picture to Kodak via a Multimedia message, MMSC verifies the sender as usual, and then the message is forwarded to Kodak right away. The message flow is illustrated in Figure 3.11. It is seen from this figure that the use of the implicit federation is much simpler than that of the explicit federation.

3.4 Interactions within the Federation

If the operator chooses an implicit federation, the federation tags is added into the user database by the MMSC and Kodak system administrators. In this case, any extra authentication from Kodak will not be needed. However, implicit federation is not recommended, because it is completely out of the control of the end-users; thus, it has security risk, and may cause billing disputes. In this thesis, we therefore focus on the explicit federation.

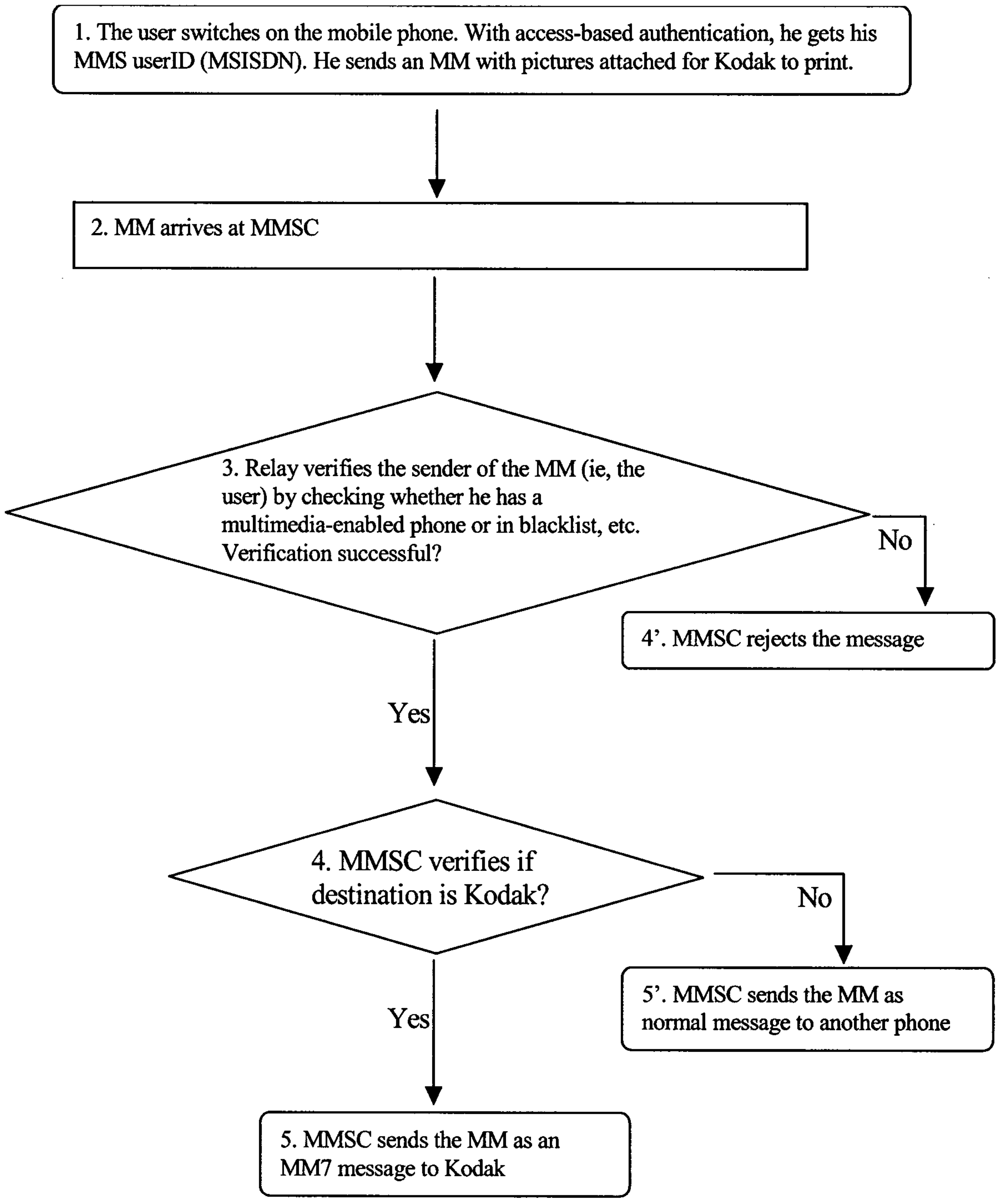

1. The user switches on the mobile phone. With access-based authentication, he gets his MMS userID (MSISDN). He sends an MM with pictures attached for Kodak to print.
2. MM arrives at MMSC
3. Relay verifies the sender of the MM (ie, the user) by checking whether he has a multimedia-enabled phone or in blacklist, etc. Verification successful?
No
4'. MMSC rejects the message
Yes
4. MMSC verifies if destination is Kodak?
No
5'. MMSC sends the MM as normal message to another phone
Yes
5. MMSC sends the MM as an MM7 message to Kodak

Figure 3.11 Implicit federation message flow

We use the *single sign-on and federation* profile that is described in [13]. Scenario 2 of explicit federation is a typical *Liberty single sign-on federation* profile. In this profile, Kodak (*service provider*) obtains an authentication assertion of a user agent from the MMSC (*Identity provider*) to facilitate the *single sign-on*. It is also the means by which an identity is federated from a *service provider* to an *identity provider* through the use of the <NameIDPolicy> element in the <lib:AuthenRequest> protocol message as specified in [16]. The workflow of scenario 2 is shown is Figure 3.12.

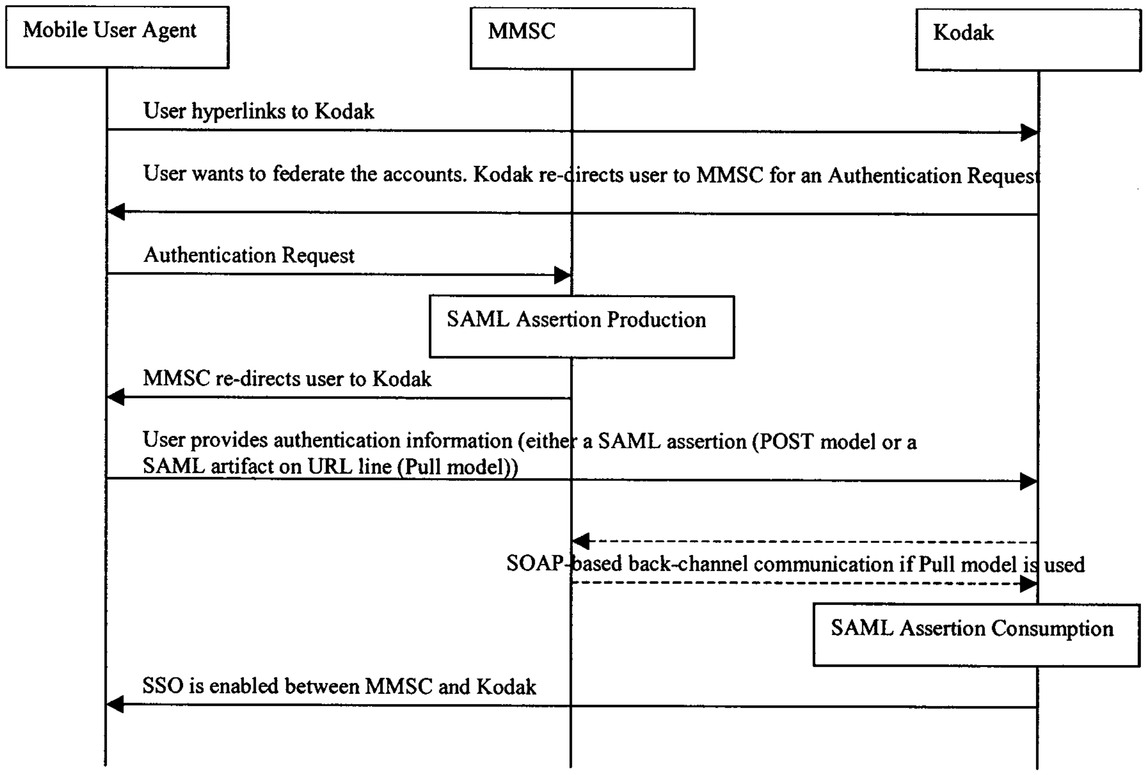

Mobile User Agent
MMSC
Kodak
User hyperlinks to Kodak
User wants to federate the accounts. Kodak re-directs user to MMSC for an Authentication Request
Authentication Request
SAML Assertion Production
MMSC re-directs user to Kodak
User provides authentication information (either a SAML assertion (POST model or a SAML artifact on URL line (Pull model))
SOAP-based back-channel communication if Pull model is used
SAML Assertion Consumption
SSO is enabled between MMSC and Kodak

Figure 3.12 Liberty SSO and Federation initiated by Kodak

In Scenario 1 of explicit federation, a mobile user automatically logs into the MMSC while sending a message. In Figure 3.9, the first six steps are features of the MMSC; at that time, federation has not yet taken place. The interactions within the federation start at Step 7 of Figure 3.9. These interactions are broken down as shown in Figure 3.13.

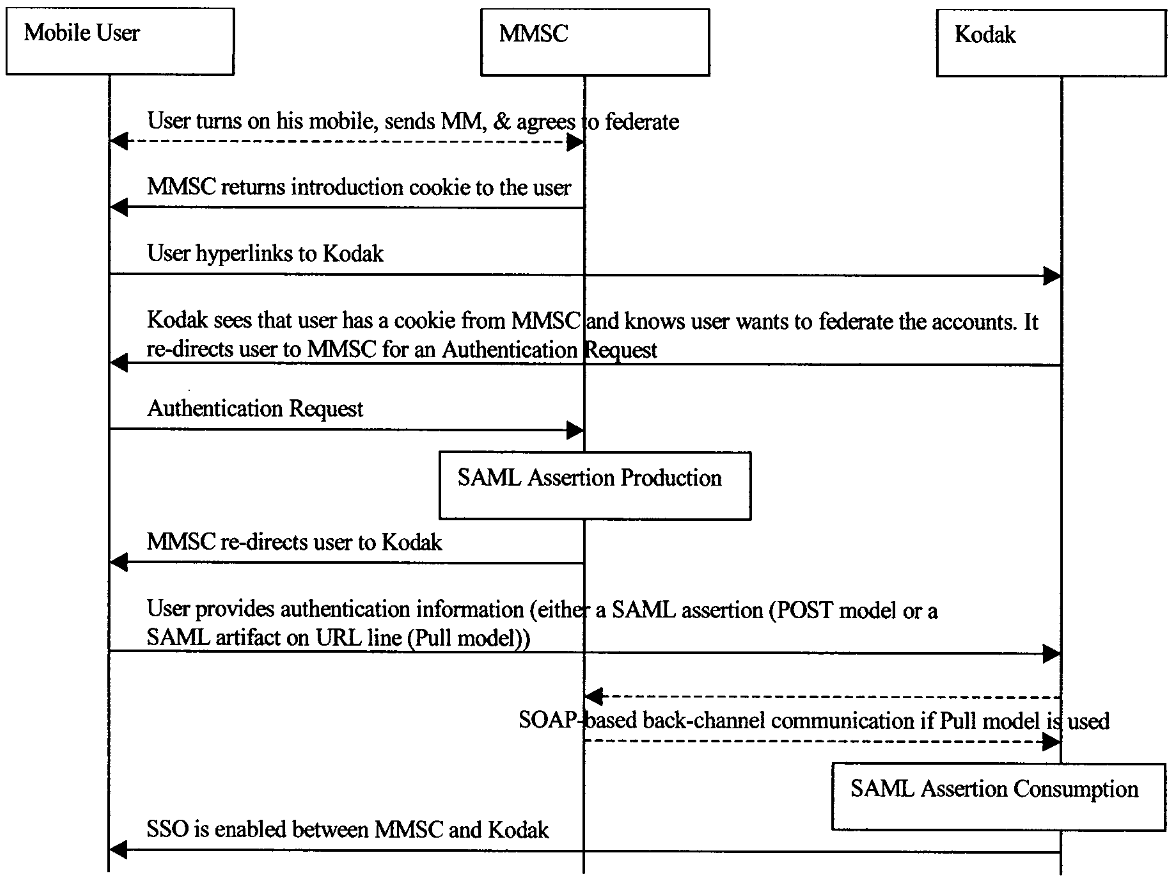

Mobile User
MMSC
Kodak
User turns on his mobile, sends MM, & agrees to federate
MMSC returns introduction cookie to the user
User hyperlinks to Kodak
Kodak sees that user has a cookie from MMSC and knows user wants to federate the accounts. It re-directs user to MMSC for an Authentication Request
Authentication Request
SAML Assertion Production
MMSC re-directs user to Kodak
User provides authentication information (either a SAML assertion (POST model or a SAML artifact on URL line (Pull model))
SOAP-based back-channel communication if Pull model is used
SAML Assertion Consumption
SSO is enabled between MMSC and Kodak

Figure 3.13 Liberty SSO and Federation initiated by MMS

3.5 Database

The persistent medium storage (such as the MMSC subscriber database or Kodak user database) might need to be extended in order to store the Liberty-specific account federation attributes. For each account in the persistent storage, it may be necessary to the store federation records, indicating that the account has been federated with an account at a remote service or identity provider. Each such "federation record" references a single account, and contains three additional attributes.

- The *provider ID* (a string) of the remote provider this account is federated with

- The *locally-provided name identifier* (a string), which the remote provider will use to identify this user when sending back messages
- The *remotely-provided name identifier* (a string), which will be used when communicating with the remote provider about this user.

For the ease of description, examples of table and column name are used in this section. Assume that a relational database is used to store the account information. Also assume that a database table “accounts” is already in place, defined with the columns of “username” and “password”. The primary key is “username”.

In this scenario, it is required to add another table “account_federation” to store the federation attributes, such as “username”, “provider_id”, “local_name”, and “remote_name”. The column “username” in this table is a foreign key which references the “username” in the “accounts” table. The columns of “local_name” and “remote_name” refer to the *locally-provided name identifier* and *remotely-provided name identifier* mentioned previously. The values in the two columns can be opaque so that the principal information, such as the real user name or the mobile number, is not distributed across the network. When the *identity provider* and the *service provider* receive the name identifiers from each other, they use specific mapping rule translate them to obtain the real user names. The details about the name identifier can be found in the description of the protocol of *name registration* and *name identifier mapping* in [12].

The “account_federation” table should be created on both the identity provider and the service provider sites. When there is a federation request, entries will be added in the “account_federation” table on all sites, in our case, the MMSC and Kodak sites. Figure 3.14 shows how one federation request affects the two databases.

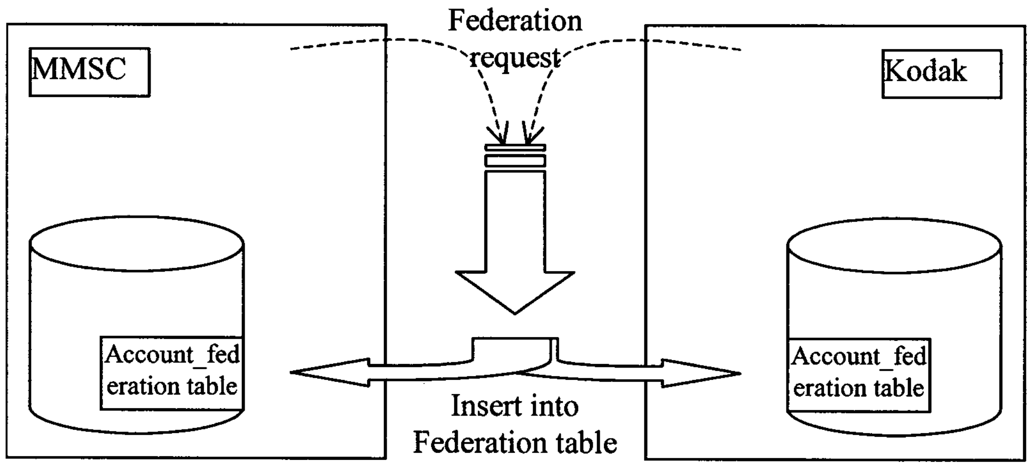


Figure 3.14 Insertion of entries into 'account_federation" tables upon a federation request

3.6 Summary

In this chapter, we have developed a model showing how the MMS could work with other service/content providers, such as the Kodak picture center. We have proposed the system architecture, system specification, message flows, and database structures.

There are two options available to perform federation – explicit or implicit. Explicit federation requires login to a service provider to initiate federation, which is more under the user's control. Implicit federation does not require any login for federation; this is setup by mutual agreement between the MMSC and the service providers.

In the next chapter, we demonstrate how requests/messages are forwarded to a service provider from the MMSC in different scenarios. Explicit federation, being the safer solution, is developed in the demo system.

Chapter 4 Simulation Model and Test Result

In this chapter, a simulation model is presented in order to test the feasibility of the MMS-based system for integrated services proposed in Chapter 3. For this purpose, appropriate software is developed to incorporate the Liberty functionalities into the MMS and Kodak applications, based on SourceID Liberty 2.0 framework. The software works in a network environment and enables the nodes in this network to communicate with one another.

4.1 Simulation System Requirement

The hardware and software requirements are given below:

4.1.1 Hardware Environment

The simulation system is developed on a LAN network that contains at least two computers to represent the MMSC and Kodak, as shown in Figure 4.1.

The hardware requirements are:

- Two computers each with a 1GHz or faster CPU, a memory of 256 MB or more, and at least 20GB of hard disk space
- LAN adapters

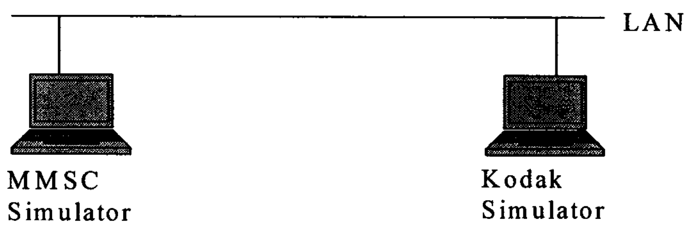


Figure 4.1 Hardware environment for simulation

4.1.2 Software Requirement

The simulation software is developed in Java using the JBoss server as the web service platform. Based on SourceID Liberty 2.0, MMS and Kodak applications are developed with customized data storage. The requirements for operating system and software are listed below.

- Windows 2000 or XP, or Linux (Kernel 2.4.2+)
- Sun Java JDK 1.4.2
- Jboss 3.2.5
- SAAJ API
- MySQL 4.0
- MySQL Connector API
- SourceID Liberty 2.0

4.2 System Development

In this simulation system, two nodes are developed – MMSC and Kodak, as shown in Figure 4.1. At each node, applications composed of Java classes, servlets, and JSP files are developed on the JBoss server. In order to integrate the Liberty protocols with these applications, the interfaces provided by SourceID Liberty 2.0 are implemented into the simulation environment.

In general, exchange of messages between the nodes, picture printing, and Liberty interactions are controlled by the workflow processes. The servlets are used to send and receive requests, as well as to start the workflow instance. The JSP files construct the login, logout, and message composition pages.

The functions of the MMSC simulator are as follows.

- Receive multimedia message from the mobile simulator
- Store messages
- Verify senders and recipients
- Forward the multimedia messages to mobile users or to VASP
- Contain federation information in its subscriber database

The functions of the Kodak simulator are as follows.

- Receive messages sent from the MMSC
- Print the picture that is attached in the multimedia message
- Access the MMSC from the Kodak page, if the user is federated with the MMSC
- Contain federation information in its user database

In addition, there is a small application that simulates the mobile handset. It consists of JSP pages and servlets. This "handset" is located at the same node of the MMSC simulator.

4.3 Architecture and Workflow

4.3.1 Architecture

The simulation system is composed of a web container [26] and a J2EE container (see the Glossary), which are mapped into the physical Java packages. The J2EE container is formed from a workflow engine and a workflow package. In this system, OBE is used as the workflow engine to handle the workflow. In the workflow package, various processes are defined. Each process is composed of participants, activities, and relevant data, etc. The web container manages the execution of JSP page and servlet components for J2EE applications.

The high-level architecture shown in Figure 4.2 is constructed by mapping the Java packages with the J2EE container and Web container into workflow conceptual architecture.

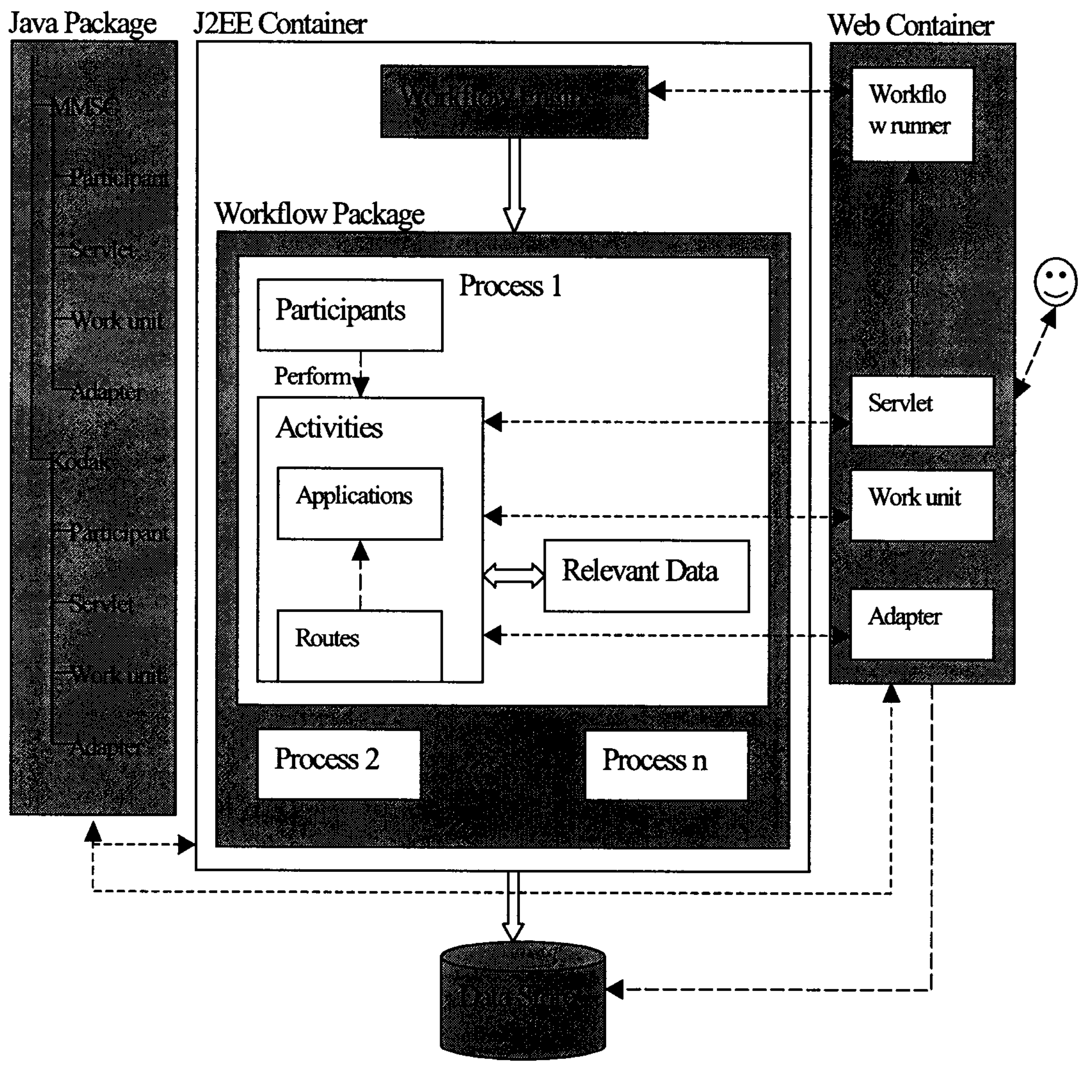


Figure 4.2 Mapping Java classes to the workflow architecture

4.3.2 Workflow Package

In the simulation system, one new process, "MMSC verify", is added to the workflow package. The pattern of the modified workflow package is illustrated in Figure 4.3.

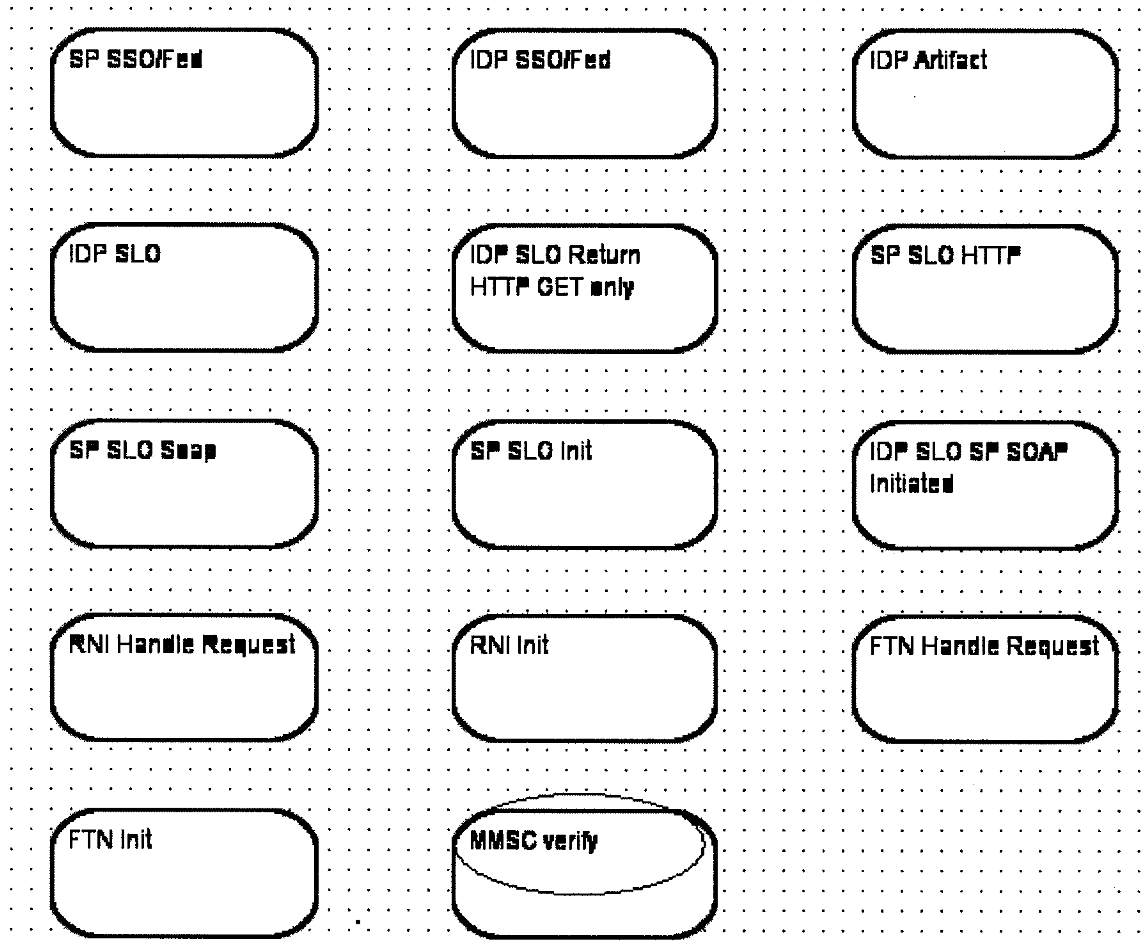


Figure 4.3 The workflow package pattern for the simulation system

The process "MMSC verify" has the activities, VerifySender, Fail-verification, Iskodakdestination, Federation-success, PrintMessage, Mmsc-end, SendMessage, Response-error, GetFedInfo, InitiateFederation, and Federation-ok. These activities are mapped into the methods in Java classes. The description of the activities can be found in Section 4.4. The inter-working of these activities forms the process flow of "MMSC verify" as shown in Figure 4.4.

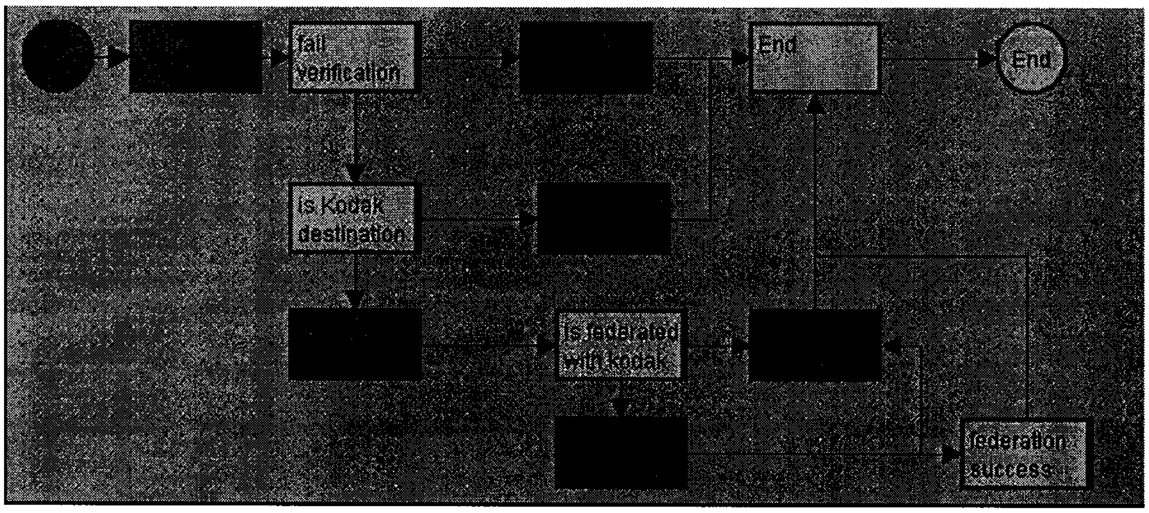


Figure 4.4 Process flow of "MMSC verify"

The process "MMSC verify" uses the relevant data, Msisdn, Mmheader, Mmbody, VaspID, SessionInfo, RelayState, FedInfo, and Soapmessage. They carry the information that is needed for the activities to be performed.

4.4 Java Classes and Packages

A Java package is made up of a set of Java classes that deal with specific tasks. In our simulation system, there are 4 types of Java packages, adapters, participants, servlets, and work units.

4.4.1 Participant

Participants perform the work represented by workflow activity instances, for example, log the user out, federate the user, etc. In this simulation system, there is one participant named "MMSC". In the participant package, there is one Java class "MMSCParitidpant.class". This class specifies the activities that are performed by the participant "MMSC". The class is illustrated in Figure 4.5 (see [27] for class diagram).

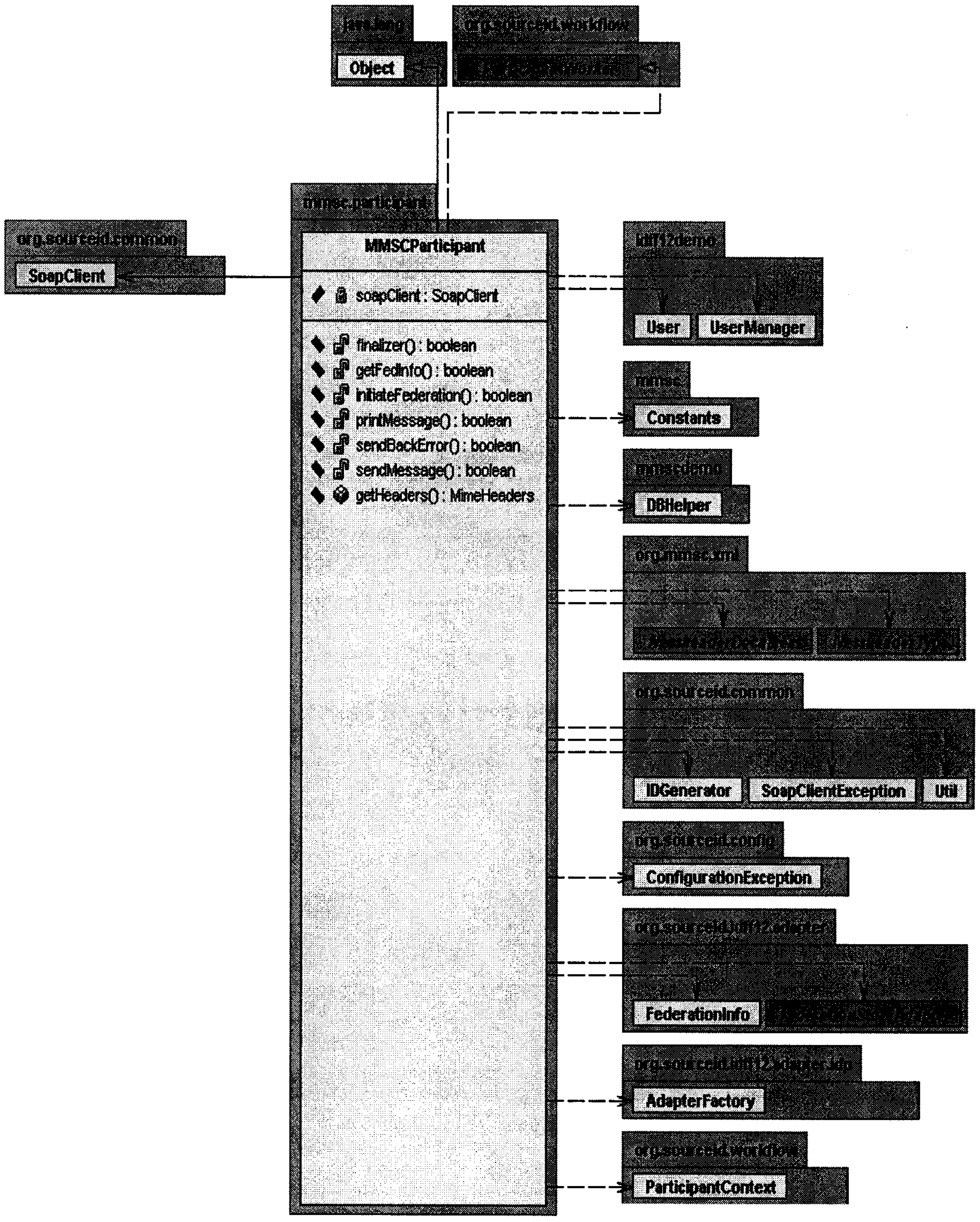


Figure 4.5 The class "MMSCParticipant.class"

The activities are as follows

GetFedInfo: to retrieve federation information and pass it to workflow

InitiateFederation: to initiate the federation

PrintMessage: to call the printing service on Kodak side to print the picture

SendBackError: to return the authentication failure from MMSC

SendMessage: to compose and send messages to Kodak

GetHeader: to get the message header

4.4.2 Work Unit

Work units are actually the workflow applications. They handle part of the processing required to support a particular activity (or activities), for example, create and return the assertion, or retrieve and return the artifacts.

In the work unit package, there is one class named "VerifySender.class" as illustrated in Figure 4.6. This class verifies if the sender is a valid MMS user. It queries the database table to check whether this user is in the blacklist, has enough funds, etc, and returns the session information indicating the result of querying.

4.4.3 Servlet

Servlets are Java classes that run as part of network services, typically web servers, and respond to requests from the clients. The *single sign-on* initiation service, *federation* termination service, etc, are implemented via servlets. In our simulation system servlets are also responsible to start or resume workflow processing. There are four servlet classes in the package, "MMSCInitiateServlet.class", "MMSCServlet.class", "ResumeCenterService.class" and "SOAPRouterServlet.class", shown in Figures 4.7-4.10, respectively.

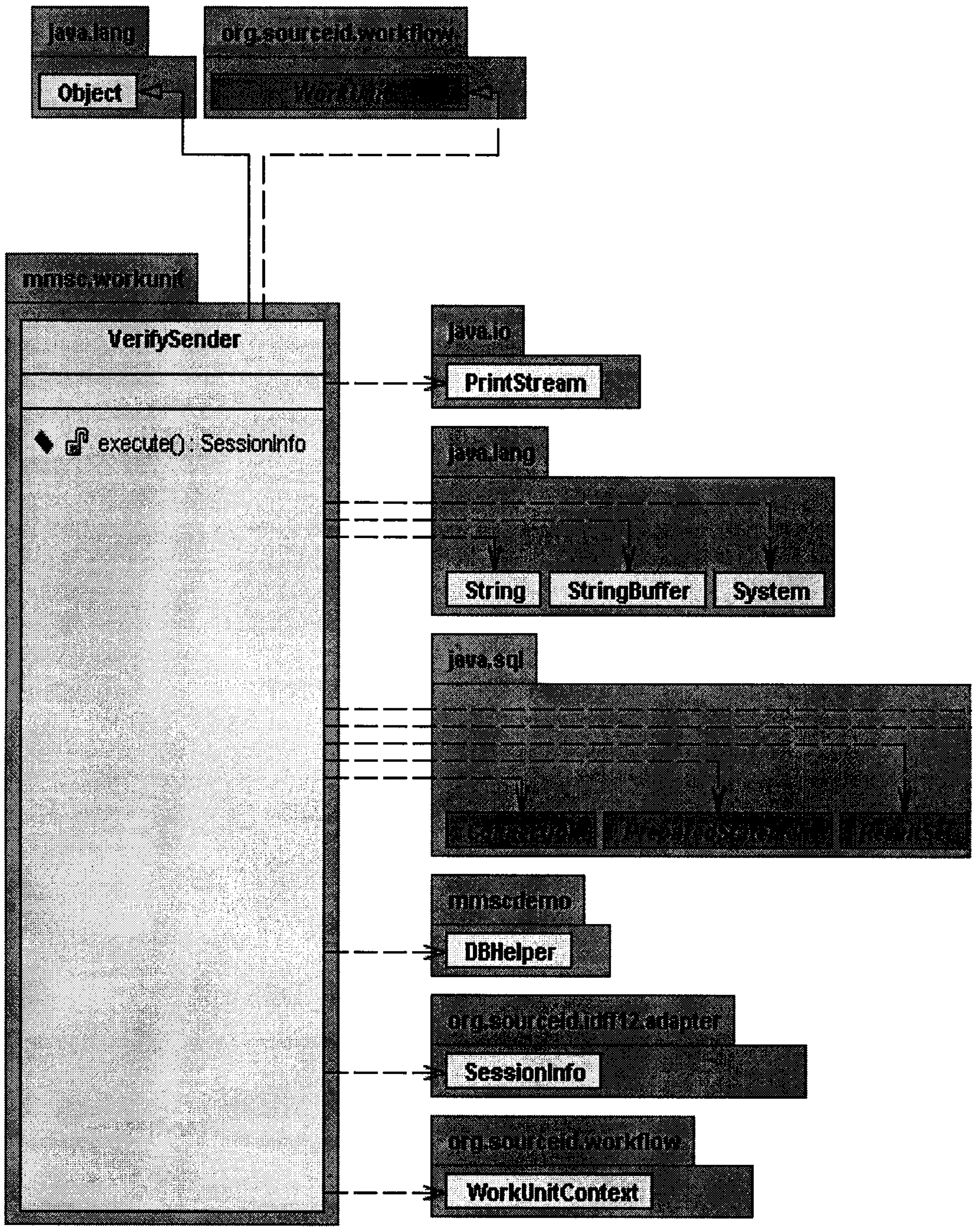


Figure 4.6 The class "VerifySender.class"

As shown in Figure 4.7, “MMSCInitiateServlet.class” contains one method, “doPost”. Its functionality is to build up the SOAP message, pass it to workflow, and start the workflow instance.

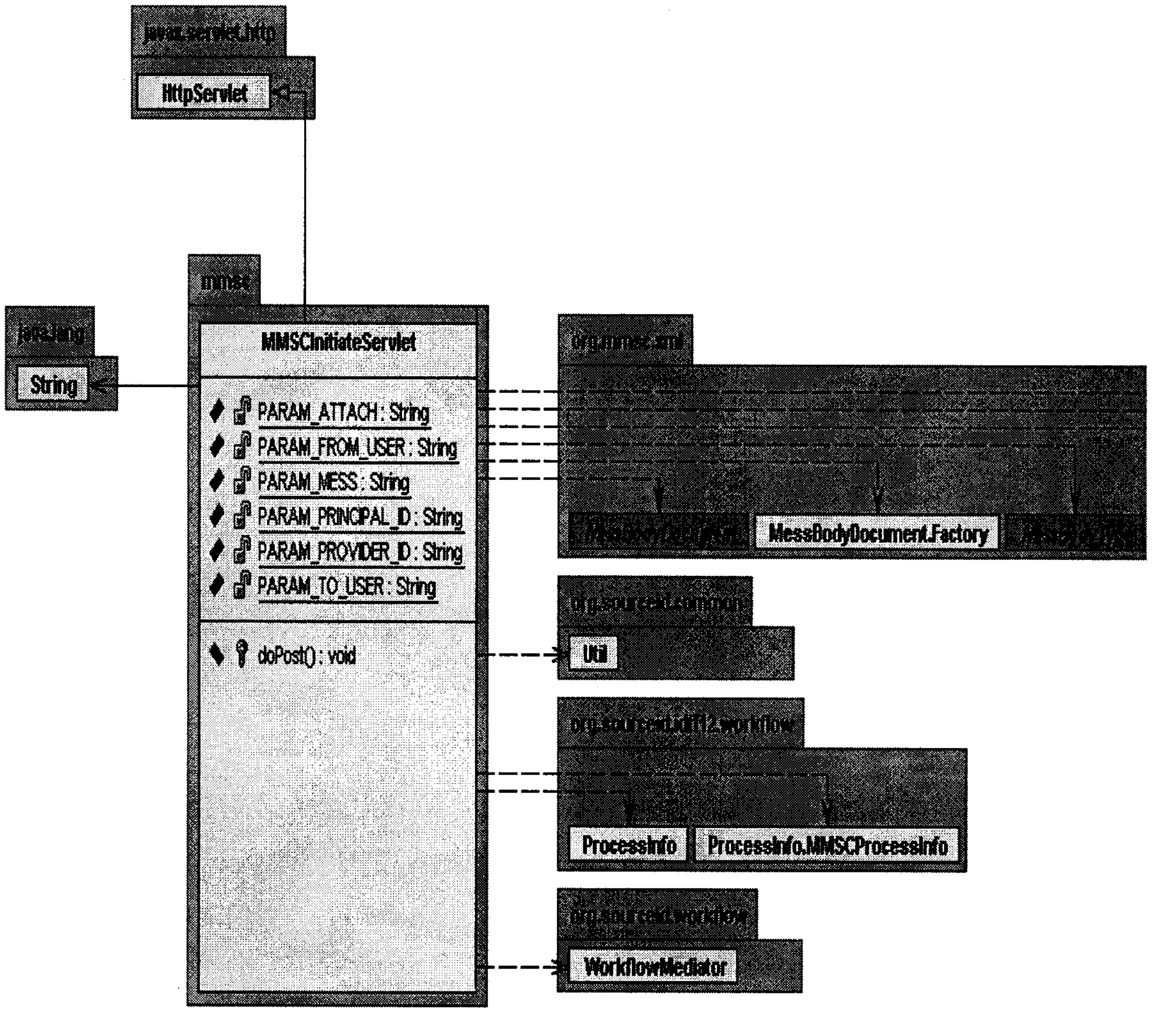


Figure 4.7 The servlet “MMSCInitiateServlet.class”

In Figure 4.8, the servlet “MMSCServlet.class” is illustrated. This servlet processes the user’s login, redirects the user to Kodak site, and lists the user messages in his/her message box. The methods and their functionalities in this Java class are as follows:

GetAllNewMessages: to list the received messages of the user

GetAllSendMessages: to list the sent messages of the user

DoGet: to authenticate the user using HTTP GET

DoPost: to authenticate the user using HTTP POST

DoProcess: to authenticate the user and direct the user to appropriate pages

GoToLoginPage: to prompt the login page of the identity provider

GoToSPApplicationPage: to login to service provider's page and meanwhile enable the access to identity provider as well

SetURLAttrs: to set the URL attributes, for example, base URL, returning URL, etc.

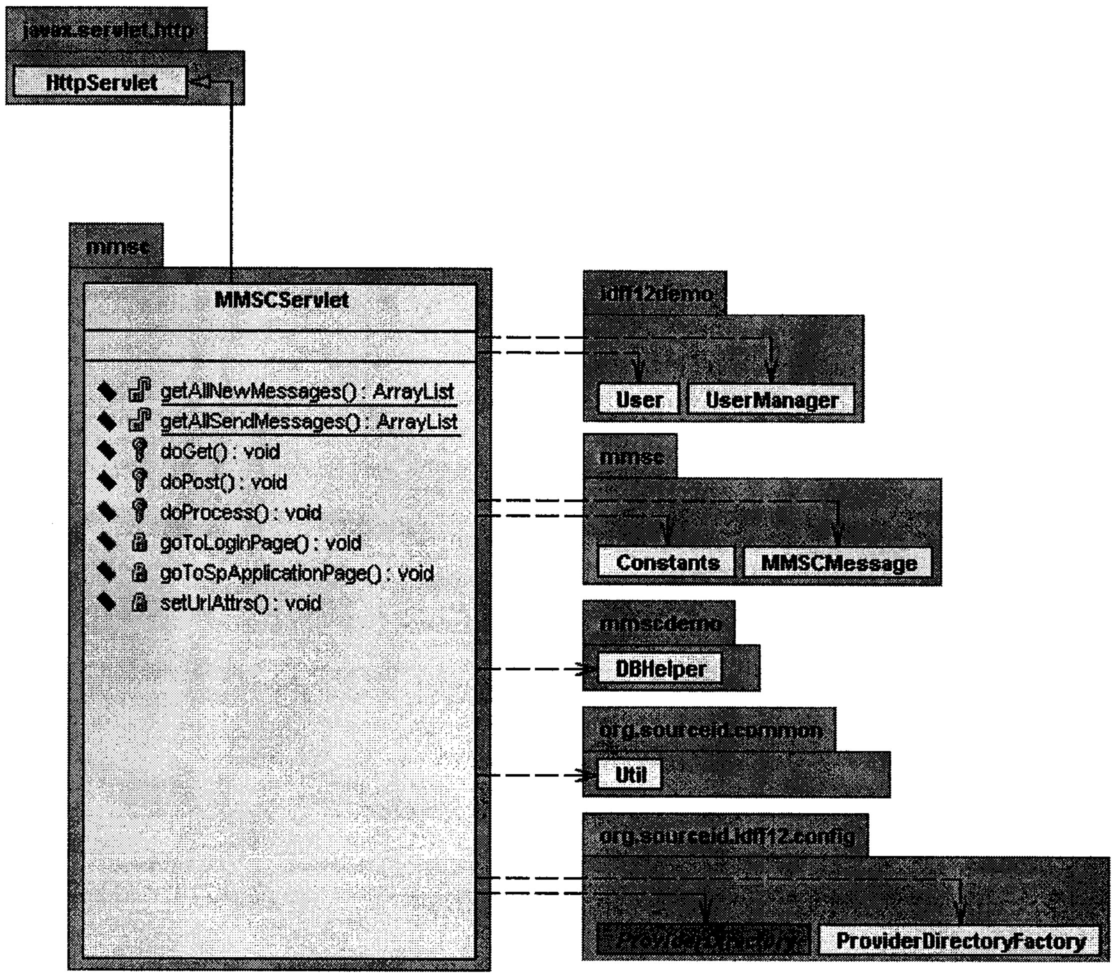


Figure 4.8 The servlet "MMSCServlet.class"

The servlet “resumeCenterService.class” is composed of one method “doGet” as shown in Figure 4.9. This servlet is to resume the workflow process. When the workflow directs the user to another node, it enters a “hanging” status. After the tasks at the other node are finished, this servlet triggers the workflow process to resume.

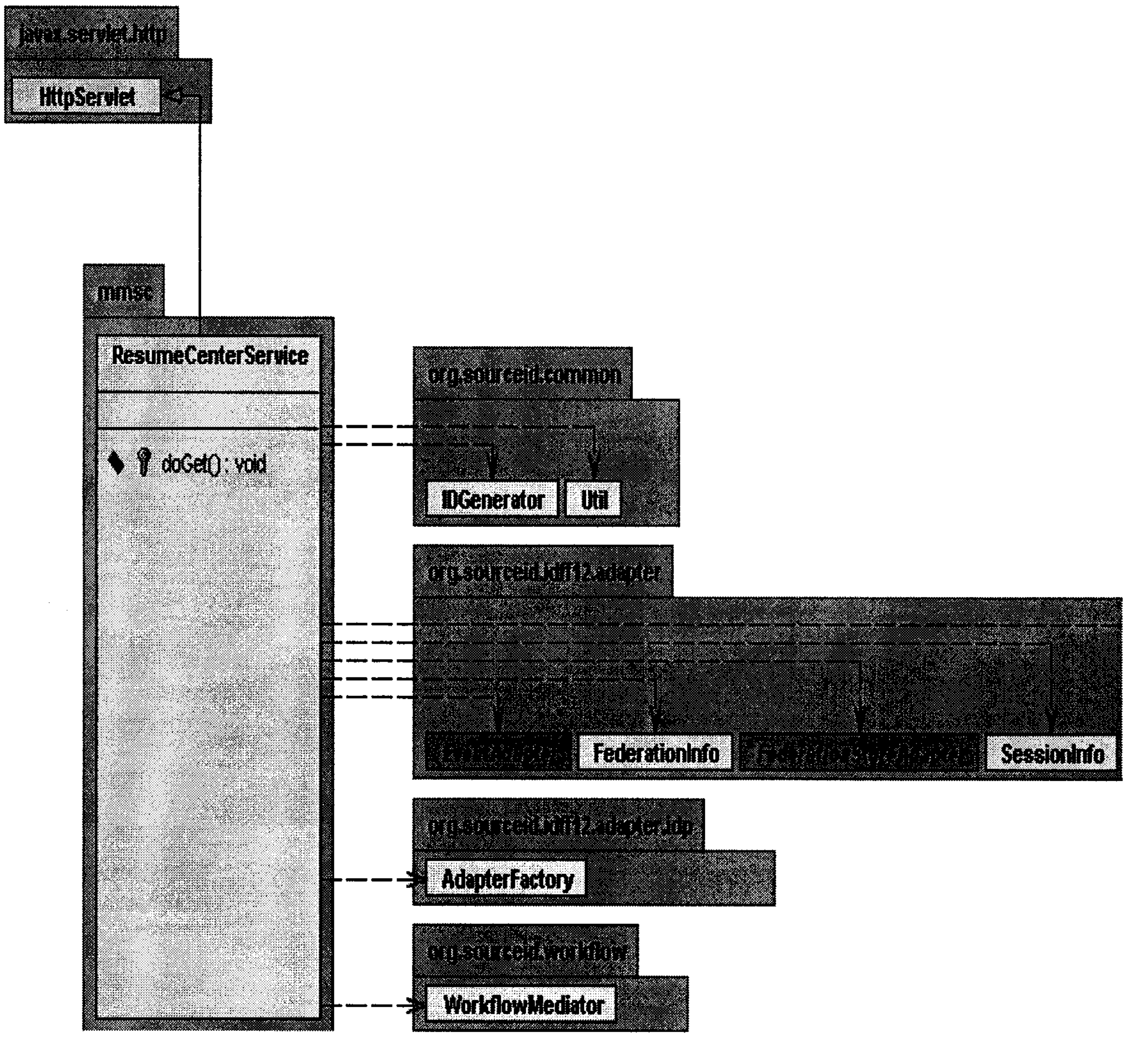


Figure 4.9 The servlet “ResumeCenterService.class’

The servlet “SOAPProuterServlet.class” is shown in Figure 4.10. It has two important methods:

DoPost: to build the SOAP message and post it to the service provider

GetHeaders: to retrieve the message header

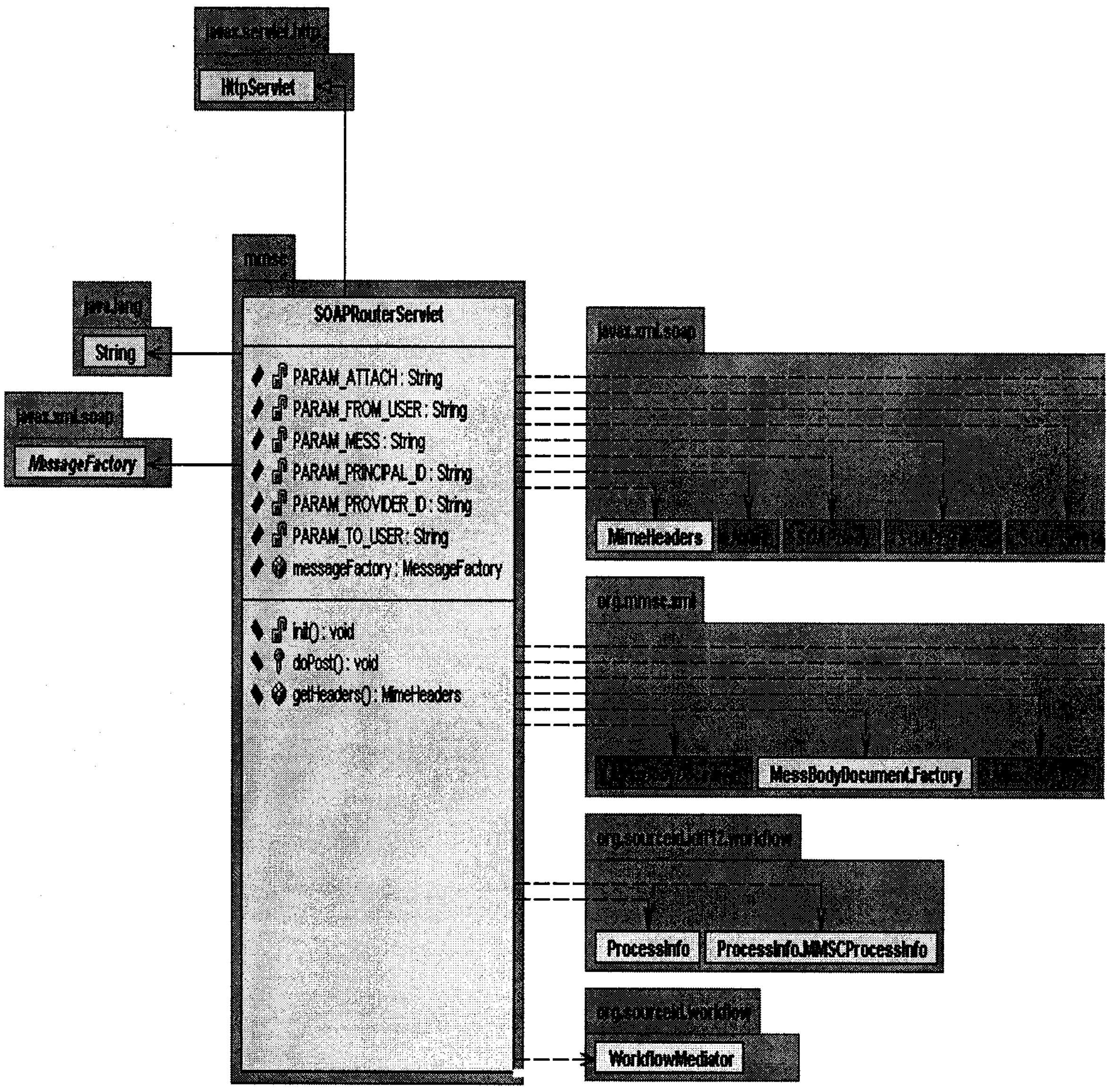


Figure 4.10 The servlet “SOAPRouterServlet.class”

4.4.4 Adapter

Through adapters, workflow activities fetch or update appropriate information, such as federation information and session information. Adapter classes are illustrated in Figures 4.11- 4.13.

The adapter "AuthnAdapterImpl.class" shown in Figure 4.11 processes the session information and get the authentication information. There are two methods in this class:

GetSessionInfo: to retrieve the session ID

GetAuthnInfo: to analyze the session information and retrieve the user's login status and principal ID.

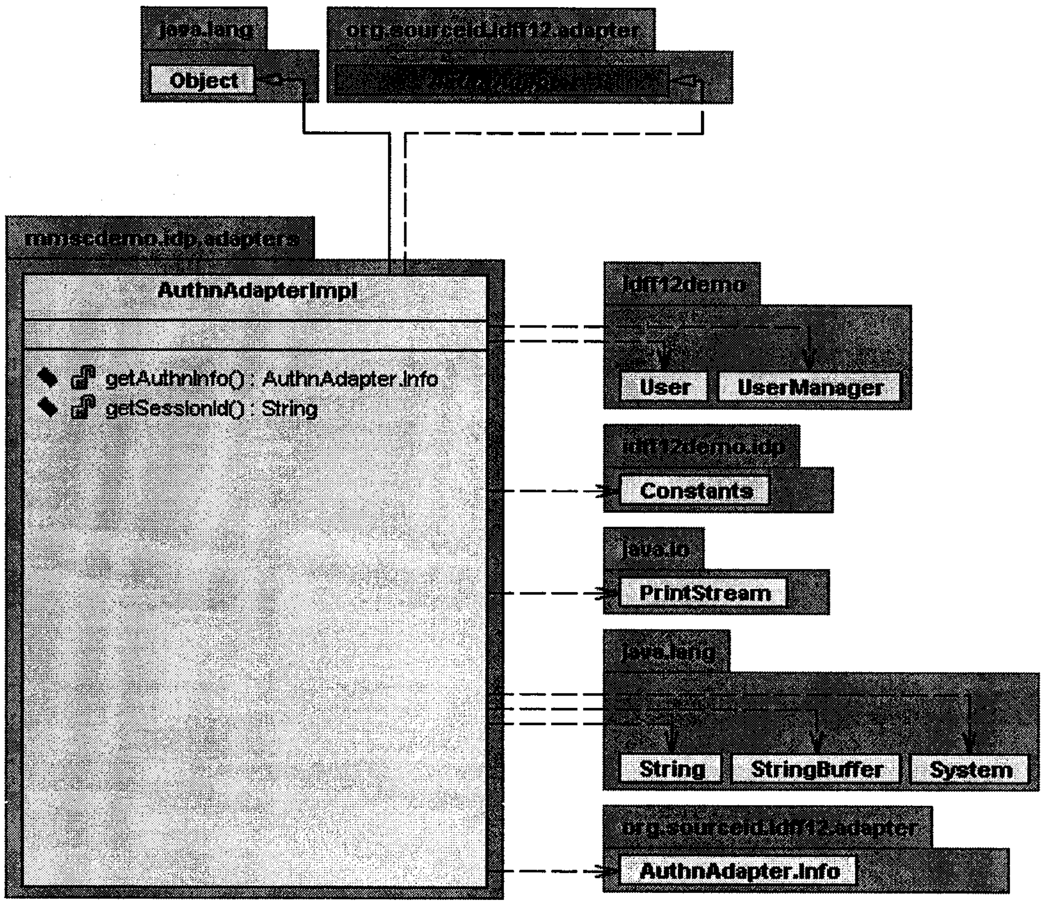


Figure 4.11 The adpter "AuthnAdapterImpl.class"

The adapter "MMSCEventAdapter.class" is shown illustrated in Figure 4.12. It adds or removes the provider ID upon the creation or termination of federation. There are two methods in this class, namely:

onFederationCreated: to add provider ID into the database upon federation created

onFederationTerminated: to remove the provider ID from the database upon federation terminated

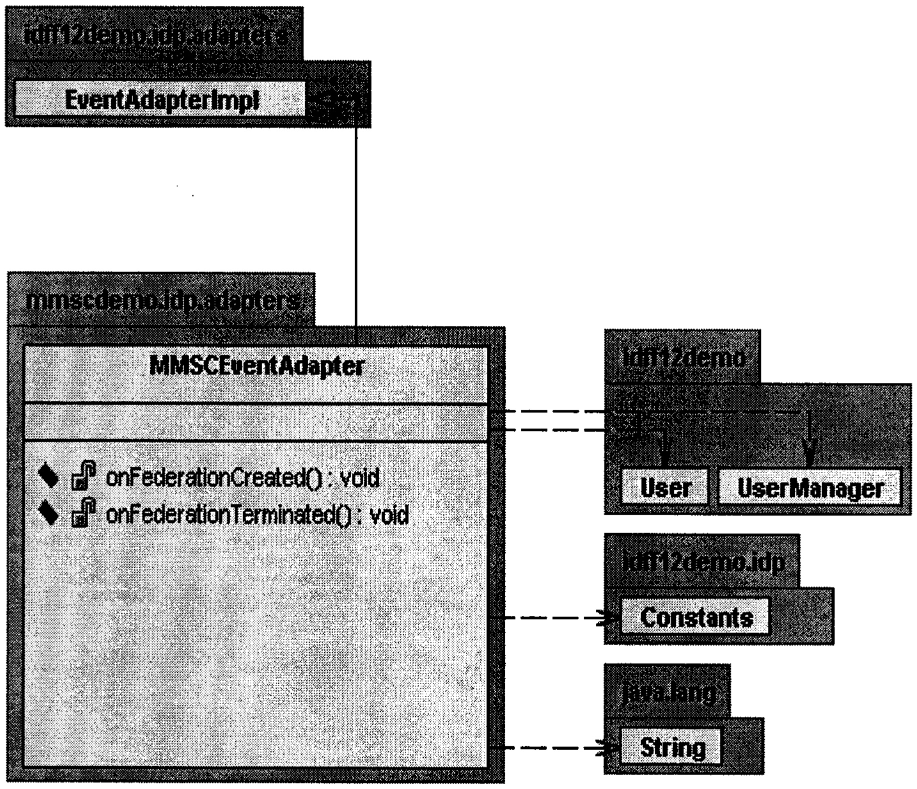


Figure 4.12 The adapter "MMSCEventAdapter.class"

The servlet "MMSCFederationStore.class" is illustrated in Figure 4.13. The method "isFederateWith" in this adapter queries whether an account is federated with the other service providers from the federation table in the database.

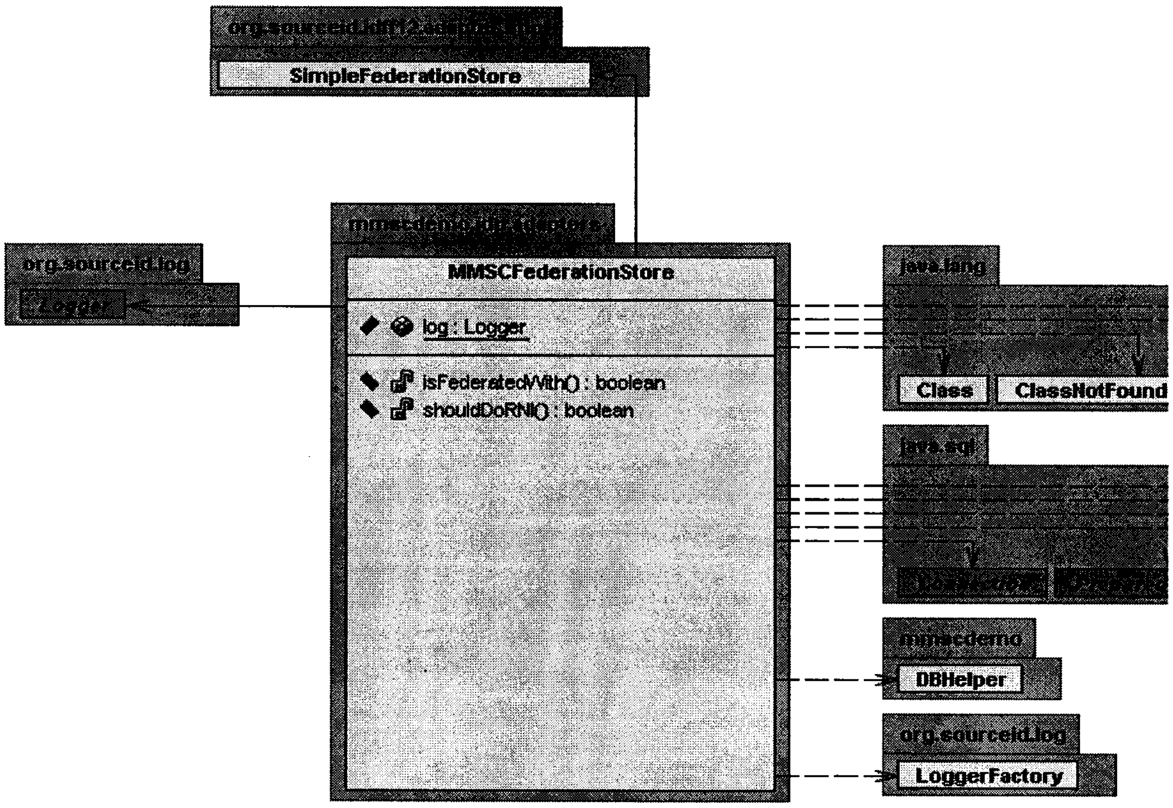


Figure 4.13 The adapter "MMSCFederationStore.class"

4.5 Configuration Files

In order that the Java packages, servlets, workflow, and the database can work together as a system, configuration files are needed. Some important configuration files are described below.

idff12.xpdl: This is the workflow configuration file. It defines the processes, activities, participants, relevant data, and so on in the workflow package.

web.xml: This file defines the mapping relationship between the servlets and Java classes, so that the web server can locate the physical class files by the servlet names.

mysql-ds.xml: This file tells the database name, database type, database location, communication protocol, and communication port. From this file, Java applications get the information as to how to communicate with the MySQL database.

sourceid-provider-directory.xml: This file tells how many providers we have, where they are located, what Liberty protocols they can use, and what profiles are created for those protocols.

sourceid-pworkers.xml: This file tells how to map the activities in the workflow to the methods in Java classes.

4.6 Database

SourceID Liberty 2.0 stores the account information in the memory of computers. In order to create a simulation system as close as possible to a production system, MySQL, a popular open source database is used. There are two databases in our simulation system – the MMSC and Kodak databases, namely "idp" and "sp".

4.6.1 Table Structure

In the "idp" database, the following tables are created:

usertbl: This table contains MSISDN, username, password, and other user information such as address and payment type, and is shown in Table 4.1.

fedtbl: This table specifies as to whether a subscriber is federated with his/her Kodak account (or accounts in other service providers if there are more service providers), and is shown in Table 4.2.

vasptbl: This table specifies as to how many VASPs the MMSC has, and which VASPs are in the circle of trust with MMSC, and is shown in Table 4.3.

Table 4.1 User table description

Column Name	Data Type	NULL?	Primary Key?
Msisdn	Varchar(15)	NOT NULL	Yes
Address	Varchar(250)		No
Is_prepaid	Char(1)		No
If_fed	Char(1)	NOT NULL	No
Password	Varchar(10)		No
username	Varchar(40)		No
multimedia	Char(1)		No

Table 4.2 MMSC Federation table description

Column Name	Data Type	NULL?	Primary Key?
Msisdn	Varchar(15)	NOT NULL	Yes
Vasp_id	Varchar(40)	NOT NULL	No
Idpname_id	Varchar(20)		No
Spname_id	Varchar(20)		No

Table 4.3 MMSC VASP table description

Column Name	Data Type	NULL?	Primary Key?
Vasp_name	Varchar(30)		No
Vasp_id	Varchar(40)	NOT NULL	Yes
Circle_of_trust	char(1)		No
Vasp_link	Varchar(255)		No

In the "sp" database, the following tables are created:

usertbl: This table contains MSISDN, username, password, and other user information, such as address and payment type. The structure of this table is the same as "idp.usertbl."

fedtbl: This table specifies as to whether a subscriber is federated with his/her Kodak account (or accounts in other service providers if there are more service providers). The structure of this table is the same as “idp.fedtbl”. Note that in some service providers, the MSISDN may not be registered. In this case, the “sp.fedtbl” does not contain the column of “MSISDN”. In stead, it contains the column of “username” and uses “idpname_id” and “spname_id” to map the MSISDN in MMSC side.

4.6.2 Database Interface and Application Classes

There are two interface classes, “DBHelper.class” and “JndiHelper.class”, that allow the system to communicate with the MySQL database.

DBHelper.class: Creates and closes the connection to a database server

JndiHelper.class: Accesses remotely the database server. It is invoked in “DBHelper” to search for the data source.

The application classes “User.class” and “UserManag.classe” manipulate the user databases through the database interface classes that were discussed above. These classes are described below:

User.class: This class creates the connection to the database by calling “DBHelper.class”; then queries and updates the table “fedtbl”. When the tasks are completed, it closes the connection via “DBHelper.class”.

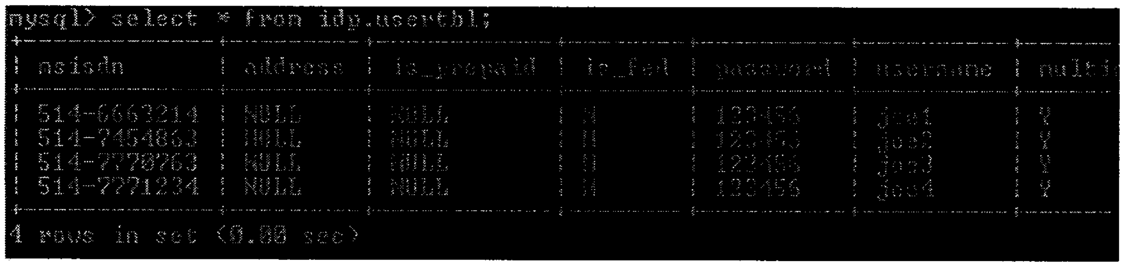
UserManager.class: This also calls “DBHelper.class” to create the database connection; then queries username and password from usertbl; finally, it closes the connection via “DBHelper.class”.

4.7 Test Results

In this section, we can provide some test results using the simulation model developed in this chapter.

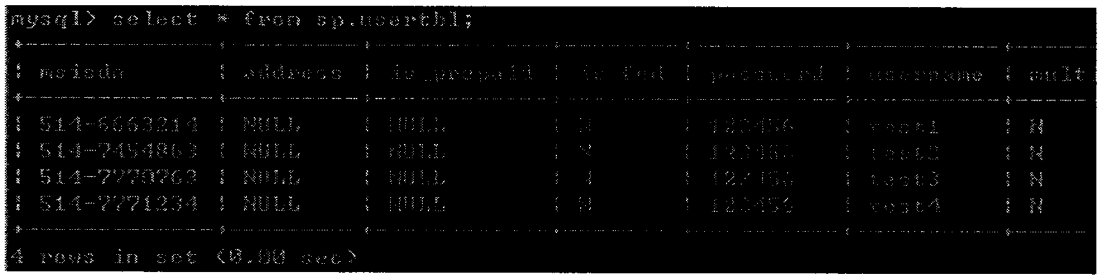
User tables before federation

Before federation, the user tables at MMSC and Kodak sides contain their own user entries. They can have a common field, MSISDN, as the foreign key. The example records for the MMSC user table is illustrated in Figure 4.14, and those for Kodak user table in Figure 4.15. "Joe1" on MMSC and "test1" on Kodak are actually the same user having an identical mobile number. At this moment, the federation tables don't contain any records yet. The empty results from the query on MMSC and Kodak sides are shown in Figure 4.16 and 4.17.



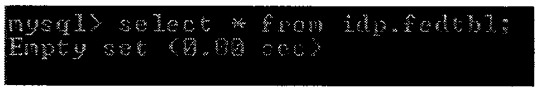
```
mysql> select * from idp.usertbl;
+--------------+---------+------------+--------+----------+----------+------
| msisdn       | address | is_prepaid | is_fed | password | username | multi
+--------------+---------+------------+--------+----------+----------+------
| 514-6663214 | NULL    | NULL       | N      | 123456   | joe1     | Y
| 514-7454863 | NULL    | NULL       | N      | 123456   | joe2     | Y
| 514-7770763 | NULL    | NULL       | N      | 123456   | joe3     | Y
| 514-7771234 | NULL    | NULL       | N      | 123456   | joe4     | Y
+--------------+---------+------------+--------+----------+----------+------
4 rows in set (0.00 sec)
```

Figure 4.14 MMSC user table before federation



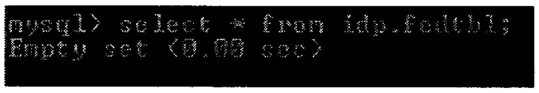
```
mysql> select * from sp.usertbl;
+--------------+---------+------------+--------+----------+----------+------
| msisdn       | address | is_prepaid | is_fed | password | username | mult
+--------------+---------+------------+--------+----------+----------+------
| 514-6663214 | NULL    | NULL       | N      | 123456   | test1    | N
| 514-7454863 | NULL    | NULL       | N      | 123456   | test2    | N
| 514-7770763 | NULL    | NULL       | N      | 123456   | test3    | N
| 514-7771234 | NULL    | NULL       | N      | 123456   | test4    | N
+--------------+---------+------------+--------+----------+----------+------
4 rows in set (0.00 sec)
```

Figure 4.15 Kodak user table before federation



```
mysql> select * from idp.fedtbl;
Empty set (0.00 sec)
```

Figure 4.16 MMSC federation table "fedtbl"

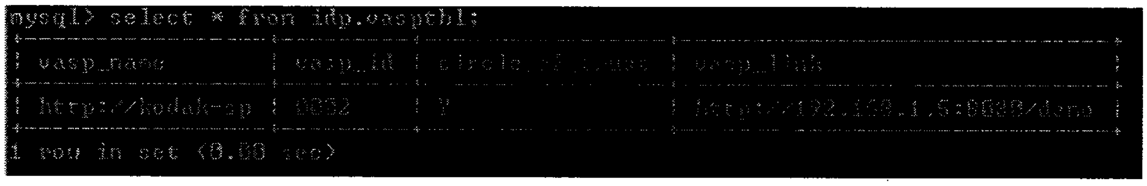


```
mysql> select * from idp.fedtbl;
Empty set (0.00 sec)
```

Figure 4.17 Kodak federation Table "fedtbl"

Mobile subscriber sending a message to Kodak

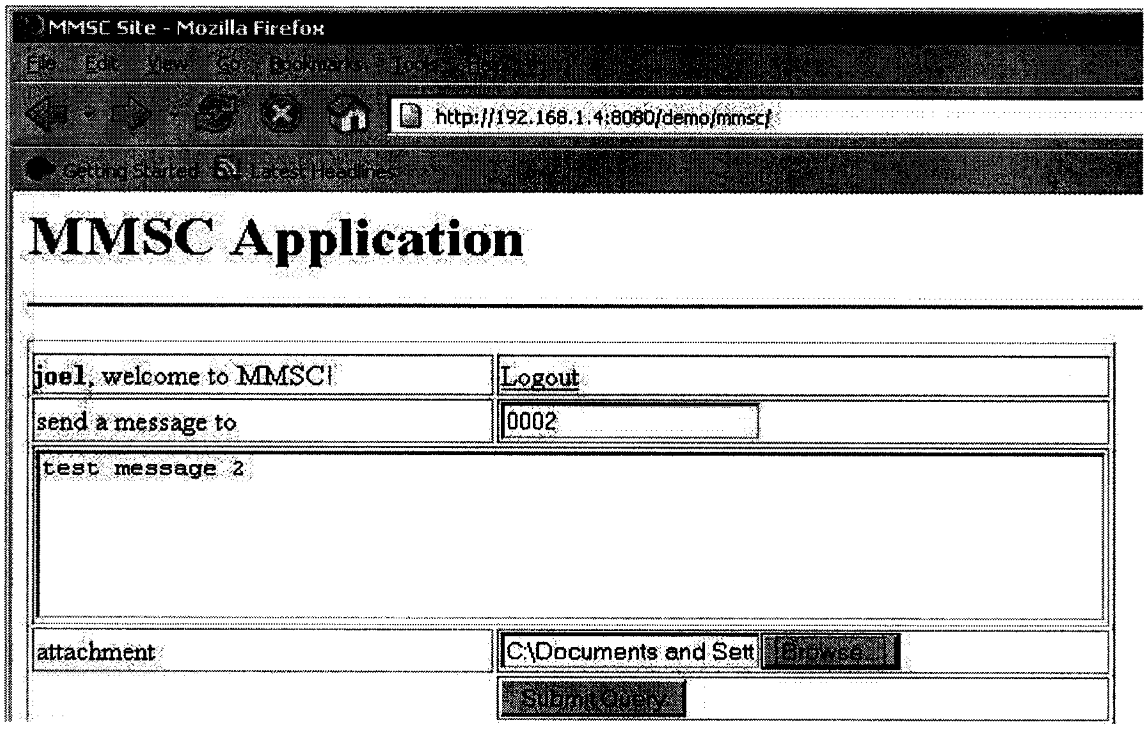
In our simulation system, the short code for Kodak is "0002". This is stored in VASP table at the MMSC side. The table structure and the records of this table is shown in Figure 4.20.



```
mysql> select * from idp.vasptbl;
+-----------------+---------+-----------------+-------------------------------+
| vasp_name       | vasp_id | circle_of_trust | vasp_link                     |
+-----------------+---------+-----------------+-------------------------------+
| http://kodak-sp | 0002    | Y               | http://192.168.1.5:8080/demo  |
+-----------------+---------+-----------------+-------------------------------+
1 row in set (0.00 sec)
```

Figure 4.18 Records of VASP table at MMSC side

In order to send a message (which can be a request for picture printing), "joe1" composes his message as shown in Figure 4.21.


MMSC Site - Mozilla Firefox
File Edit View Go Bookmarks Tools Help
http://192.168.1.4:8080/demo/mmsc/
Getting Started Latest Headlines

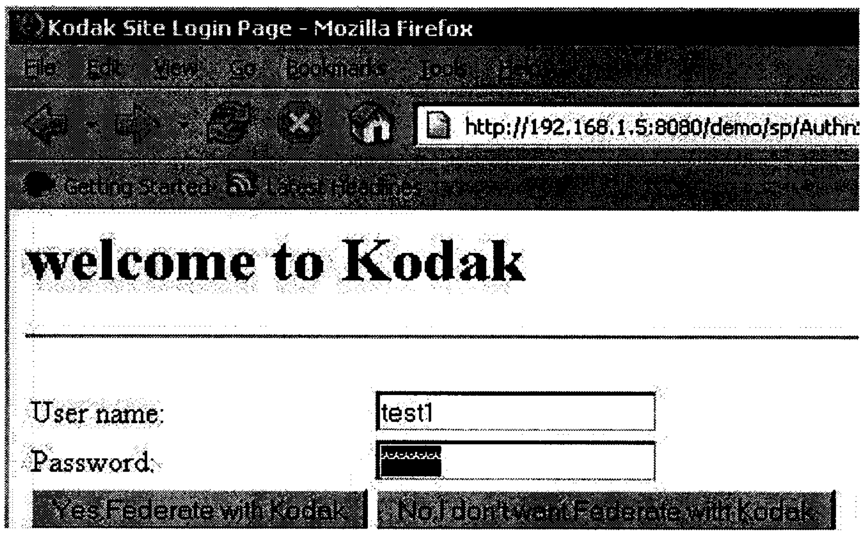
MMSC Application

joe1, welcome to MMSC!	Logout
send a message to	0002
test message 2	
attachment	C:\Documents and Sett Browse...
	Submit Query

Figure 4.19 Composition of joe1's message to be sent to Kodak

Pending confirmation of federation

If the sender is not federated with Kodak, MMSC asks if he/she would like to federate, as shown in Figure 4.20. The user can then reply yes or no to this query. The username and password are needed if the user chooses yes.


Kodak Site Login Page - Mozilla Firefox

http://192.168.1.5:8080/demo/sp/Authn

welcome to Kodak

User name: test1

Password: ******

Figure 4.20 The user's decision to federate with Koda or not

Federation of account and transmission of message

If the user agrees to federate or already federate with Kodak, a record as shown in Figure 4.21 and 4.22 is added in the federation tables at the MMSC and Kodak sites.

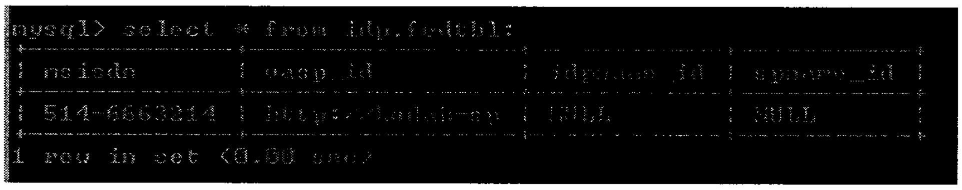

```
mysql> select * from idp.fedtbl;
+-------------+-----------------+------------+-----------+
| msisdn      | vasp_id         | idpname_id | spname_id |
+-------------+-----------------+------------+-----------+
| 514-6663214 | http://kodak-sp | NULL       | NULL      |
+-------------+-----------------+------------+-----------+
1 row in set (0.00 sec)
```

Figure 4.21 MMSC federation table after federation

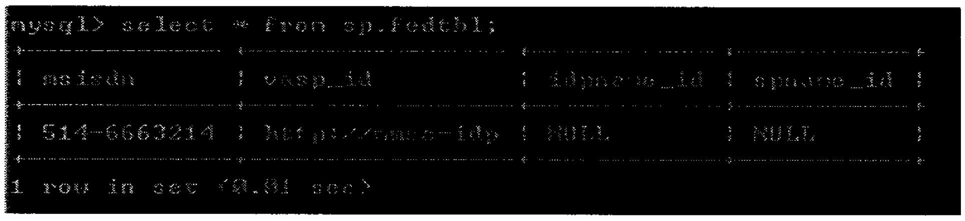
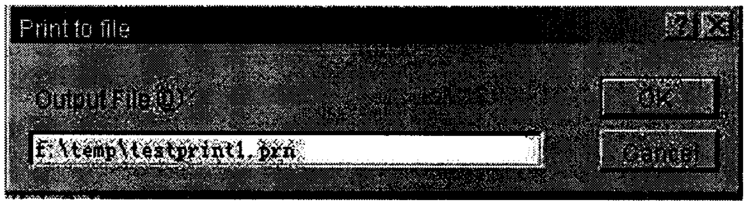

```
mysql> select * from sp.fedtbl;
+-------------+-----------------+------------+-----------+
| msisdn      | vasp_id         | idpname_id | spname_id |
+-------------+-----------------+------------+-----------+
| 514-6663214 | http://mmsc-idp | NULL       | NULL      |
+-------------+-----------------+------------+-----------+
1 row in set (0.01 sec)
```

Figure 4.22 Kodak federation table after federation

Then the client's message is sent to Kodak for printing. In our demo, this is illustrated by printing the picture to a file as shown in Figure 4.23.

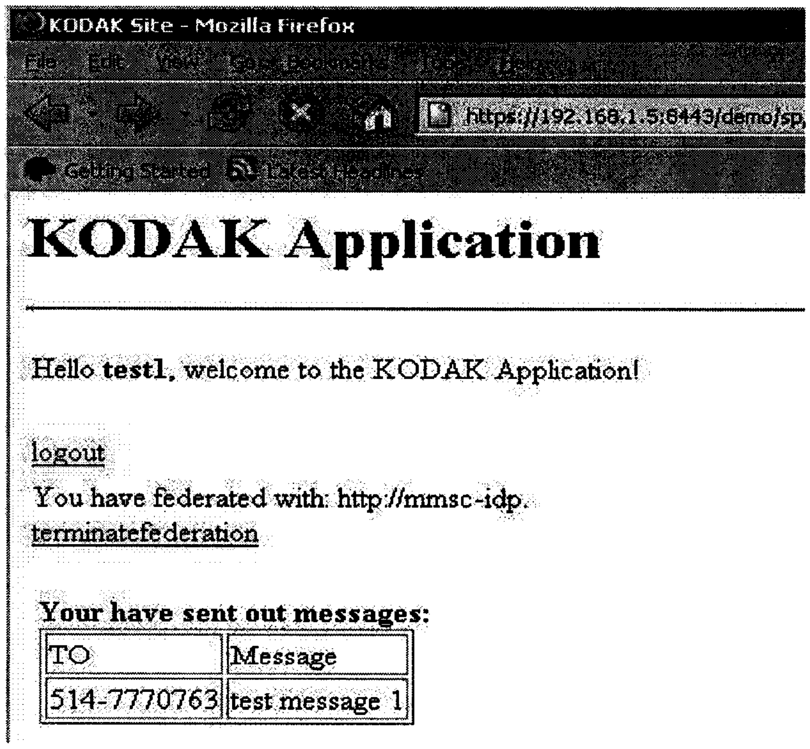


f:\temp\testprint1.prn

Figure 4.23 Printing of the attached picture to a file by Kodak

Accessing MMSC from Kodak

The user logs into the Kodak web page. Since his/her accounts in MMSC and Kodak have been federated, he/she can also check his/her multimedia messages which are stored in MMSC, as shown in Figure 4.26.


KODAK Site - Mozilla Firefox
File Edit View Go Bookmarks Tools Help
https://192.168.1.5:8443/demo/sp
Getting Started
Latest Headlines

KODAK Application

Hello test1, welcome to the KODAK Application!

logout

You have federated with: http://mmsc-idp.

terminatefederation

Your have sent out messages:

TO	Message
514-7770763	test message 1

Figure 4.24 Retrieval of user's multimedia message stored at MMSC via Kodak web page

4.8 Summary

In this chapter, a simulation model has been developed to demonstrate the implementation of the MMS-based system for integrated services proposed in Chapter 3. Two computers were connected to a LAN to simulate an MMSC and a service provider. The messages/requests sent across the LAN between two computers use the SOAP format. Incorporating the MySQL database into the simulation model has provided a substantial improvement over the memory-storage-based original SourceID Liberty 2.0 framework. The experimental results have shown that the goal of seamless and transparent access to service providers can be achieved using proposed scheme.

Chapter 5 Conclusion and Future Work

5.1 Conclusion

A prototype for implementation of *Liberty identity federation and single sign-on* in multimedia messaging service (MMS) has been proposed in this thesis. MMS offers subscribers various multimedia content services via a simple click-and-send operation. *Liberty single sign-on* and *federation* protocol enables users to login once, but access multiple service providers (SP). Hiding additional authentications provides users with a seamless environment between the multimedia messaging service center (MMSC) and service providers. This proposed prototype brings the following benefits to the end-users, operators, service/content providers, and developers.

End-users can have seamless access to the various services. No multiple authentications are needed. Since services are linked together, it becomes possible for the users to combine their bills into a single one from the MMSC. Then they do not need to pay dozens of bills from different service providers.

MMS operators will gain more value-added subscriptions, linking the customers to different types of content providers. Acting in the new role of *Identity provider* (IdP), MMS operators can establish long-term trust relationships between the subscribers and service/content providers.

The service/content providers can access the larger customer base. They will encounter fewer barriers in attempting to entice users to try new services. It becomes possible for the service/content providers to reduce their user database storage, since they can share some user profiles in the MMSC database.

The design proposed in this thesis makes it possible for developers to focus more on service business logic. *Liberty* SSO reduces the burden of service developers to re-implement algorithms for user administration and distributed access control, allowing for a wider portability and faster deployment of new services.

All these benefits will bring about an increased mobile e-commerce, increased multimedia content traffic, and of course, new revenue streams.

This thesis has also simulated the proposed prototype. This simulation has shown that a direct linkage between the MMSC and service providers is implementable. In the simulation system, simple object access protocol (SOAP) is used as the basic messaging protocol. Its most significant benefit is that it is not tied to any particular operation system or programming language. With SOAP, MMSC can exchange messages with any of the service providers. In our demonstration, the business process is automated and this improves control of the process, with far less management intervention, and very little chance for delays or misplaced work.

5.2 Scope for Future Work

This thesis has been an attempt at testing and demonstrating the advantages of using *Liberty single sign-on* in *multimedia messaging service* (MMS). Obviously, the next step is to implement this system commercially. There is always room to improve the system or make the system more general.

In the simulation system, a function box has been used to act as the MMSC system. In fact, MMS is a very complex system. Therefore, in order to add *Liberty* features to it, our job is not as easy as simply merging the class files from our simulation system into relay codes. There will have to be some major changes to be incorporated at the MMSC and some of these are given below:

Proxy-Relay: Relay needs to verify if the subscriber has federated his/her accounts with the service provider.

Message Store: Message store needs to retry delivery if the user is federated, or delete the message if he/she is not federated.

Subscriber Database: The subscriber table should contain a field indicating whether this subscriber has federated his/her account with other service providers. The record in the federation table should show which of the providers he/she is federated with. The federated providers may number more than one.

On service/content provider side, servlets are needed to receive and respond to the request coming from the MMSC.

Appendix

The computer programs have been developed using Java language and run on two computers that are connected to a LAN to demonstrate the implementation of the MMS-based system for integrated services. The programs are developed based on SourceID Liberty 2.0 framework, using JBoss as the web server, MySQL as the database, SOAP as the message format, and SAAJ API for SOAP attachment. The simulation programs, web server software, database software, and API packages are included in a CD-ROM with proper headings, as part of the thesis.

Reference

[1] "MMS—Building on the success of SMS", Ericsson Review, No. 3, 2001

[2] Changjie Wang, Fangguo Zhang and Yumin Wang, "Secure Web Transaction with Anonymous Mobile Agent", *Journal of Computer Science and Technology*, Vol. 18, No. 1, pp.84-89, 2003.

[3] Scott Stark, Marc Fleury and The JBoss Group, *"JBoss administration and development"*, Sams Publishing, 1999

[4] S. Landau and J. Hodges, "A Brief Introduction to Liberty", Sun Publications, Feb 2003

[5] Le Bodic and Gwenaël, *"Multimedia message service: an engineering approach to MMS"*, Wiley, 2003

[6] Scott B. Guthery and Mary J. Cronin "Developing MMS applications: multimedia messaging services for wireless networks" McGraw-Hill, 2003

[7] Daniel Ralph and Paul Graham, "MMS: technologies, usage, and business models" Wiley, 2004

[8] D. Livingston, "Advanced SOAP for Web Professionals", Prentice Hall, 2002

[9] Overview of SOAP
http://java.sun.com/developer/technicalArticles/xml/webservices/

[10] Bian R. Richardson, "An Architecture for Identity Management", Master Thesis, Massachusetts Institute of Techinology, Massachusetts, US, 2001

[11] Liberty Alliance Project, "Introduction to the Liberty Alliance Identity Architecture", Revision 1.0, Mar 2003

[12] S. Cantor and J. Kemp, "Liberty ID-FF Protocols and Schema Specification", Version 1.2, Liberty Alliance Project, IEEE-ISTO, 2004

[13] Scott Cantor and John Camp, “Liberty ID-FF Bindings and Profiles Specification”, Liberty Alliance Project, IEEE-ISTO, 2003

[14] Davis, Peter, eds. "Liberty Metadata Description and Discovery Specification," Version 1.0-errata-v2.0, Liberty Alliance Project, Sep 2004

[15] Maler, Eve, Mishra, Prateek, Philpott, Rob, eds. "Assertions and Protocol for the OASIS Security Assertion Markup Language (SAML) V1.1," OASIS Committee Specification, version 1.1, Organization for the Advancement of Structured Information Standards, May 2003

[16] A.Frier, P. Karlton, and P. Kocher, “The SSL Protocol Version 3.0”, RFC2616, *Hypertext Transfer Protocol -- HTTP/1.1*, June 1999

[17] Dierks, T., Allen, C., eds. (January 1999). "The TLS Protocol," Version 1.0 RFC 2246, Internet Engineering Task Force, Jan 1999

[18] “Digital Identity Basis”, PingIdentity
http://www.sourceid.org

[19] M. Lu, A. Lao, and Y. Lu, “Reference Architecture of Internet Single Sign-On”, Master Thesis, University of Waterloo, Waterloo, Ontario, Canada, Mar 2003

[20] “Workflow Management Coalition Terminology & Glossary”, The Workflow Management Coalition, Issue 3.0, Feb 1999
http://www.wfmc.org/standards/docs/TC-1011_term_glossary_v3.pdf

[21] C. Plesums, “Introduction to Workflow”, The Workflow Management Coalition, Jan 2005

[22] N. Stefanovic, S. Bojanic, and V. Puskas, “JaWE Tutorial - Java Workflow Editor”, Aug 2004
http://jawe.objectweb.org/doc/1.2/Tutorial/index.html

[23] Roberta Norin, “The Workflow Management Coalition Specification”, The Workflow Management Coalition, Version 1.0, Oct 2002

[24] SourceID Project, "SourceID Liberty 2.0 Beta User's Guide", Ping Identity Corporation, 2004

[25] J. Hodges and T. Wason, *"Liberty ID-FF Architecture Overview"*, Liberty Alliance Project, IEEE-ISTO, 2004,

[26] Scott Stark, Marc Fleury and The JBoss Group, *"JBoss administration and development"*, Sams Publishing, 1999

[27] Stevens Perdita, *"Using UML: software engineering with objects and components",* Addison-Wesley, 1999